Christian-Albrechts-University Kiel
Department of Computer Science
Software Engineering Group

Diploma Thesis

Migration of Software Systems to Platform as a Service based Cloud Environments

Sören Fenner (`sfe@informatik.uni-kiel.de`)

October 15, 2011

Advised by: Prof. Dr. Wilhelm Hasselbring
M.Sc. Sören Frey

Abstract

Cloud computing technologies that have been developed over the last few years establish new possibilities for both software as a service providers as well as for private and enterprise cloud users. These possibilities include, for example, scalable computing and storage resources that are serviced over the network, whereby services are commonly provided on a flexible pay-per-use basis. Therefore, the service consumer is released from adequate provisioning of resources, which often leads to an economic benefit.

However, software has to be designed for supporting the specific cloud technologies, in order to profit from the mentioned advantages. Thus, existing software systems either have to be redeveloped from scratch or adjusted to conform the cloud environment through a software migration. The migration of software from one environment to another implicates two major challenges: the migrated system has to both exploit the potential advantages of the target environment as well as meet its specification.

The latter issue is investigated in this thesis following the model-based CloudMIG approach. This approach uses models of both the application that is intended for migrating to the cloud as well as the target cloud environment. Based upon these models, CloudMIG identifies elements of the application that conflict with technical restrictions of the target environment, referred to as constraint violations.

This thesis presents an analysis of the violation detection capabilities of CloudMIG by investigating an open-source application for a migration to a platform as a service based cloud environment and verifying the detected violations by means of manual inspection. The results of this analysis indicate that CloudMIG's validation mechanism offers high detection precision and promises to be a useful approach for supporting reengineers in migrating software systems. In addition, an evaluation of the model extraction process, which is an essential component of the CloudMIG approach, is addressed as a minor goal. This evaluation shows that the proposed tool for model extraction has several drawbacks and is not completely reliable for CloudMIG's purposes. Further, an inspection of major issues regarding the manual migration substantiates that software migration to the cloud requires a high level of effort and constitutes a serious challenge from a reengineer's point of view.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

Software systems in general are subject to continuous evolution. This arises for various reasons, for example changing demands for functionality and increasing code complexity. Lehman (1979-1980) famously declared the *Laws of Program Evolution* as intrinsic factors for software evolution, as identified by his studies. This evolutionary process affects the operation as well as the quality and maintainability of software systems, especially in a long-term perspective.

Sometimes it is appropriate or even necessary to change components of the software, for instance, to support new features or leverage a beneficial performance gain. Changes to the involved personnel or to software or hardware in the systems' environment may also be factors leading to the demand for software reengineering activities (see Section 2.2). An example of this could be losing technical experts or the decreasing availability of special hardware components. In general scenarios the reengineering of some parts of the system is often sufficient, while leaving wide areas of the code untouched. This process is often iterated frequently, so that the system is continually modified over the years it exists.

Under certain conditions, especially if porting the existing system to a different hardware or software environment is the most influential factor in a decision to reengineer, the migration of a software system is generally a reasonable approach to attaining the desired aims.

Thus, major challenges of software migration are based on the fact that the migrated application's environment changes. For example, software libraries differ, management and access of data varies, and network configurations or communication protocols change. Therefore, adjustments regarding the considered architecture often have to be made in order to allow the application to execute correctly in the new context.

In the recent past new ways of utilization for computing resources, such as flexible allocation on a pay-per-use basis, have emerged and now constitute the popular concept of cloud computing. A variety of services is covered by cloud computing, hence it can be divided into different service models (see Section 2.1). Platform as a Service (PaaS), represented by Google's App Engine (GAE) for Java, is a particular service model that will be focused on in this thesis.

Based upon the flexibility of cloud services and business models provided, these possibilities open up both for private users as well as for large public or enterprise software consumers. Newly developed applications allow for the integration of these technologies by incorporating them in the design phase. Running existing software systems in a cloud environment to exploit the offered advantages is an important aim arising today.

The migration procedure often requires an enormous reengineering effort. Hence, it is an interesting research field, offering potential for improvements both from a technical and an economical point of view.

Resulting from recent work of the Software Engineering Group at the University of Kiel, the model-based approach *CloudMIG* (see Section 2.3) will act as a major foundation during this work. CloudMIG proposes a specific model representation for both the target cloud environment, named Cloud Environment Model (CEM) (instances of the CEM that represent a specific cloud environment are referred to as cloud profiles), as well as for the software considered as the migration candidate. Cloud Environment Constraints (CECs) are essential elements of the cloud profile representing technical limitations of the target environment that constitute potential obstacles regarding the migration process. For example, an application is not allowed to write to the local file system in Google's App Engine, which is represented by a corresponding CEC instance in the GAE cloud profile. As a fundamental component, CloudMIG provides a constraint violation detection mechanism that identifies existing CECs by static analyses on prior obtained system models.

This thesis will address challenges concerning the migration of an enterprise software system to a platform as a service-based cloud environment. In this context, especially the violation detection as a major component of CloudMIG will be analyzed. Therefore, an application will be investigated for CEC violations at first and the results will be verified by manual inspection afterwards to evaluate CloudMIG's validation quality. Although based upon a single application, which is not generally regarded as representative, a quantitative analysis is performed in order to get an impression of the number, type, and distribution of the occurred violations. Further, the model extraction process, which is an important prerequisite of the CloudMIG approach, is conducted and examined for vulnerabilities and improvement potential.

1.2 Goals

This thesis will address several goals pertaining to the migration of software systems to PaaS-based cloud environments. The major goals will be introduced in the following.

1.2.1 Evaluation of the CloudMIG CEC Violation Detection Mechanism

A major goal of this thesis will be the structured comparison of constraint violations that are statically determined by a violation detection pursuing the CloudMIG approach (see Section 2.3) and actually verified violations by manually inspecting the indicated violations. An important intention of CloudMIG is to establish a universal library of CEC validators to facilitate automatic violation detection capabilities. In order to evaluate the current efficiency of CloudMIG's violation detection under realistic circumstances, an open-source Java web application is used for a migration case study.

CloudMIG Xpress (see Section 2.4.8) is a prototype tool that includes the violation detection component of the CloudMIG approach as an extensible framework of constraint validators. In order to test the feasibility and to support the development, CloudMIG Xpress is utilized for the violation detection in this work. In this context, expecting the detection component of CloudMIG Xpress not to be completely perfect, some subsidiary improvements are integrated when a reusable solution seems to be indicated.

1.2.2 Migration to PaaS: An Inspection of Significant Migration Challenges

Besides the theoretical investigation of migrating software to the cloud, especially following the CloudMIG approach in this case, this thesis serves as a case study that aims for obtaining findings about the major challenges of migration from an reengineer's point of view.

Therefore, this thesis involves the manual realization of migration steps to a reasonable effort as well as the identification of significant issues that emerged. Chapter 5 points out several significantly challenging problems that occurred regarding this specific manual migration along with their corresponding reasons and potential answers. Moreover, important motives for executing a manual migration are discussed.

1.2.3 Analysis of the KDM Extraction Process

An essential component of the CloudMIG approach is the static analysis of the model of the designated cloud environment in conjunction with the architectural system model of the application considered for migration. The latter model unfolds in an extensive structure corresponding to the level of complexity of the considered application. Therefore, an automated generation of an appropriate model is favoured over manual treatment. MoDisco (see Section 2.4.7) is such a tool supporting the extraction of models that are based on Knowledge Discovery Meta-Models (KDM) (see Section 2.4.6), the fundamental meta-model utilized by CloudMIG, from a given software. MoDisco has recently been applied successfully by Frey and Hasselbring (2011) to extract models for different applications. Provoked by critical difficulties that were revealed, for example, during the work of Pascal Löffler in his bachelor thesis completed at the Software Engineering Group, University of Kiel, a further investigation of MoDisco's capabilities regarding the extraction of large-sized KDM models is carried out in this thesis (see Section 3.2.3).

1.3 Document Structure

The document is structured as follows. Chapter 2 introduces the foundations of this work. Starting with a definition of cloud computing, a summary of software migration and its justifications is given, and the CloudMIG approach, as a fundamental basis of this thesis is described. The foundations are completed with a description of involved technologies.

The definition and construction of relevant models is provided in Chapter 3. Headed by a description of the two essential models utilized by the violation detection, the definition of the cloud environment model and the extraction of the architectural model for the considered applications is examined in more detail. Moreover, the performance of the model extraction process is analyzed.

Chapter 4 contains the identification of constraint violations. First, an overview of relevant violations is given before CloudMIG's violation detection mechanism is explained in detail. Then, the implementation of a new model import feature in CloudMIG Xpress is presented, followed by a report and brief analysis of the application of CloudMIG Xpress' violation detection on JForum.

An inspection of significant challenges regarding the execution of the manual migration is described in Chapter 5.

Chapter 6 presents the detailed analysis of the detected violations. First, an overview and quantitative examination of violations is given. With the help of a CEC violation inspection template, which is defined before, the manual inspection of detected violations is conducted and presented here. On the basis of this investigation, a critical review of CloudMIG's violation detection capabilities is presented.

The following Chapter 7 gives an overview of related work.

Chapter 8 contains the conclusion of this thesis. It comprises a summary as well as a discussion of the results and proposes future work.

The appendix contains additional resources which are referenced by the according sections of this document.

2 Foundations

2.1 Cloud Computing

Cloud computing can be described as a framework for provisioning resources as services that are supplied over the network. Prevalent types of resources include for example storage and processing capacities (e.g., Amazon Web Services), security, and backup services or software development environments with cloud-optimized APIs (e.g., Google App Engine, MS Azure).

Service computing, such as commonly established web applications (e.g., Google Docs) is expanding in both complexity and diversity. It has already taken possession of the enterprise sector (e.g., Salesforce.com) and is successively entering other huge domains, such as entire desktop environments and operating systems (e.g., eyeOS) (see Hayes (2008)).

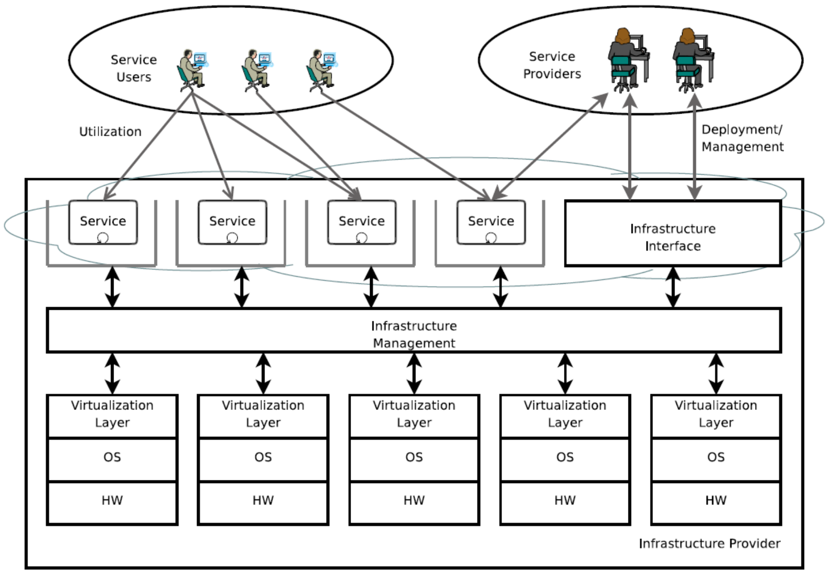


Figure 2.1: Overview of cloud computing actors and roles (Vaquero et al., 2009).

Different actors that are involved in the cloud computing system have to be contemplated in this context. Vaquero et al. (2009) distinguish between service providers (usually developers), service users (or consumers), and the infrastructure providers as the main actors represented by the involved participants of the cloud computing layer being considered (see Figure 2.1). A similar classification is described by Armbrust et al. (2009), who emphasize the recursive roles of service providers who might be service users of other (third-party) cloud services at the same time. This relationship can be in a product-independent way or with direct relationship to the offered service, for example if the used service is incorporated as a part of the offered service.

Cloud computing is a very young scientific research field and plenty of definitions have recently been proposed by different authors. Consolidated standards and definitions will most likely be established in the next few years. An accepted definition by Foster et al. (2009) says that cloud computing is:

> *"A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet."*

The remainder of this section will give an overview of the Cloud Computing Model corresponding to the definition of cloud computing that has been established by the NIST[1] (see Figure 2.2).

A first distinction can be made on the basis of the deployment model. The deployment model characterizes how and by whom the service is hosted, provided, managed, and used.

(i) *Private Cloud:* The cloud infrastructure is designed exclusively for the cloud user. It might be hosted and operated by the user or offered as a service by a cloud provider.

(ii) *Community Cloud:* Similar to the private cloud, this deployment is restricted to a specific group of users from different organizations, who share similar demands and interests. It is either hosted and operated by the community itself or by a third-party provider.

(iii) *Public Cloud:* This deployment model is aimed at the broader public. The infrastructure is commonly owned by the providing organization and sold to any user.

[1]National Institute of Standards and Technology, Agency of the U.S. Department of Commerce

(iv) *Hybrid Cloud:* A composition of multiple private, public, and community clouds is called a hybrid cloud. Generally all individual clouds are mutually independent from the users point of view but are often hosted on the same underlying infrastructure and operated by the same provider.

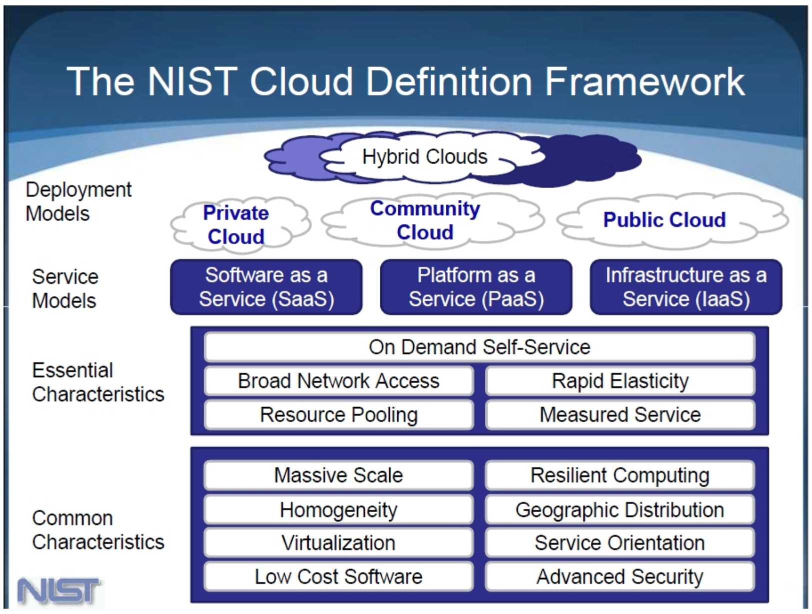


Figure 2.2: NIST cloud definition framework (Mell and Grance, 2009a).

The provided resources in the cloud computing context can be categorized according to service models (see Mell and Grance, 2009b, NIST) or layers of granularity (see Vaquero et al., 2009), which are presented in the following:

(i) *Software as a Service:* The finest granularity is represented by software as a service (*SaaS*), where a provided application is running on a cloud infrastructure. It is usually accessed through a web browser. The consumer can generally use the service without having to install, maintain, or manage any software or hardware resources.

(ii) *Platform as a Service:* The underlying layer, called platform as a service (*PaaS*), facilitates a wider foundation for both development and deployment purposes. Those services incorporate a runtime environment including a broad software stack that provides more flexibility to users for arranging their customized solutions. Since this layer can be detached even from virtual hardware, no load-balancing or managing of computational resources is required in these systems from a cloud user's point of view.

(iii) *Infrastructure as a Service:* The computer hardware or infrastructure as a service (*IaaS*), for example delivered as virtualized machines or flexible packages of computing, storage, or network resources, forms the most low-level oriented layer of cloud computing and is often considered as the basic layer. The consumer has control over the provided resources (e.g., operating systems, storage, applications) without having to manage the underlying infrastructure.

(iv) *Hardware as a Service:* Although the physical hardware (*HaaS*) might be distinguished as an independent layer (see Rimal et al. (2009) or Youseff et al. (2008)) we will restrict our discussion to the former three with the focus on PaaS.

Regardless of the considered layer, the resource is serviced over a network connection. It causes little or even no additional hardware requirements for the cloud users, which establishes a high compatibility and attaches seamlessly to the majority of existing IT structures. One essential characteristic of cloud computing is the elasticity of service or resource allocation.

The problem of efficient resource provisioning is illustrated in Figure 2.3. The graph (a) shows the portion of excessive resource capacity (gray shaded area) that results in systems which have been designed to satisfy rare peak loads. If the capacity is adjusted to cover an average of the resource demand (see graphs (b) and (c)) there will be periods of underprovisioning where increased response times or even worse problems (e.g., system malfunction and outages) might occur. Unlike physical equipment, the amount of "consumed" cloud services is flexible over time and therefore can be requested and billed on demand. This leads to an enormous economic benefit for cloud users who do not have to maintain overprovisioned resources.

Another advantage in this context is that no precise peak capacity predictions have to be made which would be required to align the resources according to the peak demands (see graph (a)). In most scenarios it might not even be possible to determine an upper boundary for the estimated capacity demand. Further, if a peak demand could be defined it would be temporally limited in most cases, and would require expensive analysis and calculations to

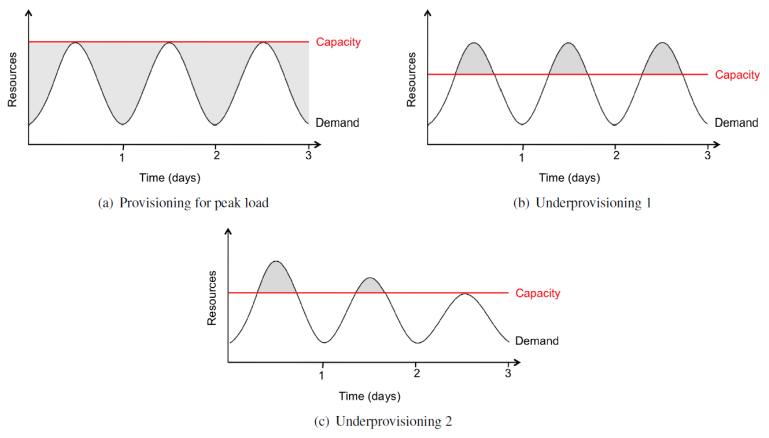


Figure 2.3: Provisioning of resources (Armbrust et al., 2009).

identify its size. Furthermore, customers can take advantage of the scale effect that cloud providers achieve through the operation of huge infrastructures. Another positive characteristic of cloud computing is the high reliability of services, assuming that professional cloud service providers apply superior arrangements, such as hardware redundancy or data backup.

2.2 Software Migration

There are several reasons for modernizing or migrating a legacy software system. Aspects, that are conventionally regarded as leading to the decision to modernize an application cover the diminishing support of programming languages, software libraries, protocols, or hardware platforms. Furthermore, a software system generally faces ongoing changes of functional demands. This leads to an increasing amount of code and continuously increasing size and complexity of the legacy application. Thus, a legacy system is usually subjected to consecutive adjustments and modifications over time. Depending on their shape and extent, these changes are assigned to different categories of system evolution (see Seacord et al. (2003)):

- **Maintenance:** Maintenance describes minor modifications to the software system, such as implementing bug fixes or small functional enhancements. It is considered as an iterative progression without changing conceptual elements of the software.

- **Modernization:** Modernization actions affect more extensive parts of the system. For example, if system restructuring or major functional enhancements are required without significantly changing the residual system. System modernization is sub-divided in terms of the depth of required knowledge about the legacy system concerned:
  - **White-Box Modernization:** Modernizations affecting structural changes of the application that require detailed knowledge of the system internals are considered *White-Box Modernizations*. In many cases, reverse engineering actions are applied to extract the required information.
  - **Black-Box Modernization:** If only knowledge of the external interfaces of the system is required for the modernization, it is considered *Black-Box Modernization*. This kind of modernization is commonly based upon *wrapping*, which encapsulates a specific part of the legacy application, generally by attaching the wrapper around the components' interfaces. The wrapped components are usually left untouched but supplied with a new interface that conforms to adopted external requirements.
- **Replacement:** If a legacy system suffers from significant deficiencies in serving business objectives or supporting technological demands (e.g, performance, compatibility, flexibility, security), that can not be resolved by *Maintenance* or *Modernization*, either due to technical or economical reasons, *Replacement* may be the only solution.

Software migration is a reengineering process that can be considered a specific form of software modernization. In comparison to other modernization activities, software migration aims to move the existing system to a different software or hardware environment while preserving the application's functionality and data instead of adding new business functionality. Reasons for software migration are mostly based upon advantages that the target environment provides compared to the old environment, for example increased hardware power, less operational costs, a more flexible environment, or better maintainability (see Seacord et al. (2003), Almonaies et al. (2010)).

The migration of legacy software systems has been of research interest for a long time. As different application areas and a wide spectrum of reasons for software migration exist, various approaches have been developed to cope with the specific issues.

Brodie and Stonebraker (1995) present an incremental migration technique, the *Chicken Little* methodology for migrating legacy systems step-by-step. This methodology differs from the *Cold Turkey*, also known as *Big Bang* strategy, that redevelops the new system from scratch and replaces the legacy system in a final step (Big Bang). A significant characteristic of the Chicken Little methodology is its requirement for data and information exchange between the legacy and target system via so-called *Gateways*, since both systems need to be operational and maintain consistent data until the migration is completed.

Bisbal et al. (1999) give an overview of existing research in legacy software migration, particularly regarding Chicken Little and the *Butterfly Methodology*. The latter is presented in Wu et al. (1997b) and Wu et al. (1997a). The Butterfly Methodology is an approach that especially addresses the migration of systems in mission-critical environments that require the parallel operation of the legacy and the target system during the migration until the new system replaces the old one in a last migration step with a minimized out-time. In comparison to the Chicken Little methodology, the Butterfly approach utilizes no *Gateways*, since it does not require the provision of continuous data consistency between the legacy and target system. Hence, it is called a *gateway-free* approach.

An approach to migrating integrated legacy systems to multi-tier architectures is presented by Hasselbring et al. (2004). They developed the *Dublo* pattern, named by the partial *DUplication of Business LOgic*. The DUBLO approach supports a smooth migration path that can be carried out step-by-step. This fact is based on the pattern design that allows both elements incorporated by the legacy code and the deployed target application server to provide (partially duplicated) business logic.

In the recent past, model-driven techniques were investigated increasingly for software migrating purposes. Fuhr et al. (2010) present a migration extension to IBM's Service-Oriented Modeling and Architecture (SOMA) that utilizes model-driven strategies to transfer legacy assets to Service Oriented Architectures. They provide a case study in which they migrate one functionality of the Java application *GanttProject* into web a service by using code analyses, service identification, and model transformation. This approach is at an early stage and does not yet provide universal means for migration, as indicated by the authors, for example, concerning the lack of support for languages other than Java. However, their method served successfully as a technical proof-of-concept.

Fleurey et al. (2007) present another migration approach situated in the context of model-driven engineering. The described migration process includes reverse engineering, transformation, and code generation based on building and transformation of models. Major goals of this approach aim for maximizing the automation as well as increasing the flexibility and reliability of the migration process. Their description is accompanied by a case study concerning the migration of a main-frame banking system to J2EE. This approach proposes a rather generic migration-supporting instrument and is regarded as being directly related to the CloudMIG approach considered in this thesis.

Today cloud computing establishes new motifs for software migration due to the aspects described in Section 2.1. Depending on the extent that the software will be affected by the migration, it has to be considered whether it is more feasible and appropriate to only transform selected parts or to re-engineer the whole program. If the latter situation occurs, it may be reasonable to reconsider rewriting the application from scratch.

One of the most challenging tasks in software migration is investigating the status quo of the legacy system. A precise understanding of functional dependencies, communication mechanisms, and storage structures for instance is obligatory to determining if a migration is both feasible and rational.

As software ages inevitably over time (see Parnas, 1994), especially large systems differ considerably from their original architecture and documentation. Therefore, an analysis of the system in question has to head the migration process. In many cases this analysis involves a reverse engineering activity. "Software reverse engineering aims to both identify the components and their interrelationships as well as giving a representation of the considered system at a higher level of abstraction." (see Chikofsky and II, 1990)

2.3 CloudMIG

Migration of applications to cloud environments as well as software migration in general are often very complex processes, where essential reengineering activities form an extensive part. CloudMIG is an approach designed to support software engineers at the reengineering process in a semi-automated manner (see Frey and Hasselbring, 2010). Following a model-based approach, CloudMIG aims at rearranging enterprise software systems to scalable and resource-efficient PaaS and IaaS-based applications.

The basic activities and components of CloudMIG are illustrated in Figure 2.4 and will be described in the following.

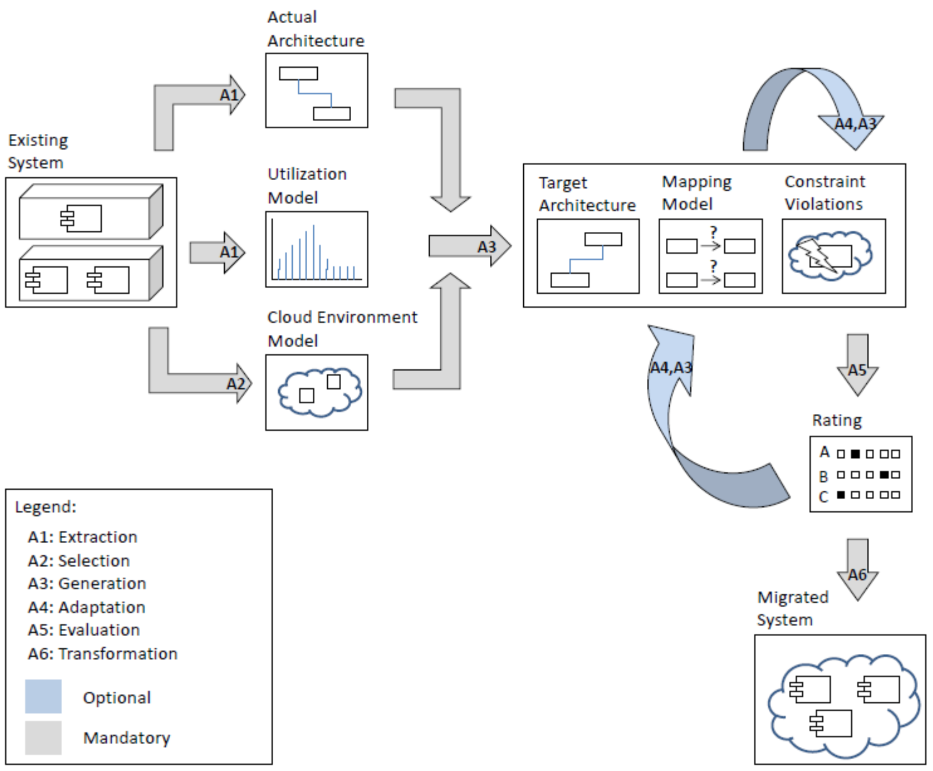


Figure 2.4: Activities and components of CloudMIG (Frey and Hasselbring, 2010).

(i) **A1 Extraction:** Models that represent the application's actual architecture and utilization are required for further steps, especially in activity *A3*. In *A1*, these models are extracted from the application.

(ii) **A2 Selection:** When the specific Cloud Environment Model (CEM) according to the considered target cloud platform has not been created before, for example if the cloud environment is investigated for the first time, it has to be created once at this point. Otherwise the corresponding CEM is selected here.

(iii) **A3 Generation:** During this activity, the actual architecture has to be translated to the target architecture. Therefore, all Cloud Environment Constraint (CEC) violations are detected by a constraint validation mechanism. Further, a mapping model is created that identifies unsuitable features from the actual architecture which have to be converted to proper elements of the target architecture.

(iv) **A4 Adaptation:** This activity allows manual adjustments of the target architecture to match specific requirements of the reengineer's concept.

(v) **A5 Evaluation:** The target architecture, provided by activities *A3* and *A4*, is evaluated by static and dynamic analysis. This evaluation activity combined with the preceding activities *A3* and *A4* can be carried out iteratively to refine the resulting target architecture.

(vi) **A6 Transformation:** This activity describes the manual transformation of the system from the generated target architecture to the cloud environment.

The models that result from activities *A1* (architectural model) and *A2* (cloud environment model) are of particular importance for this thesis, therefore they are further described in Section 3.2 and Section 3.1, respectively. A brief description is given in the following:

- **Architectural model:** The architectural model of the legacy system is required conceptually by CloudMIG to allow the model transformation process between the legacy and the target architecture. CloudMIG proposes the Knowledge Discovery Meta-Model (KDM) (see Section 2.4.6), a specification defined by the Object Management Group (OMG), as a model format for the applications' architectural information. The extraction of architectural information is achieved by running a software reverse engineering process on the considered application. CloudMIG proposes the tool MoDisco (see Section 2.4.7) for this activity.

- **Cloud Environment Model:** The Cloud Environment Model (CEM) represents the specific characteristics of the target cloud system. These characteristics are grouped to layered packages (see Figure 2.5). For example, the *core package* comprises basic elements and abstract elements as a basis for elements in other packages. One of the most relevant characteristics of a cloud system is the set of Cloud Environment Constraints (CEC) that imposes technical limitations to applications running in the target cloud environment. The *constraint package* defines these constraints for hosted applications. An excerpt of an CEM instance for Google App Engine is presented in Section 3.1. A more detailed view on the constraint package of CEM in terms of its violations is given in Section 4.1.

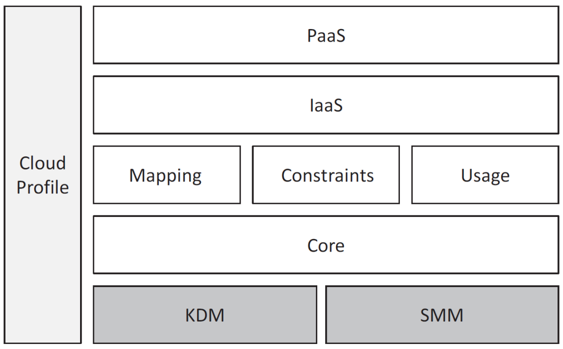


Figure 2.5: The packages of CEM (Frey and Hasselbring, 2010).

The CEM has to be instantiated once for each cloud environment. The model can be reused for subsequent investigations on the same cloud environment. For this purpose CloudMIG will provide a repository of existing CEMs in the future.

Besides the architectural model of the legacy system and the defined cloud profile, which are main components for the functional mapping of the system towards the target architecture, CloudMIG supports the possibility for taking a utilization model of the legacy system into account. CloudMIG proposes the extraction of the legacy system's utilization model according to OMG's Structured Metrics Meta-Model (SMM). Information about the utilization of the system's resources is used to design the target architecture in order to exploit general performance potentials and assure that the migrated system can harness cloud elasticity. The extraction of a utilization model is not intended here.

The model-based CloudMIG approach especially addresses the migration of enterprise software systems to IaaS and PaaS-based clouds from a SaaS provider's point of view. It utilizes model-driven reengineering techniques and rule-based heuristics to support semi-automatic generation of the target system architecture. One major activity of the CloudMIG approach is the conformance analysis of the considered software regarding the potential target cloud environment. This analysis is provided by an extensible framework of validators that examine model representations of the considered software system for violations concerning the target environment. A case study that applies the CloudMIG approach to investigate five open-source applications for a migration to Google App Engine for Java is conducted as a proof-of-concept in Frey and Hasselbring (2011). The experiments show the feasibility of the approach by introducing the prototype tool CloudMIG Xpress (see Section 2.4.8) with an explicit focus on violation detection.

2.4 Involved Technologies

2.4.1 Eucalyptus

Eucalyptus is an open-source infrastructure software for building and managing cloud computing environments within computer clusters. The name Eucalyptus is an acronym for "Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems." Originating from a research project (see Nurmi et al., 2009) at the University of California, Santa Barbara in 2008, the project is now maintained by *Eucalyptus Systems, Inc.*, founded by the original authors of Eucalyptus to handle the incipient rapid growth of the product and its community.

Eucalyptus operates on most common Linux distributions such as Ubuntu, Red Hat Enterprise Linux (RHEL), CentOS, SUSE Linux Enterprise Server (SLES), OpenSUSE, Debian, and Fedora. The standard interface is entirely compatible with Amazon Web Services (AWS) such as Elastic Computing Cloud (EC2), Elastic Block Storage (EBS), and Simple Storage Service (S3).

Figure 2.6 illustrates the topology of a typical Eucalyptus cloud environment and the relationships between its components. Eucalyptus consists of four main components, each acting as a stand-alone web service. In the following, these components are described in hierarchical order, beginning at the lowermost:

(i) **Node Controller (NC)** run on every node that hosts virtual machine (VM) instances. A node controller controls the execution, inspection and, termination of instances.

(ii) **Cluster Controller (CC)** provide scheduling of VM executions over available node controllers as well as network management facilities.

(iii) **Storage Controller (Walrus)** implement Amazon's S3 interface for storing and accessing VM images and user data.

(iv) **Cloud Controller (CLC)** control the underlying components and provides resource information to cloud users and administrators. Further, a cloud controller handles authentication and exposes system management tools.

Eucalyptus is interchangeable with the Amazon environment, allowing customers to perform development and testing actions on their private cloud prior to actually deploying

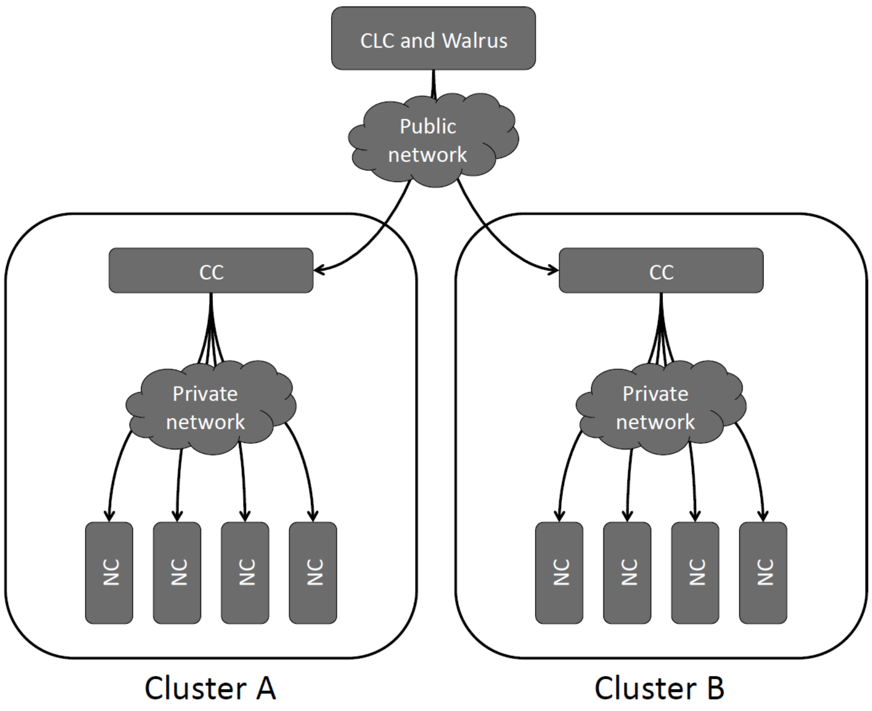


Figure 2.6: Design of a typical Eucalyptus cloud environment (Nurmi et al., 2009).

their solutions into the Amazon cloud. Currently up-to-date hypervisors like Xen or KVM are supported by default to act as underlying virtualization technology for virtual machine instances.

Besides an open-source edition that includes the core elements of the system, an enterprise edition[2] with additional features exists that supplies further hypervisors (e.g., *VMware*), guest operating systems (e.g., *MS Windows*), management options (users, quotas, and others) and database bindings and other extensions. This work will concentrate on the open-source solution as it satisfies all arising requirements.

2.4.2 Google App Engine

Google App Engine (GAE) is a software framework that allows the development and deployment of web applications. It runs applications on Google's server infrastructure and can be classified in the PaaS cloud computing domain. At the present time the GAE supports Java

[2]http://www.eucalyptus.com (September 20, 2011)

and JVM runtime environment-based languages, for example Clojure, Groovy, or Scala as well as dedicated Python and Go runtime environments.

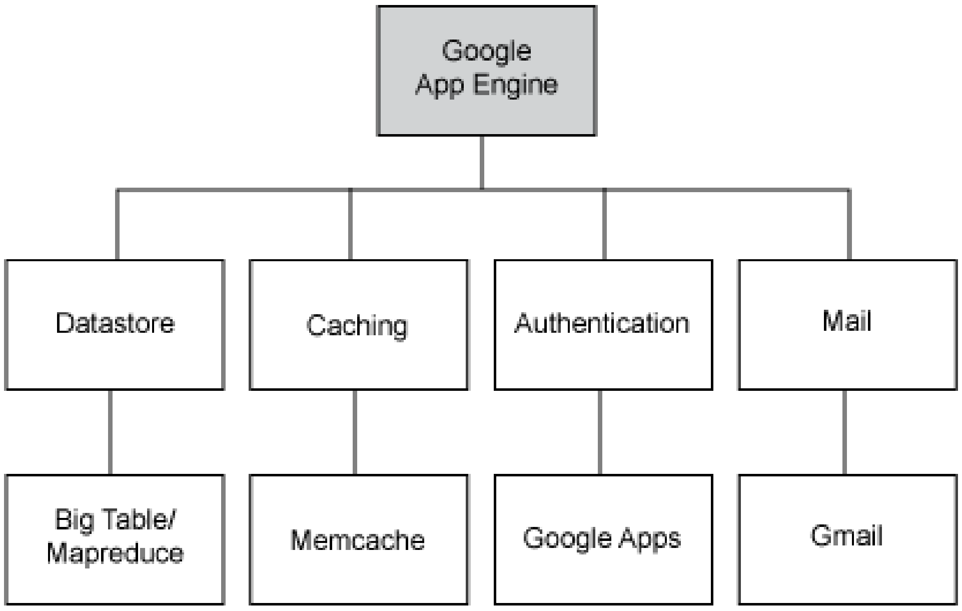


Figure 2.7: Google App Engine service implementations.[3]

In order to enable scalability and to enhance the robustness and stability of services, the scope of libraries that these languages are allowed to use is restricted by the framework. These restrictions include, for example, communication constraints, limited access to the local file system, interdiction of spawning background threads and response time limits for web requests (see Chohan et al. (2010). In the context of this thesis, these restrictions have a notable significance. In Section 3.1 for instance, restrictions will be defined in terms of cloud environment constraints (CECs) describing essential characteristics of cloud environments.

In addition to standard programming language features the App Engine offers interfaces to Google's proprietary technologies (e.g., Google File System, BigTable, MapReduce, MemCache, GoogleMail, GoogleAuthentication). Figure 2.7 shows a partial overview of provided services and its implementations in the App Engine.

[3]Figure taken from "Cloud services for your virtual infrastructure, Part 2", published at IBM developerWorks, Technical library, http://www.ibm.com/developerworks/opensource/library/ (September 20, 2011)

2.4.3 AppScale

AppScale is an open-source implementation of the Google App Engine APIs and is being developed at the RACELab (Research on Adaptive Compilation Environments), which is a research group of the University of California, Santa Barbara. AppScale is a platform-as-a-service cloud runtime system, that can be operated on different underlying cluster resources see (see Bunch et al., 2011). Besides popular cloud infrastructures, such as Amazon EZ2 and Eucalyptus, AppScale can be deployed on individual private server clusters, supporting Xen and KVM as virtualization technologies. Figure 2.8 shows different deployment modes that are possible to operate with AppScale.

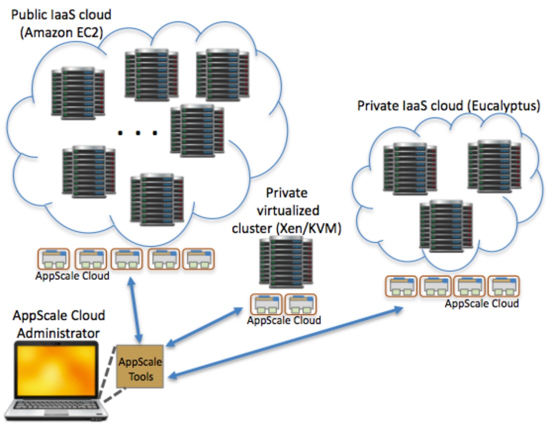

Public IaaS cloud (Amazon EC2)
Private IaaS cloud (Eucalyptus)
Private virtualized cluster (Xen/KVM)
AppScale Cloud
AppScale Cloud
AppScale Cloud
AppScale Cloud Administrator
AppScale Tools

Figure 2.8: Overview of AppScale cloud deployment modes.[4]

Adding support for executing GAE applications over virtualized clusters as well as providing the first open-source PaaS system are the declared key objectives of AppScale.

To offer users and developers a comfortable researching and testing environment according to their customed needs, AppScale implements the Google App Engine open APIs, thereby adding support for underlying IaaS cloud systems like EC2 and Eucalyptus. This allows developing, testing, debugging, and executing of GAE applications on the users' own clusters, focusing on scalability in particular. Existing GAE applications can be deployed to AppScale without the requirement to make any modifications (see Bunch et al., 2009).

Compared to the App Engine (see Figure 2.7), AppScale supports different implementa-

[4]Figure taken from AppScale project homepage, http://code.google.com/p/appscale/ (Sep. 29, 2011)

tions of the provided services as shown in Figure 2.9. For example, the GAE Datastore API can be implemented using different underlying database technologies. This is provided by an additional layer that links Google's datastore API with both relational (e.g., MySQL, Oracle) and key-value based (e.g., Voldemort, Cassandra, MongoDB) databases (see Bunch et al., 2010). From a programmer's point of view, the interface that connects to the Datastore layer is still accessed in a NoSQL style.

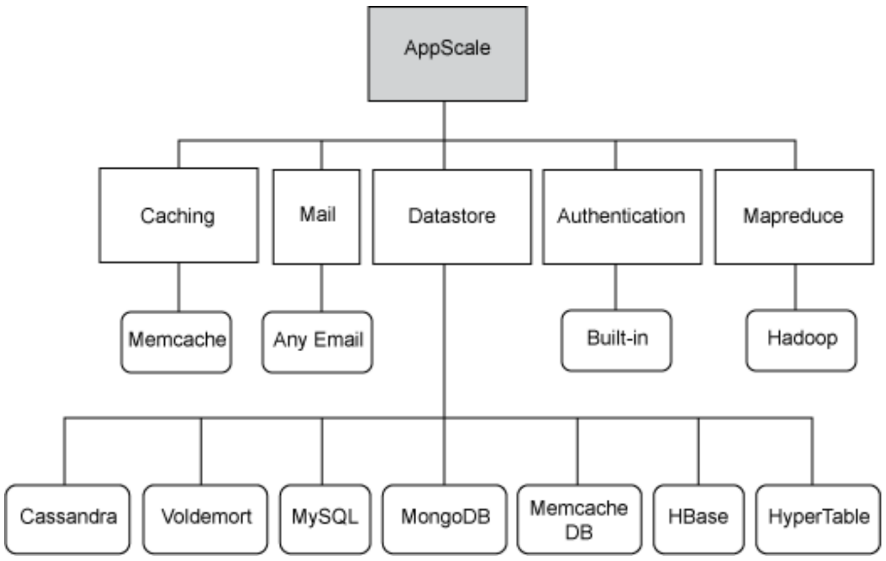


Figure 2.9: AppScale Service implementations.[5]

2.4.4 ADempiere

ADempiere is a community-developed open-source business software suite. It consists of a collection of business applications, including Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM), Financial Performance Analysis as well as integrated Point of Sale (POS) and Web Store solutions. Figure 2.10 shows an overview of the key features ADempiere supplies to facilitate process management and support business tasks.

[5]Figure taken from "Cloud services for your virtual infrastructure, Part 2", published at IBM developerWorks, Technical library, http://www.ibm.com/developerworks/opensource/library/ (September 20, 2011)

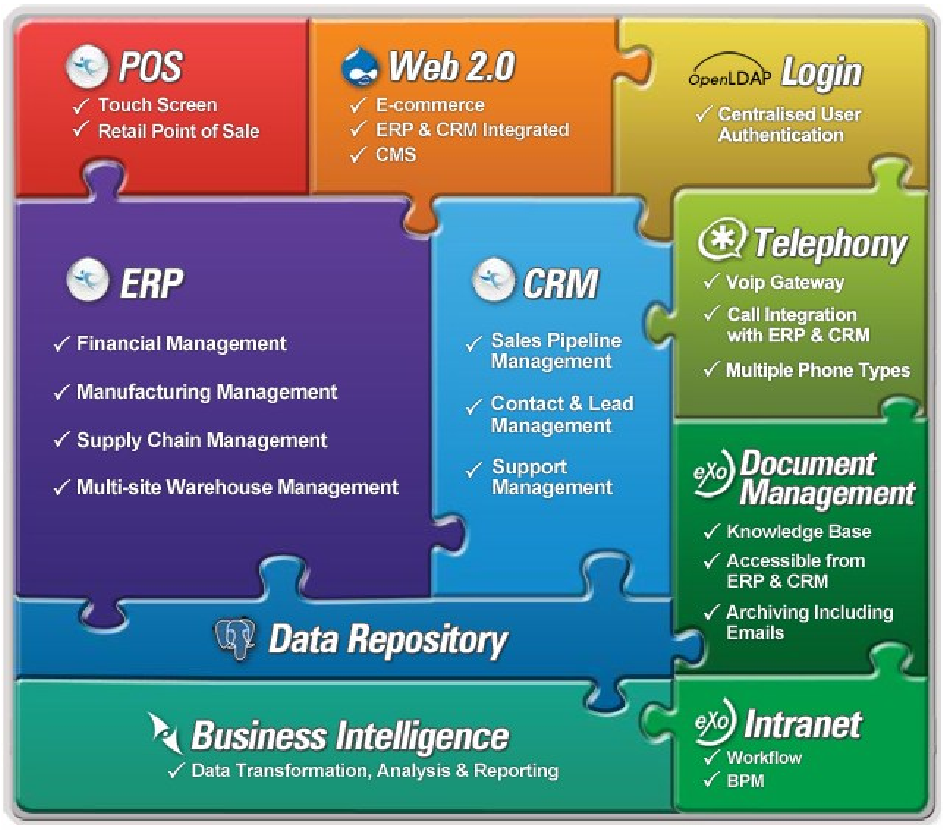

POS
✓ Touch Screen
✓ Retail Point of Sale
Web 2.0
✓ E-commerce
✓ ERP & CRM Integrated
✓ CMS
OpenLDAP Login
✓ Centralised User Authentication
ERP
✓ Financial Management
✓ Manufacturing Management
✓ Supply Chain Management
✓ Multi-site Warehouse Management
CRM
✓ Sales Pipeline Management
✓ Contact & Lead Management
✓ Support Management
Telephony
✓ Voip Gateway
✓ Call Integration with ERP & CRM
✓ Multiple Phone Types
eXo Document Management
✓ Knowledge Base
✓ Accessible from ERP & CRM
✓ Archiving Including Emails
Data Repository
Business Intelligence
✓ Data Transformation, Analysis & Reporting
eXo Intranet
✓ Workflow
✓ BPM

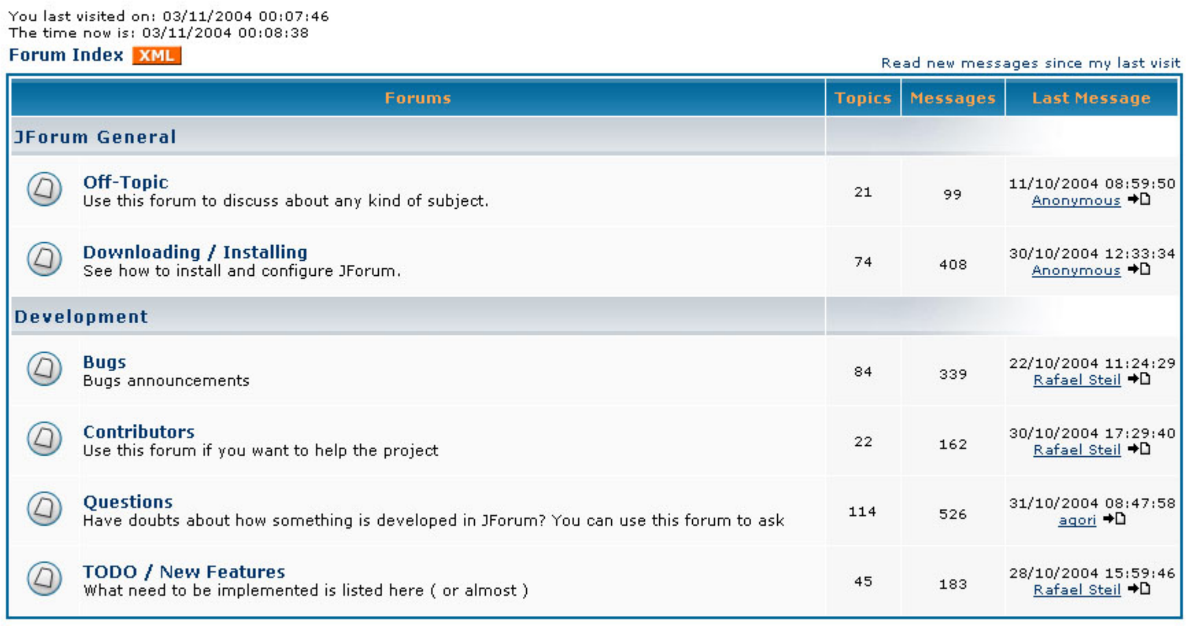
Figure 2.10: Overview of ADempiere Business Suite features.[6]

The ADempiere project was created in September 2006 as a fork of the Compiere open-source ERP, which has been founded and mainly developed by Compiere Incorporated. ADempiere is developed with JavaEE technologies, designed for the JBoss application Server. Currently supported database sytems are Oracle and PostgreSQL.

2.4.5 JForum

JForum is an open-source discussion board software. Figure 2.11 shows a screenshot of a typical web browser view of JForum with a forum listing. JForum is developed with Java technologies and runs on any established Java application server, including Tomcat, JBoss, and GlassFish. Several commonly accepted databases, such as MySQL, PostgreSQL, Oracle, and HSQLDB are supported.

[6]Figure taken from ADempiere product homepage, http://www.adempiere.com/ (Sep. 29, 2011)


You last visited on: 03/11/2004 00:07:46
The time now is: 03/11/2004 00:08:38
Forum Index XML

Read new messages since my last visit

Forums	Topics	Messages	Last Message
JForum General			
Off-Topic Use this forum to discuss about any kind of subject.	21	99	11/10/2004 08:59:50 Anonymous
Downloading / Installing See how to install and configure JForum.	74	408	30/10/2004 12:33:34 Anonymous
Development			
Bugs Bugs announcements	84	339	22/10/2004 11:24:29 Rafael Steil
Contributors Use this forum if you want to help the project	22	162	30/10/2004 17:29:40 Rafael Steil
Questions Have doubts about how something is developed in JForum? You can use this forum to ask	114	526	31/10/2004 08:47:58 agori
TODO / New Features What need to be implemented is listed here (or almost)	45	183	28/10/2004 15:59:46 Rafael Steil

Figure 2.11: Screenshot of the forum listings view of JForum.[7]

Some of characteristic features and advantages over similar products emphasized on the project homepage include:

- **Fast response times:** Through the caching of frequently accessed data, excessive database querying is avoided and the system remains fast and scalable.
- **Multi-language support:** Support for different languages with the ability to add new languages easily.
- **Customizability:** Individual profiles, language settings, themes, and templates per user.
- **Security:** Advanced HTML filter, highly customized permission control system on a forum / category and group / user basis.

[7]Figure taken from JForum product homepage, http://www.jforum.net/ (Sep. 29, 2011)

2.4.6 Knowledge Discovery Meta-Model (KDM)

Knowledge Discovery Meta-Model (KDM) is a specification defined by the Object Management Group (OMG). KDM represents architectural information (e.g., entities, attributes, relationships) about the software considered as well as additional artifacts (e.g., operational environment, components that interact with the software) (see Ulrich, 2004).

KDM uses a specific XML Metadata Interchange (XMI) model interchange format that is also developed by OMG and that is based on Extensible Markup Language (XML). One of the main characteristics of KDM is that it is independent of the programming language and the runtime platform. Another key feature is the ability to extend the core components with domain-, application-, and implementation-specific knowledge.[8]

KDM consists of four different layers that are built consecutively on top of each other. Figure 2.12 shows the layered structure of KDM. Each layer comprises several packages (12 overall) that represent different specific elements of the entire knowledge about the system.

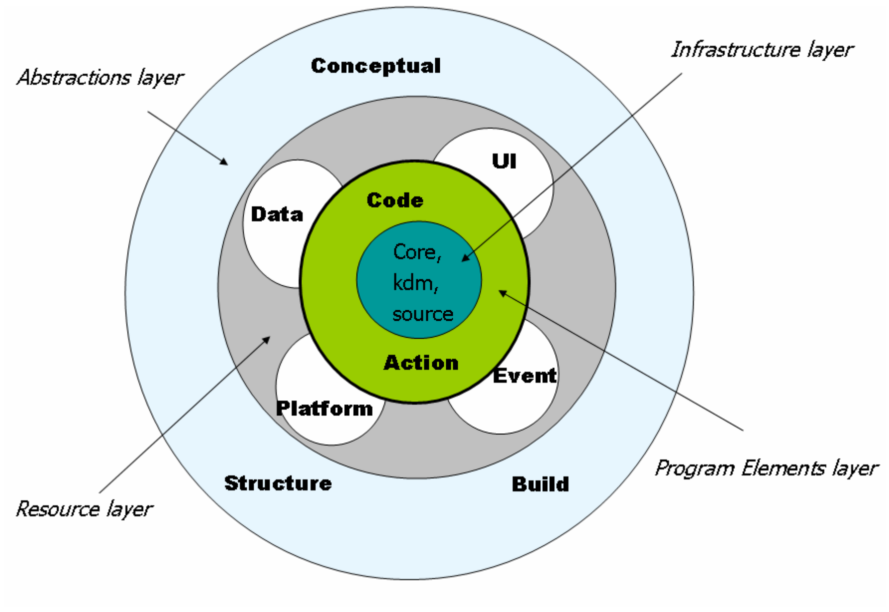


Figure 2.12: Layered structure of the Knowledge Discovery Meta-Model.[8]

[8]Taken from KDM technical overview web page, http://www.kdmanalytics.com/kdm/kdm_overview.html (Sep. 29, 2011)

2.4.7 MoDisco

MoDisco is a generic and extensible framework for model-driven reverse engineering. It is primarily designed to support legacy software modernization. The MoDisco platform is developed as an open-source project and is available as an Eclipse plug-in and part of the Eclipse Modeling Project.

One of the main issues concerning modernization of legacy software is getting detailed information about the software system that has to be modernized. This problem is even more relevant if no exact information about the system's actual architecture exists. Incomplete or obsolete information is not unusual, especially for software that is situated in environments with successively changing demands or when the system underlies altering hardware infrastructure or software components. MoDisco aims to retrieve this information by reverse engineering the application considered. A schematic overview of the reverse engineering process is illustrated in Figure 2.13.

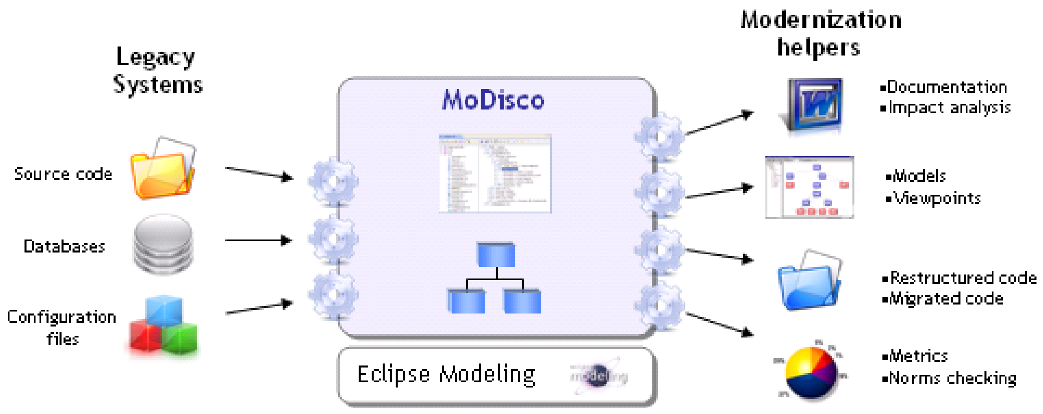


Figure 2.13: Overview of MoDisco's reverse engineering process.[9]

In the first phase of this reengineering process, the source code or other data artifacts of the applications are taken as input. This can be, for example, one specific discoverer component of MoDisco, which extracts a unified model as basic output. The discoverer is provided with a meta-model corresponding to the type of application that has to be reverse engineered.

[9]Figure taken from MoDisco homepage, http://www.eclipse.org/MoDisco/ (Sep. 29, 2011)

The second phase is used for various refining activities, where the basic models from the first phase are transformed and analyzed to obtain the desired target information, including different model representations, metric computations, re-factoring, generation of code, and documentation (see Bruneliere et al., 2010).

2.4.8 CloudMIG Xpress

CloudMIG Xpress is a prototype tool presented by Frey and Hasselbring (2011) that supports reengineers in migrating software systems to the cloud following the CloudMIG approach (see Section 2.3). It has a modular structure that allows for extension of its functional range by attaching plug-in features. Figure 2.14 shows an overview of CloudMIG Xpress including a set of possible plug-in components annotated with the respective corresponding activity. Furthermore, the *cloud profile* should be mentioned as an additional artifact representing the extensible repository of cloud profiles that are modeled in CloudMIG's selection activity *A2*.

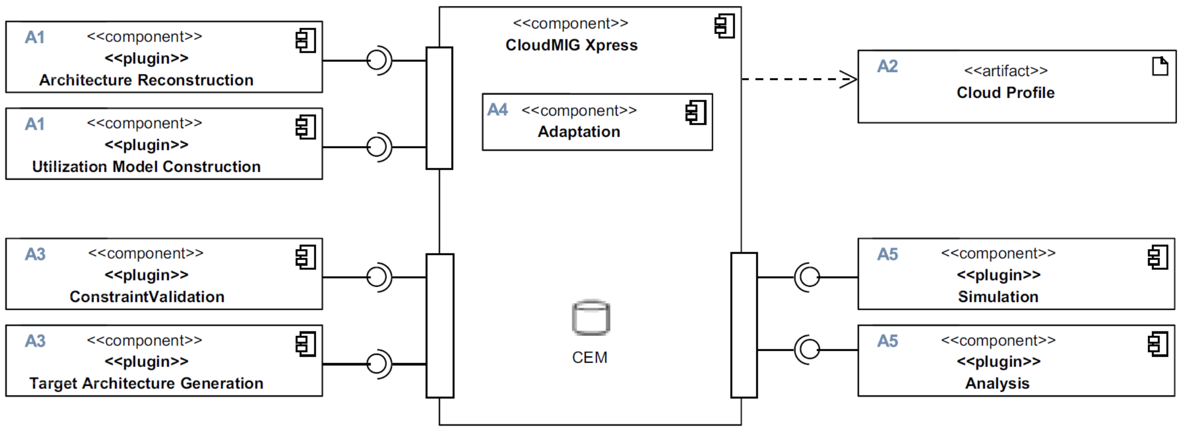

A1 <<component>> <<plugin>> Architecture Reconstruction
A1 <<component>> <<plugin>> Utilization Model Construction
A3 <<component>> <<plugin>> ConstraintValidation
A3 <<component>> <<plugin>> Target Architecture Generation
<<component>> CloudMIG Xpress
A4 <<component>> Adaptation
CEM
A2 <<artifact>> Cloud Profile
A5 <<component>> <<plugin>> Simulation
A5 <<component>> <<plugin>> Analysis

Figure 2.14: Overview of CloudMIG Xpress (Frey and Hasselbring, 2011).

An implementation of the *architecture reconstruction* component that is based on open-source libraries of MoDisco (see Section 2.4.7) has already been realized by the authors of CloudMIG. This component provides the extraction of architectural information from the target application by means of a software reverse engineering process and supplies the result as a KDM-conform model. A supplementary feature that allows for importing already existing models to CloudMIG Xpress as an alternative to reconstruction has been implemented in this thesis (see Section 3.2). *Constraint validation*, one of the central aspects of CloudMIG's activity *A3 Generation*, is designed to be carried out by additional constraint validators, following a plug-in concept. Several constraint validator plug-ins have already been implemented as well. The validation component is a component of significance in this thesis as it is investigated for its detection capabilities in Section 6.2. Due to the fact that CloudMIG Xpress only provides static analyses so far, it is not capable of detecting constraint violations that only occur at runtime.

This thesis will focus on the features mentioned above. The remaining components (*Utilization Model Construction*, *Target Architecture Generation*, *Adaption*, *Simulation*, and *Analysis*) that are shown in Figure 2.14 are conceptually planned for further development but not yet implemented.

3 Model Construction

According to the CloudMIG approach (see Section 2.3), several models are utilized to accomplish the generation of the target architecture as well as the detection of CEC violations. These models represent essential elements of the migration process and are described in more detail in the following:

- An instance (cloud profile) of the **Cloud Environment Model (CEM)** for the specific *AppScale* cloud system describes the technical characteristics of the target environment. As the cloud environment imposes the constraints on the hosted applications, the created CEM instance combined with the architectural model of the legacy system allows the implemented *constraint validators* to identify constraints that would lead to CEC violations if the application were to be deployed unchanged to the target cloud. Based on a generic CEM proposed by CloudMIG, a specific instance is derived to represent the exact shape of the considered cloud environment. This activity is dealt with in Section 3.1.

- The extraction of the **Architectural Model** (see Section 3.2) is proposed as being performed with the aid of *MoDisco*, a software reconstruction tool, resulting in an instance of the Knowledge Discovery Meta-Model (KDM). In the course of this process an evaluation of the KDM extraction mechanism will be addressed as the minor goal described in Section 1.2.3.

3.1 Definition of the Cloud Profile

The Cloud Profile of the AppScale system is a specific instance of the generic Cloud Environment Model (CEM) Meta-Model proposed by CloudMIG. This model is constructed by manually describing elements of the meta-model with matching specifications of the cloud environment. As there are no tools available that support the reengineer in gathering these characteristics automatically, one has to obtain the required information by examining the system's documentation or source code.

The documentation of AppScale lacks the essential information for describing the characteristics and deriving specific constraints of the AppScale cloud environment. Further, no related work exists that could deliver the corresponding data. An inspection of the source code indicated that AppScale implements the open-source GAE SDK, that has been affirmed by the developers as well. These facts led us to revert to the existing instance of the CEM Model describing Google App Engine for Java SDK 1.3.6 as the basis for further steps. An instance of the Google App Engine CEM has already been established by Frey and Hasselbring (2011) for the investigation of CEC violations of different applications performing the CloudMIG approach.

Listing 3.1 shows an excerpt of the GAE cloud profile. It includes the description of one cloud service (*PersistenceCloudService (GAE Datastore)* [lines: 7-11], three constraints (*FilesystemAccessConstraint (Write)* [lines: 13-21], *MethodCallConstraint (Systemexit)* [lines: 22-28], and *TypesWhitelistConstraint (AllowedJRETypes)* [lines: 29-39]) and one type list *WhiteList (GAE JRE Type Whitelist)* [lines: 41-47] that defines allowed types that are checked by the TypesWhiteListConstraint validator.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <cloudprofile:CloudEnvironment xmi:version="2.0"
 3    [...]
 4    id="org.cloudmig.cloudprofiles.gae"
 5    providerName="Google Inc." version="0.1">
 6   <environmentConfiguration id="org.cloudmig.cloudprofiles.gae.java">
 7     <cloudService xsi:type="iaas:PersistenceCloudService"
 8        description="The App Engine datastore is a [...]"
 9        id="org.cloudmig.cloudprofiles.gae.datastore"
10        name="Google App Engine Datastore"
11        [...] />
12     <constraintConfiguration name="Constraints">
13       <constraint xsi:type="constraint:FilesystemAccessConstraint"
14          id="org.cloudmig.cloudprofiles.gae.java.contraint.
15             filesystemaccess.nofilesystemwrite"
16          descr="An App Engine application cannot write to the
17             filesystem. [...]" name="No Filesystem Write"
18          violationSeverity="Critical">
19            <proposedSolution solution="Utilize the App Engine
20               Datastore for storing persistent data."/>
21       </constraint>
22       <constraint xsi:type="constraint:constraint:MethodCallConstraint"
23          id="org.cloudmig.cloudprofiles.gae.java.contraint.methodcall.
24             systemexit"
25          descr="The call to java.lang.System.exit() method does nothing
26             in App Engine."
27          [...]
28       </constraint>
29       <constraint xsi:type="constraint:TypesWhitelistConstraint"
30          id="org.cloudmig.cloudprofiles.gae.java.contraint.typeswhitelist.
31             allowedjretypes"
```

```
         descr="Access to the classes in the Java standard library (the
            Java Runtime Environment, or JRE) is limited to the classes
            in the App Engine JRE White List."
         types="//@environmentConfiguration.0/@constraintConfiguration.0/
            @typeLists.0"
         violationSeverity="Critical"
         [...]
      </constraint>
      [...]
      <typeLists id="org.cloudmig.cloudprofiles.gae.java.contraint.
         typelist.whitelist" name="Google App Engine JRE Whitelist">
        <type name="Boolean" package="java.lang"/>
        <type name="Byte" package="java.lang"/>
        <type name="StringReader" package="java.io"/>
        [...]
      </typeLists>
      [...]
    </constraintConfiguration>
  </environmentConfiguration>
</cloudprofile:CloudEnvironment>
```

Listing 3.1: Excerpt of the App Engine cloud profile.

The definition of new cloud profiles is a complex and error-prone activity, as highlighted by the following points:

- Since the information has to be extracted from different non-standardized sources, such as documentation, case studies, and one's own research, the data is not only difficult to obtain but is, moreover, often vague and incomplete.
- The required information is basically provided in textual form, so that a translation to the abstract elements of the cloud environment model has to be made. This circumstance often leads to a certain degree of inaccuracy, which is caused by the fact that a model always provides only a limited representation of reality. Furthermore, this translation depends on the subjective appraisal of the engineer.
- Both the information retrieval as well as the development of the model instance are manual and non-standardized activities that are not subject to automated validation mechanisms. As for every manually conducted process, errors are never completely avoidable.

These drawbacks are diminished by the fact that a cloud profile, including its implemented constraint validators, has to be developed only once for each target cloud system. Further, to generate related profiles, an existing model can be (partly) adopted and enhanced to build different shapes of the profiles. Therefore, CloudMIG aims at providing a public repository of cloud profiles in the future to exploit the re-usability and benefits of cooperation.

3.2 Extraction of the Architectural Model

CloudMIG is designed to support reengineers in migrating software systems to PaaS- and IaaS-based cloud environments. The goal is to identify potential obstacles that would impede the application from running in the target cloud in order to be able to adjust the existing system to match suitably with the cloud's specifications. This intention requires a high level of information about the considered software system. The desired application for migration in this thesis has initially been the complex ERP software ADempiere (see Section 2.4.4). The contemplated extraction procedure using MoDisco has not proven to be applicable. The particular procedure and problems exposed will be addressed in Section 3.2.1, while a critical examination and appraisal concerning this issue is covered in Section 3.2.3.
Since this risk has been conceivable from the beginning (see Section 1.2.3), JForum (see Section 2.4.5) has been chosen as an alternative option from a pool of applications known to be suitable for model extraction in order to proceed with the work. The model extraction procedure of the fallback solution JForum is described in Section 3.2.2.

3.2.1 Extraction of ADempiere's Model

A first attempt of extracting an architectural model of ADempiere was made on a typical desktop PC platform. The system's initial hardware and software configuration is listed in Table 3.1.

Desktop PC		
	CPU:	Intel Core2Duo E6750 2core 2.66GHz
	RAM:	2x1GB DDR3-1066MHz
	OS:	MS Windows 7 64 Bit
	JVM:	Java HotSpot(TM) Server VM
	Eclipse:	Helios 64 Bit Modeling Package
	MoDisco:	Eclipse Plug-in (Version 0.8.2)

Table 3.1: Configuration of the desktop PC for model extraction.

The MoDisco Model Discovering task was launched with the default settings. After a few hours of executing with full CPU utilization, the process ended up with an error message from the underlying JVM, indicating a heap overflow error. To avoid this problem, as a first measure the procedure was repeated with an increased JVM heap size that exploited the available memory of 2 GB. This attempt failed in the same way.

Thereafter, several settings of MoDisco were operated subsequently. None of the different settings, which included an option to run the procedure *incrementally*, thereby intending to occupy memory more economically, could avert the heap overflow error.

Next, the initial software and hardware setup was changed. The process still failed with updated software versions (including Nightly Builds) of Eclipse and MoDisco as well as with a larger amount of memory (4 GB RAM).

To take advantage of a further increased hardware environment utilizing the blade servers of the Software Engineering Group at the Christian-Albrechts-University Kiel, the extraction had to be released from the Eclipse IDE integration and carried out as an independent action, executable from the command line of a common SSH shell. An overview of the provided server configuration is given in Table 3.2.

Server		
Blade 1		
	Type:	X6270
	CPU:	2xIntel Xeon E5540 4core 2.53GHz
	RAM:	12x2GB DDR3-1066MHz ECC
	OS:	Solaris 10 (SunOS 5.10)
	JVM:	Java HotSpot(TM) Server VM
Blade 2		
	Type:	X6270
	CPU:	2xIntel Xeon E5540 4core 2.53GHz
	RAM:	12x2GB DDR3-1066MHz ECC
	OS:	Linux Debian 2.6.26-2-amd64
	JVM:	OpenJDK 64-Bit Server VM
Blade 3		
	Type:	T6340
	CPU:	2xUltraSparcT2+ 8core 1.4GHz
	RAM:	16x4GB FB DIMM
	OS:	Solaris 10 (SunOS 5.10)
	JVM:	Java HotSpot(TM) Server VM

Table 3.2: Configuration of the blade servers for model extraction.

The model extraction procedure of MoDisco is a combined procedure, in which a prior generated intermediate model is transformed to the final KDM model. This two-stage technique maximizes flexibility for varying reverse engineering purposes. Besides the KDM-based model representation of the application's architecture that the CloudMIG approach aims for, a reengineer might have other intentions that depend on different information (see Section 2.4.7). Therefore, as a first step, the considered application source code and additional artifacts (e.g., configuration files or database data) are examined for any relevant information and a model is generated that gives a universal representation of the system (see Bruneliere et al., 2010). The resulting intermediate model serves as a basis for further information retrieval.

In our case, the first stage is detected as completing successfully whereas the second stage causes the errors in Eclipse. Hence, the successfully obtained intermediate model is intended to be retained and transformed by executing an ATL-based[1] model-to-model transformation on the blade servers. A schematic overview of the transformation principle, according to Allilaire et al. (2006) is illustrated in Figure 3.1, where a model *Ma* is transformed to a model *Mb*.

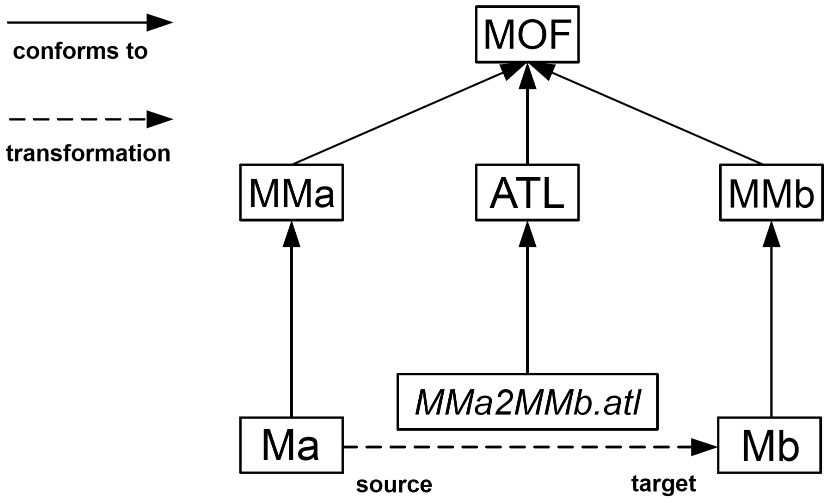


Figure 3.1: ATL model-to-model transformation schema (according to Allilaire et al., 2006).

[1]ATL: Atlas Transformation Language, http://www.eclipse.org/atl/ (Sep. 29, 2011)

The source model *Ma* conforms to the meta-model *MMa* which defines the model specification. On the target side, there is an analogues combination of the target model *Mb* and its meta-model *MMb*. The ATL transformation engine, which defines a model transformation meta-model *ATL*, is provided with a conform model transformation model *MMa2MMb.atl* that describes the mapping of source elements to target elements. The Meta Object Facility *MOF*, a standard of the Object Management Group (OMG), lies on top of the three meta-models that each have to conform to the *MOF*.

Listing 3.2 shows a source code excerpt of the extracted model transformation procedure that corresponds to the ATL model-to-model transformation described above.[2] Initially, the meta-model definitions of the source (*javaModel*) and target (*kdmModel*) models are initialized and the source model (which is the intermediate model of the MoDisco model extraction) is loaded according to its meta-model from the local file system (*inFilePath*). Then, a new instance (*launcher*) of the transformation launcher is supplied with the source and target models, some configuration options (*objects*), and the transformation model (*javaToKdm.asm*). Afterwards the launcher is executed and the target model is exported to the local file system (*outFilePath*).

```
[...]
// Get instances of the input and output models used
// during transformation and load (inject) input model
IModel javaModel = factory.newModel(javaMetaModel);
IModel kdmModel = factory.newModel(kdmMetaModel);
injector.inject(javaModel, inFilePath);

// Get an instance of the transformation launcher and
// add models to the transformation context
ILauncher launcher = new EMFVMLauncher();
launcher.initialize(Collections.<String, Object> emptyMap());
launcher.addInModel(javaModel, "IN", "java");
launcher.addOutModel(kdmModel, "OUT", "kdm");

// Set up options of the ATL-VM used for the
// transformation process
HashMap<String, Object> objects = new HashMap<String, Object>();
objects.put("step", "false");
objects.put("printExecutionTime", "true");
objects.put("allowInterModelReferences", "true");
objects.put("showSummary", "true");

// Set up the resource URL of the asm definition file
URL transformationResourceURL =
    Java2Kdm.class.getResource("/javaToKdm.asm");

```

[2] A more detailed source code excerpt is given in Appendix A

```
// Launch the transformation using given parameters
launcher.launch(ILauncher.DEBUG_MODE, new NullProgressMonitor(),
    objects, transformationResourceURL.openStream());

// Export the KDM formatted model to output file
extractor.extract(kdmModel, outFilePath);
[...]
```

Listing 3.2: Source code excerpt of the command line program for ATL model-to-model transformation.

The model transformation was executed on each of the three blades with the program mentioned above. Even after several days of execution time with heavily utilized CPU and peak memory usages of 13 GB, none of the processes had finished and it was decided to interrupt them.

As a final effort concerning the memory issue, the individual packages of ADempiere (e.g., *base*, *util*) were treated separately during the model discovery with MoDisco. Every package, except the *base* package processed successfully with a maximum execution time of a few hours. Regrettably, the *base* package showed the same deficiency as the entire application did before.

Both our own initial investigations as well as confirming statements of MoDisco developers indicated the error source as residing in the ATL script used to translate the intermediate model to KDM. In consideration of the residual goals of this thesis we decided not to proceed with the investigation of this problem at this point, but to switch to the fallback application JForum instead.

3.2.2 Extraction of JForum's Model

JForum has been designated as the fallback application to examine instead of ADempiere, which proved to be infeasible in this context, due to extraction errors with MoDisco (see previous Section 3.2.1).

The extraction of the architectural model of JForum is described in this section. The MoDisco Discoverer of the Eclipse plug-in was applied on JForum and its libraries. The previously mentioned standard desktop PC (see Table 3.1) was used again and the Discoverer was executed using default settings with the *incrementally* option activated. Both JForum's own sources as well as the libraries could be processed without errors.

3.2.3 Analysis of the KDM Extraction Process

As the thesis of Pascal Löffler has already shown, difficulties might occur in the model extraction process with MoDisco, especially when working with large applications. Being aware of this threat, exposing potential issues and investigating solutions to keep following up the intended strategy of the CloudMIG approach have been declared as minor goals of this thesis.

Unfortunately, the raised concerns were well-founded and our attempt to extract the architectural model of ADempiere with MoDisco failed (see Section 3.2.1). Different modifications with the intention of solving the problems arising from the lack of memory merely bypassed the errors and caused a non-terminating extraction process, probably due to an infinite execution loop.

Since the developers of MoDisco had not encountered such issues before, the consultation did not provide a proper solution to the problem. Supposing that the memory problems were caused by the huge sources of ADempiere, they recommended either reducing the extent of relevant expressions and code elements by setting filters in the MoDisco Discovering action or dividing the responsible *base* package of ADempiere into smaller parts. Both measures seemed to be rather inadequate to our approach, because it would imply a grave loss of information or an exaggerated amount of manual work that would massively weaken the benefit of the CloudMIG Approach.

Taking the young history and active development of MoDisco into account, it might still be reasonable to regard them as a promising candidate for providing the required KDM models in the future.

We decided not to pursue further debugging actions concerning the extraction procedure, such as uncovering endless loops, which may have been a reason for the erroneous behaviour, as this would disregard the other goals and would go beyond the scope of this examination.

4 CEC Violation Identification with CloudMIG

Constraints describe limiting factors of the cloud environment that prevent technical actions of guest applications from being executed correctly (see Frey and Hasselbring, 2011). These constraints are imposed by the specific cloud characteristics, hence they will be specified in the CEM instance. The according framework is represented by the CEM Constraints package. To define a concrete constraint, a library of abstract constraints containing classes like `AbstractNetworkConstraint` or `AbstractTypeListConstraint` is provided by the CEM, which are inherited to match the particular attributes.

4.1 Overview of CEC Violations

The CloudMIG Approach defines a set of CECs (see Section 2.3 and Section 3.1). If a specific CEC is not met by some part of the application it causes a CEC Violation, according to the constraint it is based on.

A short description[1] of several instantiated constraints (based on the constraint definition of the CloudMIG Approach) for the GAE cloud profile and an example of a typical violation for the according constraint is described in the following. Constraints that caused no violation (e.g., LanguageConstraint, FirewallPortRangeConstraint, RuntimeContainerLifetime-Constraint) are not considered here.

- **TypesWhiteList:** Access to the classes in the Java standard library (the Java Runtime Environment, or JRE) is limited to the classes in the App Engine JRE White List.

[1]The descriptions are adopted from the (sandboxed) App Engine Java API documentation, http://code.google.com/intl/de-DE/appengine/docs/java/runtime.html (October 5, 2011)

Example: Implementing the interface *java.rmi.server.Skeleton*. This interface is not available in the App Engine and will cause a runtime error.

- **TypesInstantiation:** Java applications can create neither a new *java.lang.Thread* nor a new *java.lang.ThreadGroup*. These restrictions also apply to JRE classes that make use of threads. For example, an application cannot create a new *java.util.Timer*, or a *java.util.concurrent.ThreadPoolExecutor*. An application can perform operations against the current thread, such as *Thread.currentThread().dumpStack()*.

  Example: Instantiation of *java.lang.Thread*. This class is not available in the App Engine and will cause a runtime error.

- **FilesystemAccess (Read):** An App Engine application can read files, but only files uploaded with the application code.

  Example: Instantiation of *java.io.File*. If the instantiated file is not part of the uploaded application content this operation is not allowed and will cause a runtime error.

- **FilesystemAccess (Write):** An App Engine application cannot write to the file system. Applications must use the App Engine datastore for storing persistent data.

  Example: Call of the method *java.io.File.delete()*. This method writes to the local file system and will cause a runtime error.

- **Reflection:** An application is allowed full, unrestricted, reflective access to its own classes. It may use *java.lang.reflect.AccessibleObject.setAccessible()*, query any private members, and read/set private members. An application can also reflect on JRE and API classes, such as *java.lang.String* and *javax.servlet.http.HttpServletRequest*. However, it can only access public members of these classes, not protected or private members. An application cannot reflect against any other classes not belonging to itself, and it can not use the *setAccessible()* method to circumvent these restrictions. However, tests showed that reflecting is even possible on classes of imported libraries.

  Example: Call of the method *java.lang.reflect.Method.invoke()*. If the reflected method is not a member of an own class or a public member of any other class this method call is not allowed and will cause a runtime error.

- **MethodCall (SystemExit):** The call to *java.lang.System.exit()* method does nothing in the App Engine.

  Example: Call of the method *java.lang.System.exit().* This call does nothing, which probably was not the developer's intention.

- **SocketOpening:** Google App Engine-based applications cannot make socket connections directly.

  Example: Instantiating *java.net.Socket.* The opening of sockets directly is not allowed in the App Engine and will cause a runtime error.

CEC violations are assigned with a severity that specifies the complexity of resolving the violation during the migration process. The CloudMIG approach proposes three levels of severity (see Frey and Hasselbring, 2011) which, it should be noted, comprise a pessimistic, subjective rating. The developer of a cloud profile has to classify the constraints by the following severity levels:

(i) *Warning* violations need moderate adjustments for elimination.

(ii) *Critical* severity means that one can cope with these violations too, though extensive actions are required to handle them.

(iii) *Breaking* violations prevent the migration from being carried out successfully.

4.2 CloudMIG's CEC Violation Detection Mechanism (Constraint Validators)

CloudMIG provides the detection of CEC violations as a fundamental part of the *Generation* activity *A3* (see Section 2.3). In that phase, all constraints have been defined as CEM elements in the considered cloud profile and a KDM-based architectural model of the system to be migrated exists. CloudMIG allows the development and plugging-in of one or multiple validator components for every constraint type. Figure 4.1 shows classes that are incorporated in the constraint validation design of CloudMIG Xpress. Each validator plug-in has to provide a subclass of `AbstractConstraintValidator`. Therefore, one of the already available abstract constraint validator classes can be extended. These classes are designed hierarchically and represent common characteristics of recently developed validator types. Currently predefined abstract validators are `AbstractStaticConstraintValidator` (static CEC validation), `AbstractConstraintKDMValidator` (static KDM-based CEC validation), and `AbstractConstraintSMMKDMValidator` (static KDM-based CEC validation with SMM-based metrics data).

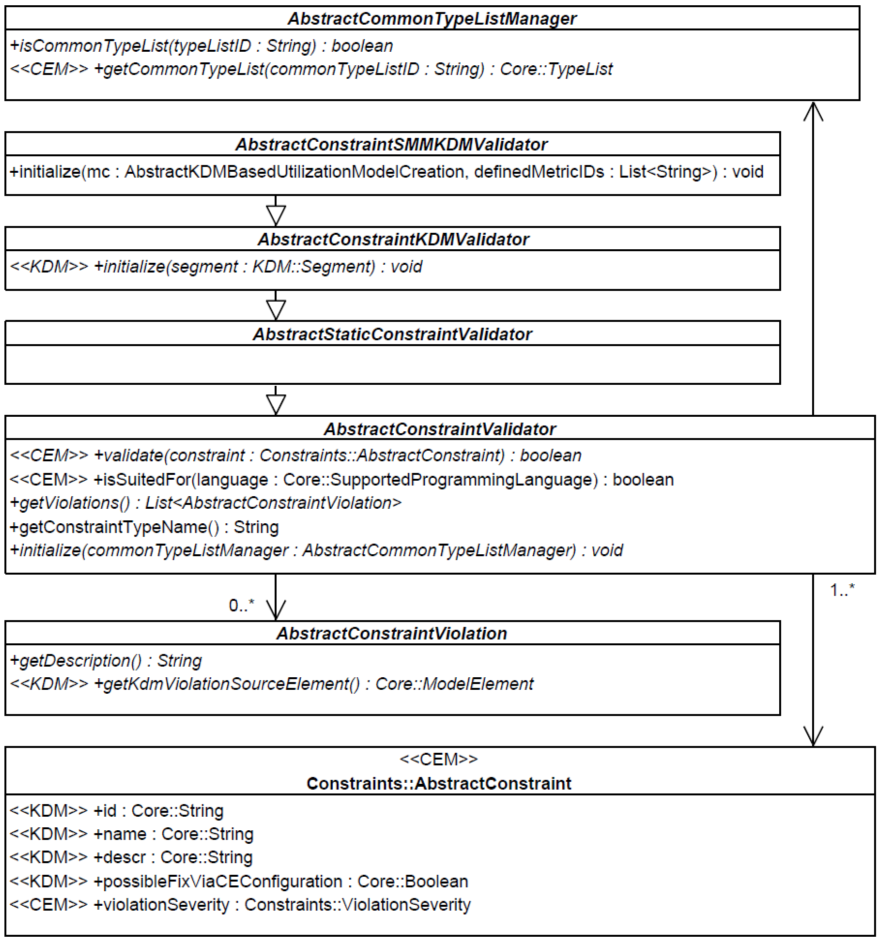

AbstractCommonTypeListManager
+isCommonTypeList(typeListID : String) : boolean
<<CEM>> +getCommonTypeList(commonTypeListID : String) : Core::TypeList
AbstractConstraintSMMKDMValidator
+initialize(mc : AbstractKDMBasedUtilizationModelCreation, definedMetricIDs : List<String>) : void
AbstractConstraintKDMValidator
<<KDM>> +initialize(segment : KDM::Segment) : void
AbstractStaticConstraintValidator
AbstractConstraintValidator
<<CEM>> +validate(constraint : Constraints::AbstractConstraint) : boolean
<<CEM>> +isSuitedFor(language : Core::SupportedProgrammingLanguage) : boolean
+getViolations() : List<AbstractConstraintViolation>
+getConstraintTypeName() : String
+initialize(commonTypeListManager : AbstractCommonTypeListManager) : void
0..*
1..*
AbstractConstraintViolation
+getDescription() : String
<<KDM>> +getKdmViolationSourceElement() : Core::ModelElement
<<CEM>>
Constraints::AbstractConstraint
<<KDM>> +id : Core::String
<<KDM>> +name : Core::String
<<KDM>> +descr : Core::String
<<KDM>> +possibleFixViaCEConfiguration : Core::Boolean
<<CEM>> +violationSeverity : Constraints::ViolationSeverity

Figure 4.1: Classes incorporated in CEC validation process (Frey and Hasselbring, 2011).[2]

[2]Updated version (Sep. 2011)

When the violation detection is started in CloudMIG Xpress, each validator that is both suited for the programming language (`isSuitedFor()`) and relevant for any of the existing constraints (`getConstraintTypeName()`) is executed by calling `initialize()` and `validate()`. The latter method returns a Boolean value, indicating whether the validation holds or violations are identified. The `getViolations()` method delivers the corresponding list of CEC violations identified by the validator (see Frey and Hasselbring, 2011).

Following the CloudMIG approach, the constraints are defined as CECs in KDM-based cloud profiles. Even though the KDM specification allows the representation of architectural information independent from the application's programming language (see Section 2.4.6), several validators require language-specific knowledge for the detection of particular constraint violations. Table 4.1 differentiates between constraints that depend on language-specific analysis and constraints that generic analysis complies with.

Generic	**Language-specific**
LanguageConstraint	FileSystemAccessConstraint
MaxTotalNrOfFilesConstraint	LocalTransientStorageConstraint
MethodCallConstraint	ReflectionConstraint
SMMConstraint	SocketOpeningConstraint
TypesInstantiationConstraint	
TypesWhitelistConstraint	

Table 4.1: Generic and language-specific KDM-based constraint validation.

4.3 Implementation of a Model Import Feature in CloudMIG Xpress

The current version of CloudMIG Xpress is based on the Eclipse Indigo framework and incorporates the Indigo release of MoDisco (0.9.x) components. With this particular version of the MoDisco Discoverer component, the model extraction of JForum raised an error[3][4] and no KDM model could be generated successfully. As preceding work has shown, earlier Helios releases of MoDisco (0.8.x) worked without errors when performing the model extraction on JForum and a stand-alone copy of JForum's KDM model was already available outside of CloudMIG Xpress. A further investigation of the causes for this extraction error will not be part of this thesis.

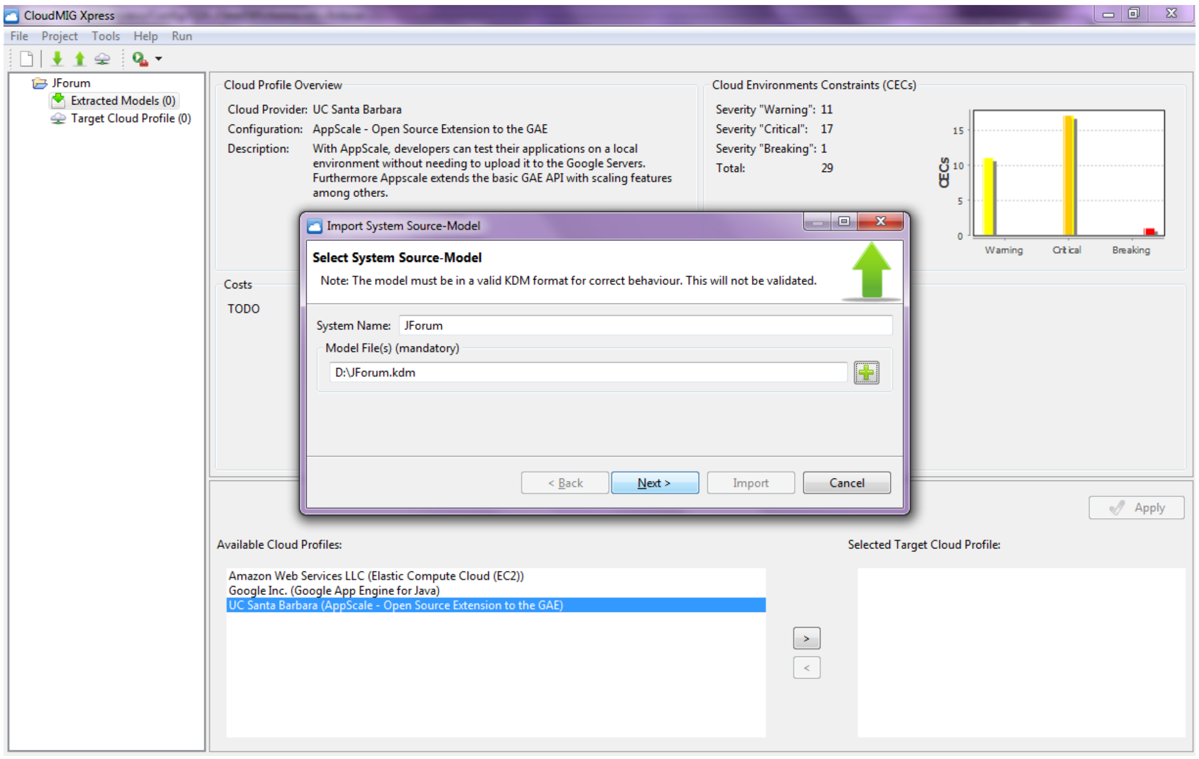
In order to exploit potential advantages of newer versions, CloudMIG Xpress should in general utilize the most recently developed components. Furthermore, the development of MoDisco bears several major API breaks from the Helios to the Indigo version, so that re-incorporating the older component in CloudMIG Xpress would have required a substantial reengineering effort.

As a result of these facts, the obligatory model extraction of CloudMIG Xpress was extended with a new import feature. This exhibits more flexibility in CloudMIG Xpress, for example to be able to outsource the extraction process of huge applications on dedicated high-performance servers or to re-use existing models. Figure 4.2 shows a model file selection dialog of the new import feature, with the main screen of CloudMIG Xpress in the background.

The import function loads the model resource from the local file system and integrates it into the internal model database, the same way as on-the-fly extracted models are handled. The current version provides no integrated validation check of the imported models, so that the user is responsible for the imported model's conformance to the required KDM model specification. This would be a reasonable project for future releases in order to enhance the reliability and usability of CloudMIG Xpress.

[3]Error Message: An internal error occurred during: "MoDisco discoverer "org.eclipse.modisco.java.discoverer.javaProjectToKDM" in progress...".

[4]Exception Stack Trace: org.eclipse.m2m.atl.engine.vm.VMException: Feature type does not exist on java!SingleVariableAccess at A.CreateUsesType(1 : Mjava!TypeAccess;) : ??#32(javaToKdm.atl[1886:10-1886:18]) local variables = tgt=OUT!<unnamed>, src=IN!<unnamed>, self=javaToKdm : ASMModule local stack = [OUT!<unnamed>, OUT!<unnamed>, javaToKdm : ASMModule]



Cloud Profile Overview

Cloud Provider: UC Santa Barbara
Configuration: AppScale - Open Source Extension to the GAE
Description: With AppScale, developers can test their applications on a local environment without needing to upload it to the Google Servers. Furthermore Appscale extends the basic GAE API with scaling features among others.

Costs

TODO

Cloud Environments Constraints (CECs)

Severity "Warning": 11
Severity "Critical": 17
Severity "Breaking": 1
Total: 29

Import System Source-Model

Select System Source-Model

Note: The model must be in a valid KDM format for correct behaviour. This will not be validated.

System Name: JForum

Model File(s) (mandatory)

D:\JForum.kdm

Available Cloud Profiles:

Amazon Web Services LLC (Elastic Compute Cloud (EC2))
Google Inc. (Google App Engine for Java)
UC Santa Barbara (AppScale - Open Source Extension to the GAE)

Selected Target Cloud Profile:

Figure 4.2: View of CloudMIG Xpress showing the new model import feature.

4.4 Application of CloudMIG Xpress' Violation Detection on JForum

This section describes the investigation of JForum for constraint violations with the help of the violation detection component of CloudMIG Xpress.

Architectural models (KDM source model) of JForum and its libraries had already been extracted using the Eclipse Plug-in of MoDisco, described in Section 3.2.2. Furthermore, the current model extraction component of CloudMIG Xpress showed problems with the extraction process of JForum, due to inconsistencies in the underlying MoDisco Discoverer module (see preceding Section 4.3). Hence, prior to the actual violation detection, a new feature was added to CloudMIG Xpress to be able to import existing models.

After the successful implementation of the import feature, the violation detection process of CloudMIG Xpress could be carried out successfully. An instance of Google App Engine CEM was selected as the target cloud profile and the models of both JForum's own sources and its libraries were subsequently loaded with the import feature and validated against the cloud profile. Each validation procedure executed swiftly and the results were exported as csv-based data into text files of the local file system without any difficulties.

Overall, the identification resulted in 4,716 (0 / 4,639 / 77)[5] violations, of which the JForum sources yielded 250 (0 / 173 / 77) violations and the libraries combined added up to 4,466 (0 / 4,466 / 0) violations.

Table 4.2 shows a quantitative overview of detected violations of JForum and its libraries. Additionally, the number of violations is compared to the number of classes and lines of code in order to obtain more significant information about violation frequencies. Some libraries' source code was unfortunately not provided and the corresponding values (indicated with 'x') are missing. Hence, adjusted summaries are given to prevent obscured data.

Table 4.3 gives a more detailed overview of violation frequencies in JForum's own classes. All classes that contain at least one violation are listed individually, provided with the number of violations, lines of code (LoC), and the according ratios for violations per 1,000 LoC.

A more detailed analysis of the detected violations is presented in Section 6.1.1.

[5]Sum (Breaking / Critical / Warning)

	Violations		Classes		LoC[6]		V/C[7]	$V/KLoC$[8]
Source	#	%	#	%	#	%		
JForum	**250**	**5.30**	**340**	**10.13**	**31,865**	**7.53**	**0.74**	**7.85**
Libraries	**4,466**	**94.70**						
Libraries (adjusted)[9]	**1,671**	**35.43**	**3,018**	**89.87**	**391,280**	**92.47**	**0.55**	**4.27**
jboss-jmx	2,297	48.71	x	x	x	x	x	x
jgroups-all-2.2.9-beta2	813	17.24	340	10.13	55,211	13.05	2.39	14.73
jboss-system	277	5.87	x	x	x	x	x	x
log4j-1.2.12	259	5.49	176	5.24	15,314	3.62	1.47	16.91
jcaptcha-all-1.0-RC2.0.1	198	4.20	163	4.85	7,109	1.68	1.21	27.85
jboss-cache-1.2.4	179	3.80	x	x	x	x	x	x
junit	134	2.84	77	2.29	3,118	0.74	1.74	42.98
htmlparser-1.5	113	2.40	232	6.91	35,073	8.29	0.49	3.22
postgresql-8.0-313.jdbc3	33	0.70	172	5.12	23,748	5.61	0.19	1.39
freemarker-2.3.9	28	0.59	284	8.46	40,018	9.46	0.10	0.70
mysql-connector-java-5.0.3-bin	26	0.55	110	3.28	37,825	8.94	0.24	0.69
activation	26	0.55	28	0.83	2,625	0.62	0.93	9.90
jboss-common	24	0.51	x	x	x	x	x	x
ojdbc14	18	0.38	x	x	x	x	x	x
mail	17	0.36	191	5.69	20,366	4.81	0.09	0.83
quartz-1.5.1	14	0.30	124	3.69	21,422	5.06	0.11	0.65
lucene-core-2.2.0	7	0.15	258	7.68	29,102	6.88	0.03	0.24
ehcache-1.1	2	0.04	13	0.39	2,045	0.48	0.15	0.98
concurrent	1	0.02	106	3.16	12,760	3.02	0.01	0.08
c3p0	0	0.00	250	7.44	25,317	5.98	0.00	0.00
tomcat-jasper	0	0.00	118	3.51	29,143	6.89	0.00	0.00
tomcat-servlet	0	0.00	113	3.37	4,269	1.01	0.00	0.00
commons-lang	0	0.00	77	2.29	16,921	4.00	0.00	0.00
tomcat-jsp	0	0.00	46	1.37	1,906	0.45	0.00	0.00
servlet-api	0	0.00	40	1.19	1,522	0.36	0.00	0.00
commons-io	0	0.00	39	1.16	3,301	0.78	0.00	0.00
tomcat-el	0	0.00	22	0.66	1,512	0.36	0.00	0.00
tomcat-annotations	0	0.00	20	0.60	220	0.05	0.00	0.00
lucene-highlighter	0	0.00	18	0.54	1,122	0.27	0.00	0.00
jargs	0	0.00	1	0.03	311	0.07	0.00	0.00
Total	**4,716**	**100.00**						
Total (adjusted)[10]	**1,921**	**40.73**	**3,358**	**100.00**	**423,145**	**100.00**	**0.57**	**4.54**

Table 4.2: Key figures of JForum and its referenced third-party libraries.

[6]Lines of code (pure source code without comments and blank lines), measured with CLOC V. 1.52, http://cloc.sourceforge.net/ (Sep. 28, 2011)

[7]V/C: #Violations per class

[8]$V/KLoC$: #Violations per 1,000 LoC

[9]Libraries (adjusted): Only libraries with complete data are considered here

[10]Total (adjusted): Only sources with complete data are considered here

Class	Violations		LoC[11]		$V/KLoC$[12]
	#	%	#	%	
Classes w/o violations	**0**	**0.00**	**25,729**	**80.74**	**0.00**
Classes with violations	**250**	**100.00**	**6,136**	**19.26**	**40.74**
net.jforum.util.image.ImageUtils	68	27.20	165	0.52	412.12
net.jforum.util.legacy.commons.fileupload.disk.DiskFileItem	20	8.00	230	0.72	86.96
net.jforum.view.install.InstallAction	20	8.00	675	2.12	29.63
net.jforum.dao.MySQLVersionWorkarounder	15	6.00	194	0.61	77.32
net.jforum.sso.LDAPAuthenticator	14	5.60	83	0.26	168.67
net.jforum.view.forum.common.UserCommon	11	4.40	170	0.53	64.71
net.jforum.view.forum.common.AttachmentCommon	9	3.60	298	0.94	30.20
net.jforum.util.preferences.SystemGlobals	9	3.60	200	0.63	45.00
net.jforum.ConfigLoader	8	3.20	186	0.58	43.01
net.jforum.view.forum.common.UploadUtils	8	3.20	54	0.17	148.15
net.jforum.util.Captcha	5	2.00	125	0.39	40.00
net.jforum.util.I18n	5	2.00	196	0.62	25.51
net.jforum.C3P0PooledConnection	5	2.00	85	0.27	58.82
net.jforum.view.forum.UserAction	4	1.60	179	0.56	22.35
net.jforum.view.forum.PostAction	4	1.60	1,111	3.49	3.60
net.jforum.DataSourceConnection	4	1.60	40	0.13	100.00
net.jforum.view.admin.ConfigAction	4	1.60	123	0.39	32.52
net.jforum.util.FileMonitor	3	1.20	35	0.11	85.71
net.jforum.context.web.WebRequestContext	3	1.20	287	0.90	10.45
net.jforum.entities.UserSession	3	1.20	222	0.70	13.51
net.jforum.TestCaseUtils	3	1.20	53	0.17	56.60
net.jforum.view.install.ParseDBStructFile	2	0.80	70	0.22	28.57
net.jforum.entities.Attachment	2	0.80	65	0.20	30.77
net.jforum.view.install.ParseDBDumpFile	2	0.80	38	0.12	52.63
net.jforum.view.admin.SmiliesAction	2	0.80	93	0.29	21.51
net.jforum.repository.Tpl	2	0.80	42	0.13	47.62
net.jforum.util.legacy.commons.fileupload.DiskFileUpload	2	0.80	43	0.13	46.51
net.jforum.util.legacy.commons.fileupload.DefaultFileItemFactory	1	0.40	20	0.06	50.00
net.jforum.util.legacy.commons.fileupload.FileItem	1	0.40	27	0.08	37.04
net.jforum.util.legacy.commons.fileupload.disk.DiskFileItemFactory	1	0.40	36	0.11	27.78
net.jforum.view.admin.AdminAction	1	0.40	162	0.51	6.17
net.jforum.security.XMLPermissionControl	1	0.40	169	0.53	5.92
net.jforum.util.FileMonitor.FileMonitorTask	1	0.40	22	0.07	45.45
net.jforum.util.bbcode.BBCodeHandler	1	0.40	112	0.35	8.93
net.jforum.view.admin.LuceneStatsAction	1	0.40	137	0.43	7.30
net.jforum.tools.search.LuceneCommandLineReindexer	1	0.40	138	0.43	7.25
net.jforum.Command	1	0.40	70	0.22	14.29
net.jforum.JForumBaseServlet	1	0.40	87	0.27	11.49
net.jforum.util.legacy.commons.fileupload.DefaultFileItem	1	0.40	12	0.04	83.33
net.jforum.util.legacy.clickstream.config.ConfigLoader	1	0.40	82	0.26	12.20
Total	**250**	**100.00**	**31,865**	**100.00**	**7.85**

Table 4.3: Violation frequency in JForum classes.

[11]Lines of code (pure source code without comments and blank lines), measured with CLOC V. 1.52, http://cloc.sourceforge.net/ (Sep. 28, 2011)

[12]$V/KLoC$: #Violations per 1,000 LoC

5 An Inspection of Significant PaaS Migration Challenges

This chapter describes the major challenges of the actual migration of the guest application towards the cloud environment, applying previously obtained information such as the prior identified violations (see Chapter 3). It should be indicated in advance that the migration could not be realized completely. However, several key issues should be discussed in order to make reengineers aware of and to serve as an entry point for further investigations. Prior to that, for exemplifying the motivation and intention for executing the migration, the major motives are pointed out in the following:

(i) The CloudMIG approach promises to support the reengineer in migrating applications to a cloud environment, the Google App Engine in this case. However, an actual migration allowing the application to be deployed in the target environment has not been executed in preceding studies so far. This migration aims to serve as a proof-of-concept for the feasibility of the CloudMIG approach.

(ii) In order to enhance the basis for evaluating CloudMIG's automatic CEC violation detection capability, the application has to be migrated successfully to the desired cloud environment. This procedure, combined with run-time tests, is intended in order to bring up the full coverage of violations that are effectually existing. Therefore the manual migration serves as a supporting measure for the pursued Goal 1.2.1.

(iii) Besides a theoretical examination of the migration, which is of course the more important aspect in this context, a real execution often offers useful information that either helps to better understand theoretically obtained issues or allows for additional findings.

The remainder of this chapter describes key aspects of the migration efforts, regarding the latter point. As the enormous amount of detected CEC violations in JForum and its libraries of 250 and 4,716, respectively already indicates, extensive manual reengineering is required to migrate JForum to the App Engine. Some of the attempts could not show successful results, nevertheless their efforts provide interesting findings concerning major problems, which are presented in the following:

- The most significant incompatibility of JForum regarding a migration to Google's App Engine is the different database model. The App Engine does not provide relational databases, such as MySQL, PostgreSQL, or similar implementations, which are utilized by JForum as its persistence layer. JForum on the other hand uses the JDBC driver to connect to the database server and has no support for NoSQL-based databases, that would offer using, for example, Google's DataStore service. This is considered a paradigm-breaking incompatibility that affects wide parts of the core logic and requires serious adjustments.

  One possible approach is to utilize an adapter that serves as an interface on an intermediate layer between the SQL-based application (using JDBC) and Google's NoSQL datastore API. Therefore, the open-source tool *jiql* that addresses this problem seemed a promising candidate. *jiql* is a Java library that primarily consists of a database engine and a JDBC driver that act together as a JDBC wrapper, as illustrated in Figure 5.1. In order to substitute the real database connection, *jiql*'s driver has to be loaded by the application instead of the original JDBC driver. Then, all connections are made through *jiql*'s JDBC interface and database requests, such as SQL statements are translated to DataStore-conform commands by the internal database engine.

  This promised to be a helpful attempt, but *jiql* could not solve the problem adequately, due to technical limitations. It offers only a limited SQL support, for example SQL data aggregation (by the keyword `MAX`), which is used in multiple SQL statements of JForum, is not provided.

  In order to be able to continue the migration of JForum, the database problem was bypassed by means of utilizing the *HSQLDB* database configured in *memory-only-mode*. Using this configuration allows for implementing a JDBC-based relational database in the App Engine, because it requires no access to the local file system.

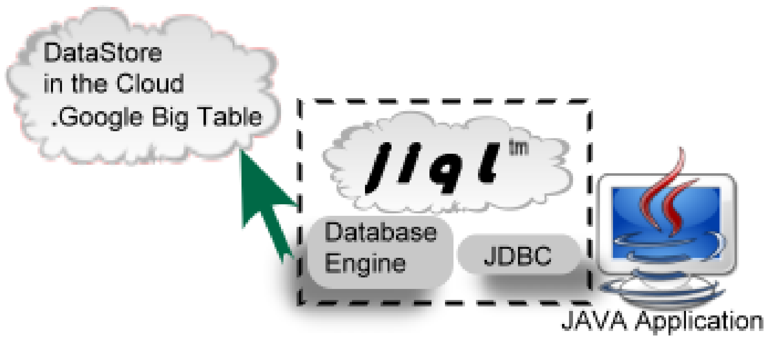


Figure 5.1: jiql between a Java application and Google's DataStore API.[1]

This is acceptable for debugging and testing purposes, but is no realistic solution for an actual migration, especially since a non-persistent database offers less reliability of data.

- The extraction of ADempiere's architectural model with the help of MoDisco could not be realized successfully. Since there exists no appropriate alternative for obtaining this model from the application by now, the migration of ADempiere was interrupted and JForum was used as the alternative candidate for the migration. Section 3.2.1 describes the taken steps for obtaining the model from ADempiere in more detail.

- The debugging process of App Engine applications is complicated, due to multiple reasons. Since the Google App Engine for Java dictates a maximum servlet response time of 30 seconds, open requests have to be answered within this time to prevent losing of information or further time-out errors. This circumstance is a potential source for seemingly existing errors during debugging activities that are caused by breakpoints or other delays arising from the debugging mechanism itself. Furthermore, Java applications that are intended to be deployed to the App Engine have to be debugged on a substitute of the App Engine, for example the web server that is included in the App Engine SDK for Eclipse. This substitute is required because the App Engine itself provides no debugging tools and therefore constitutes a risk for deviant behaviour compared to the target environment as the debugging system is only a representation of the actual App Engine.

[1]Figure taken from jiql project homepage, http://www.jiql.org/ (Oct. 10, 2011)

- The deployment interface of the App Engine provides insufficient validation of uploaded source code. This fact allows for an error-free deploying of applications that contain restricted code elements. These erroneous applications execute correctly until a restricted element is effectively loaded, which then raises an error and causes the interruption of the execution. Therefore, the reengineer is responsible for validating the conformance of the migrated application regarding the App Engine run-time environment prior to deployment.

- Another notable finding is that the reengineer does not necessarily have to adjust all the libraries to conform the target environment. Nowadays, existing applications often still use libraries that have been developed before cloud computing became prevalent. Hence, sometimes already updated versions or even App Engine adjusted implementations of these libraries are available. This situation has been the case, for example for the open-source *freemarker* library. This particular library (*freemarker Version 2.3.9*) that was developed a few years ago, was incorporated by JForum in this case. It caused several violations, due to restricted types (e.g., `javax.swing.tree.TreeNode`) and usage of the Java reflection API. Since this incompatibilities have been reported by other projects, a GAE-conform release of this library is already developed and available. Regarding JForum, the *freemarker* library could be exchanged with a fixed version that conformed the App Engine and showed no violations. This circumstance might recur in many migration scenarios, especially when dealing with open-source software, as there often exists a large and active developing communities, that have encountered and already solved the discrepancies.

6 Analysis

This chapter aims primarily to give a quantitative and qualitative analysis of the CEC violations identified during the investigation of JForum. This analysis will be described in Section 6.1. The subsequent Section 6.2 gives an evaluation of the detection capabilities of the detection mechanism of CloudMIG Xpress.

Section 6.1.1 begins with an overview of the entirety of violations determined by CloudMIG Xpress during the identification procedure described in Chapter 4. In Section 6.1.2, a template is presented, which is designed to support a structured data acquisition in the study of violations. Using this template for the following manual inspection of violations in Section 6.1.3, a selection of violations will be examined in more detail. This data collection forms the basis for further analyses regarding the detection mechanism of CloudMIG Xpress in Section 6.2.

6.1 CEC Violation Analysis

This section gives a detailed analysis of the CEC violations considered during the migration. In this context we will generally distinguish between two main categories of violations:

(i) *Automatically identified violations* that have been detected by applying CloudMIG Xpress, as described in Chapter 4 and

(ii) *Manually identified violations* that have been determined by manual inspection (see Section 6.1.3).

At the beginning of Section 6.1.1 all violations of the first category will be gathered and compared quantitatively to get an overview of the amount and structure of violations. It will be differentiated as to whether the violations originate from JForum's sources or its third-party libraries. The comparison will further be divided into the constraint severities and the different CEC types.

At this point there is still no certainty that the violations detected by CloudMIG Xpress are actually violations. The verification of whether the violations of the first category are classified correctly is done by the manual inspection of violations in Section 6.1.3.

The second category, the manually identified violations, represents the set of actually existing violations of JForum regarding the App Engine environment. Here it should be noted that violations of the second category will be considered primarily during the analysis of violations of the first category with the intention of their verification. Correctly identified violations are also referred to as *true positives*.

A complete investigation of existing violations not included in the first category (this is referred to as the set of *false negatives*) has not been performed in this thesis for reasons of scope, but is proposed as future work. An entire set of violations of the second category would allow identification of further statistical measures such as *sensitivity* (hit rate) or *false negative rate* (miss rate). In addition to a fully conducted migration, including the fixing of all CEC violations, extensive functional tests with a maximized coverage would have to be applied to the migrated application in the target runtime environment to ensure that no undiscovered violations are left.

6.1.1 Overview and Categorization of Violations

A first basic quantitative examination of all violations detected by CloudMIG Xpress is shown in Table 6.1. This aggregation displays the own sources of JForum as well as the third-party libraries separately. The violations are grouped according to severity (see Section 4.1) and the corresponding CEC types, which are their main differentiating attributes. Some particular values and relations with significant informative relevance are pointed out in the following:

(i) There are no violations with *Breaking* severity. Currently, the only CEC with *Breaking* severity is the *LanguageConstraint*. Since JForum is written in Java, it conforms to the language support of the App Engine for Java.

(ii) The clearly larger number of violations lies within the libraries ($\simeq 95\%$). The ratio of the violations of the JForum sources and the libraries is $^{250}/_{4,466} \simeq {}^{1}/_{18}$.

	Violations					
	JForum		Libraries		Sub Total	
Severity / Constraint Type	#	%	#	%	#	%
Critical	**173**	**69.20**	**4,466**	**100.00**	**4,639**	**98.37**
TypesWhiteList	140	56.00	4,380	98.07	4,520	95.84
TypesInstantiation	3	1.20	76	1.70	79	1.68
FilesystemAccess (Write)	24	9.60	-	-	24	0.51
Reflection	6	2.40	9	0.20	15	0.32
SocketOpening	-	-	1	0.02	1	0.02
Warning	**77**	**30.80**	**-**	**-**	**77**	**1.63**
FilesystemAccess (Read)	76	30.40	-	-	76	1.61
Method Call (SystemExit)	1	0.40	-	-	1	0.02
Total	**250**	**100.00**	**4,466**	**100.00**	**4,716**	**100.00**

Table 6.1: Overview of CEC violations detected in JForum (inclusive libraries).

(iii) Violations of *Critical* severity are remarkably more strongly represented than *Warning*. In JForum's own sources the percentage is 69.20% for *Critical* violations compared to 30.80% for *Warning*. In the libraries, only *Critical* violations were detected. The overall *Critical* / *Warning* percentages are 98.37% / 1.63%, which is a ratio of $^{60}/_{1}$.

(iv) *TypesWhiteListConstraint* is, as expected, the dominant CEC type causing violations both in the JForum's own sources as well as in the libraries. They mark percentages of $^{140}/_{250} \simeq 56\%$ in JForum's own sources, $^{4,380}/_{4,466} \simeq 98\%$ in the libraries, and $^{4,520}/_{4,716} \simeq 96\%$ collectively. This is due to the numerous instances that this constraint forms, since it is produced by a high number of different types that are not covered by the type white-lists of the GAE.

Table 6.2 provides a detailed perspective of the distribution of violations over the classes of JForum's own sources.[1] Only classes comprising at least one violation are included, while violation-free classes are omitted. The quota of JForum classes and the corresponding lines of code (LoC) that raised violations is 40 (6,136 LoC) from a total of 340 classes (31,865 LoC), which is a percentage of $\simeq 11.76\%$ (19.26%). The violations are grouped according to severity and the underlying CEC type.

[1]A diagram that illustrates the distribution is given in Appendix C

Considering the distribution (also with regard to Table 4.3 on page 48) there are some remarkable findings that are pointed out in the following:

(i) One single class (*ImageUtils*) comprises 68 of 250 violations, which is the biggest part of all violations by far with a percentage of $\simeq 27.20\%$. Considering the *Critical* violations exclusively, the quotient is still higher with about $^{65}/_{173} \simeq 37.57\%$, though this class only consists of 165 LoC, which constitutes $\simeq 0.52\%$ of overall LoC. This class has a high $^{V}/_{KLoC}$ ratio of $^{68}/_{165} \simeq 412.12$ compared to an overall average of $^{250}/_{31,865} \simeq 7.85$.

(ii) Grouping the four highest-ranked classes, which represent only $^{4}/_{340} \simeq 1.18\%$ of all classes and $^{1,264}/_{31,865} \simeq 3.97\%$ of JForum's LoC results in a summarized fraction of $^{123}/_{250} \simeq 49.20\%$ overall and $^{105}/_{173} \simeq 60.69\%$ of the *Critical* violations.

Table 6.3 describes the quantitative structure of violations in the libraries used by JForum.[2] As with the class analysis before, only libraries with at least one violation are displayed. Eleven ($\simeq 37\%$) of the 30 used libraries show no violations at all and are therefore omitted. Due to partially missing data regarding the libraries' size in terms of lines of code, no corresponding quotas can be stated here. The table partitions the violations into CEC types which establishes the following distribution characteristics:

(i) All discovered violations have *Critical* severity.

(ii) The distribution of violations is very unbalanced. 3,110 of 4,466 ($\simeq 70\%$) violations fall upon only 2 of 30 ($\simeq 7\%$) of the libraries. Considering the highest ranked library *jboss-jmx* solely, it covers 51.43% of all violations. Added up with *jgroups-all-2.2.9-beta2*, these two classes comprise - excluding the single *SocketOpeningConstraint* violation - more than half of the violations of every individual CEC type.

[2] A diagram that illustrates the distribution is given in Appendix C

	#Violations									
	Critical					Warning			Class Total	
Class	TypesWhiteList	FilesystemAccess (Write)	Reflection	TypesInstantiation	Critical Sub Total	FilesystemAccess (Read)	MethodCall (SystemExit)	Warning Sub Total	#	%
net.jforum.util.image.ImageUtils	65	-	-	-	65	3	-	3	**68**	**27.20**
net.jforum.util.legacy.commons.fileupload.disk.DiskFileItem	8	6	-	-	14	6	-	6	**20**	**8.00**
net.jforum.view.install.InstallAction	9	5	-	-	14	6	-	6	**20**	**8.00**
net.jforum.dao.MySQLVersionWorkarounder	9	3	-	-	12	3	-	3	**15**	**6.00**
net.jforum.sso.LDAPAuthenticator	14	-	-	-	14	-	-	-	**14**	**5.60**
net.jforum.view.forum.common.UserCommon	7	2	-	-	9	2	-	2	**11**	**4.40**
net.jforum.view.forum.common.AttachmentCommon	3	3	-	-	6	3	-	3	**9**	**3.60**
net.jforum.util.preferences.SystemGlobals	3	1	-	-	4	5	-	5	**9**	**3.60**
net.jforum.ConfigLoader	2	-	-	-	2	6	-	6	**8**	**3.20**
net.jforum.view.forum.common.UploadUtils	6	2	-	-	8	-	-	-	**8**	**3.20**
net.jforum.util.Captcha	5	-	-	-	5	-	-	-	**5**	**2.00**
net.jforum.util.I18n	-	-	-	-	-	5	-	5	**5**	**2.00**
net.jforum.C3P0PooledConnection	-	-	5	-	5	-	-	-	**5**	**2.00**
net.jforum.view.forum.UserAction	-	-	-	-	-	4	-	4	**4**	**1.60**
net.jforum.view.forum.PostAction	-	-	-	-	-	4	-	4	**4**	**1.60**
net.jforum.DataSourceConnection	4	-	-	-	4	-	-	-	**4**	**1.60**
net.jforum.view.admin.ConfigAction	2	-	-	-	2	2	-	2	**4**	**1.60**
net.jforum.util.FileMonitor	-	-	-	2	2	1	-	1	**3**	**1.20**
net.jforum.context.web.WebRequestContext	-	1	-	-	1	2	-	2	**3**	**1.20**
net.jforum.entities.UserSession	3	-	-	-	3	-	-	-	**3**	**1.20**
net.jforum.TestCaseUtils	-	-	-	-	-	3	-	3	**3**	**1.20**
net.jforum.view.install.ParseDBStructFile	-	-	-	-	-	2	-	2	**2**	**0.80**
net.jforum.entities.Attachment	-	-	-	-	-	2	-	2	**2**	**0.80**
net.jforum.view.install.ParseDBDumpFile	-	-	-	-	-	2	-	2	**2**	**0.80**
net.jforum.view.admin.SmiliesAction	-	1	-	-	1	1	-	1	**2**	**0.80**
net.jforum.repository.Tpl	-	-	-	-	-	2	-	2	**2**	**0.80**
net.jforum.util.legacy.commons.fileupload.DiskFileUpload	-	-	-	-	-	2	-	2	**2**	**0.80**
net.jforum.util.legacy.commons.fileupload.DefaultFileItemFactory	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.util.legacy.commons.fileupload.FileItem	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.util.legacy.commons.fileupload.disk.DiskFileItemFactory	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.view.admin.AdminAction	-	-	-	1	1	-	-	-	**1**	**0.40**
net.jforum.security.XMLPermissionControl	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.util.FileMonitor.FileMonitorTask	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.util.bbcode.BBCodeHandler	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.view.admin.LuceneStatsAction	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.tools.search.LuceneCommandLineReindexer	-	-	-	-	-	-	1	1	**1**	**0.40**
net.jforum.Command	-	-	1	-	1	-	-	-	**1**	**0.40**
net.jforum.JForumBaseServlet	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.util.legacy.commons.fileupload.DefaultFileItem	-	-	-	-	-	1	-	1	**1**	**0.40**
net.jforum.util.legacy.clickstream.config.ConfigLoader	-	-	-	-	-	1	-	1	**1**	**0.40**
Total	140	24	6	3	**173**	76	1	**77**	**250**	**100**

Table 6.2: Overview of CEC violations detected in JForum (w/o libs) per class.

	Violations									
	TypesWhiteList		TypesInst.		Reflection		SocketOp.		Sub Total	
Library	#	%	#	%	#	%	#	%	#	%
jboss-jmx	2,291	52.31	1	1.32	5	55.56 -	-	-	2,297	51.43
jgroups-all-2.2.9-beta2	764	17.44	49	64.47	-	-	-	-	813	18.20
jboss-system	273	6.23	3	3.95	1	11.11	-	-	277	6.20
log4j-1.2.12	253	5.78	6	7.89	-	-	-	-	259	5.80
jcaptcha-all-1.0-RC2.0.1	198	4.52	-	-	-	-	-	-	198	4.43
jboss-cache-1.2.4	175	4.00	4	5.26	-	-	-	-	179	4.01
junit	134	3.06	-	-	-	-	-	-	134	3.00
htmlparser-1.5	111	2.53	2	2.63	-	-	-	-	113	2.53
postgresql-8.0-313.jdbc3	31	0.71	-	-	2	22.22	-	-	33	0.74
freemarker-2.3.9	28	0.64	-	-	-	-	-	-	28	0.63
mysql-connector-java-5.0.3-bin	25	0.57	-	-	-	-	1	100.00	26	0.58
activation	26	0.59	-	-	-	-	-	-	26	0.58
jboss-common	23	0.53	-	-	1	11.11	-	-	24	0.54
ojdbc14	16	0.37	2	2.63	-	-	-	-	18	0.40
mail	17	0.39	-	-	-	-	-	-	17	0.38
quartz-1.5.1	9	0.21	5	6.58	-	-	-	-	14	0.31
lucene-core-2.2.0	6	0.14	1	1.32	-	-	-	-	7	0.16
ehcache-1.1	-	-	2	2.63	-	-	-	-	2	0.04
concurrent	-	-	1	1.32	-	-	-	-	1	0.02
Total	**4,380**	**100.00**	**76**	**100.00**	**9**	**100.00**	**1**	**100.00**	**4,466**	**100.00**

Table 6.3: Overview of CEC violations detected in JForum per library.

6.1.2 CEC Violation Inspection Template

A huge number (4,716) of violations has been identified performing the violation detection with CloudMIG Xpress. To allow a homogeneous and clearly represented analysis of the violations in Section 6.1.3, a template is introduced here. An empty template is shown in Figure 6.1. The fields of the template that will describe the inspected violations are explained in the following:

- **CEC Violation ID:** Each violation gets a unique ID. This ID consists of the prefix “V-” and the number that represents the sequence the violations have been saved after identification during the recognition by CloudMIG Xpress.

- **CEC Type:** The type of the violation that corresponds to the underlying Cloud Environment Constraint is specified here. A definition of CECs is given in the GAE cloud profile by the CloudMIG approach (see Listing 3.1). An excerpt of CEC types that are relevant for this analysis is considered in Section 4.1.

- **Violation Severity**: The severity (see Section 6.1.1) of the violation according to the

CEC Violation ID:	V-#
CEC Type:	...
Violation Severity:	Warning / Critical / Breaking
Source Location:	...
Violation Context:	...
Validation Correctness:	Correct / False
Justification / Appraisal:	...

Figure 6.1: Template for CEC violation inspection.

constraint type of the violation is given here.

- **Source Location**: The location of a violation is described by the class in the source code of JForum, where the Constraint Validator has determined the origin of the raised violation.
- **Violation Context**: In order to get a better understanding of the situation, this point outlines the semantic context in which the violation cause is situated.
- **Validation Correctness**: The correctness of the violation will be investigated and it will be ascertained as to whether the detected violation has been identified correctly by CloudMIG Xpress or not.
- **Justification / Appraisal:** Reasons that explain the violation relevance and verify the correctness, for example, IDE validation support, code reviews, and tests.

6.1.3 Manual Inspection of Violations

In this section the manual inspection of violations will be conducted and results will be recorded with the help of the inspection template. Due to the enormous amount of 4,716 violations, an execution of the inspection procedure is not possible for the entire set of violations at this point. However, all 250 violations of JForum's own sources and several additional violations of its libraries have been inspected. The results of the manual inspection including a critical evaluation of CloudMIG's violation detection capabilities is presented in Section 6.2. The remainder of this section addresses several violation inspections and tries to select adequate violations that cover representative sets of violations to allow a maximum of generic analysis regarding the *Validation Correctness* in particular. The fact that many violations are of the same *CEC Type* and furthermore situated in a similar *Violation Context* and *Source Location* allows for the assumption of conclusions concerning groups of related violations.

Manual Inspection of Violation V-13

CEC Violation ID:	V-13
CEC Type:	FilesystemAccessConstraint (Write)
Violation Severity:	Critical
Source Location:	net.JForum.util.legacy.commons.fileupload.disk.DiskFileItem
Violation Context:	Every request that triggers handling of a servlet multipart content creates temporary files (e.g., picture or attachment file uploads). The *finalize()* method is called by the garbage collection when there is no existing reference to the file object left.
Validation Correctness:	Correct
Justification / Appraisal:	Manual code review (see Listing 6.1) verified that this method calls *File.io.delete()*, which is not allowed in the App Engine Java Environment by definition. Temporary files, as well as uploaded files in JForum, are saved in the web server's local file system. This data management is conceptually not supported by the App Engine.

Table 6.4: Inspection of CEC violation V-13.

```
package net.jforum.dao; [...]
public class MySQLVersionWorkarounder { [...]
  /**
   * Removes the file contents from the temporary storage.
   */
  protected void finalize() {
    File outputFile = dfos.getFile();
    if (outputFile != null && outputFile.exists()) {
        outputFile.delete();
    }
  }
}
```

Listing 6.1: Source code location excerpt of CEC violation V-13.

Manual Inspection of Violation V-35

CEC Violation ID:	V-35
CEC Type:	FileSystemAccess (Read)
Violation Severity:	Warning
Source Location:	net.JForum.entities.Attachment
Violation Context:	The *hasThumb()* method of the *Attachment* type tests whether thumbnails for picture attachments in posts are enabled in the global configuration of JForum and if the actual attachment has a physical thumbnail file in the local file system.
Validation Correctness:	Correct
Justification / Appraisal:	Manual code review (see Listing 6.2) verifies the instantiation of a *java.io.File* in a local file system folder for attachments, which is set in the global configuration of JForum. Instantiating a file object in the local file system is not allowed in the App Engine.

Table 6.5: Inspection of CEC violation V-35.

```java
package net.jforum.entities; [...]
public class Attachment { [...]
  public boolean hasThumb() {
    return SystemGlobals.getBoolValue(
       ConfigKeys.ATTACHMENTS_IMAGES_CREATE_THUMB)
       && new File(SystemGlobals.getValue(
          ConfigKeys.ATTACHMENTS_STORE_DIR)
          + '/'
          + this.info.getPhysicalFilename() + "_thumb").exists();
  }
}
```

Listing 6.2: Source code location excerpt of CEC violation V-35.

Manual Inspection of Violation V-101

CEC Violation ID:	V-101
CEC Type:	MethodCallConstraint (SystemExit)
Violation Severity:	Warning
Source Location:	net.JForum.tools.search.LuceneCommandLineReindexer
Violation Context:	The *printUsage()* method is part of an administrative tool to manage database indexing. The administrator of the JForum deployment executes this as a command line program. If the arguments are not properly passed, a help message is displayed that shows how to pass the arguments. Afterwards, the program exits with *System.exit()*.
Validation Correctness:	Correct
Justification / Appraisal:	Manual code review (see Listing 6.3) verifies the call of a *System.exit()*. This tool is designed to be executed on a shell aiming to re-index the SQL database of the JForum deployment. Usage of this tool is inappropriate in the App Engine context anyway.

Table 6.6: Inspection of CEC violation V-101.

```java
package net.jforum.tools.search; [...]
public class LuceneCommandLineReindexer { [...]
  private void printUsage() {
    System.out.println("\nUsage: LuceneCommandLineReindexer \n"
       + " --path full_path_to_JForum_root_directory \n"
       + " --type {date|message} \n"
       + " --firstPostId a_id \n"
       + " --lastPostId a_id \n"
       + " --fromDate dd/MM/yyyy \n"
       + " --toDate dd/MM/yyyy \n"
       + " [--recreateIndex]\n"
       + " [--avoidDuplicatedRecords]");
    System.exit(1);
  }
}
```

Listing 6.3: Source code location excerpt of CEC violation V-101.

Manual Inspection of Violation V-102

CEC Violation ID:	V-102
CEC Type:	TypesInstantiationConstraint
Violation Severity:	Critical
Source Location:	net.JForum.view.admin.AdminAction
Violation Context:	The functionality to listen for POP e-mails starts a new thread that runs in the background.
Validation Correctness:	Correct
Justification / Appraisal:	Manual code review (see Listing 6.4) verifies the instantiation of a *java.lang.Thread*. Instantiating a new thread is not allowed in the App Engine.

Table 6.7: Inspection of CEC violation V-102.

```java
package net.jforum.view.admin; [...]
public class AdminAction extends Command { [...]
  public void fetchMail() throws Exception {
    new Thread(new Runnable() {
      public void run() {
        try {
          new POPListener().execute(null);
        }
        catch (Exception e) {
          e.printStackTrace();
        }
      }
        }).start();
    this.main();
  }
}
```

Listing 6.4: Source code location excerpt of CEC violation V-102.

Manual Inspection of Violation V-109

CEC Violation ID:	V-109
CEC Type:	ReflectionConstraint
Violation Severity:	Critical
Source Location:	net.JForum.Command
Violation Context:	The *Command* type is an abstract class that is extended by all functional classes that process presentation actions. Reflection is used to invoke a dynamically determined method of the *Command* extending class.
Validation Correctness:	False[3]
Justification / Appraisal:	Manual code review (see Listing 6.5) showed that this is not an actual violation. Depending on what module is loaded when executing the *service()* method of the corresponding servlet, the appropriate method for the requested action is determined and invoked by the reflection method *java.lang.reflect.Method.invoke()*. The reflection target is conceptually its own class, as the class is defined by module-mapping properties.

Table 6.8: Inspection of CEC violation V-109.

```
package net.jforum; [...]
public abstract class Command { [...]
  public Template process(RequestContext request,
     ResponseContext response, SimpleHash context) {
        this.request = request;
        this.response = response;
        this.context = context;
        String action = this.request.getAction();
        if (!this.ignoreAction) {
      try {
        this.getClass().getMethod(action, NO_ARGS_CLASS).
           invoke(this, NO_ARGS_OBJECT);
      }
      catch (NoSuchMethodException e) {
        this.list();
      }
      catch (Exception e) {
        throw new ForumException(e);
      }
    } [...]
  }
}
```

Listing 6.5: Source code location excerpt of CEC violation V-109.

[3]A further discussion of this specific violation and the violations regarding ReflectionConstraint in general is presented in Section 6.2.

Manual Inspection of Violation V-117

CEC Violation ID:	V-117
CEC Type:	TypesWhiteListConstraint
Violation Severity:	Critical
Source Location:	net.JForum.util.Captcha
Violation Context:	The *Captcha* module uses constants of the *java.awt.Color* type to generate graphical elements in Captchas.
Validation Correctness:	Correct
Justification / Appraisal:	Manual code review (see Listing 6.6) verified the usage of *java.awt.Color*. This type is not available in the Java App Engine environment.

Table 6.9: Inspection of CEC violation V-117.

```java
package net.jforum.util; [...]
public class Captcha extends ListImageCaptchaEngine { [...]
  protected void buildInitialFactories() {
    this.initializeChars();
      this.backgroundGeneratorList = new ArrayList();
      this.textPasterList = new ArrayList();
      this.fontGeneratorList = new ArrayList();
      int width = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_WIDTH);
      int height = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_HEIGHT);
      int minWords = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MIN_WORDS);
      int maxWords = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MAX_WORDS);
      int minFontSize = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MIN_FONT_SIZE);
      int maxFontSize = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MAX_FONT_SIZE);
      this.backgroundGeneratorList.add(new GradientBackgroundGenerator(
         new Integer(width), new Integer(height), Color.PINK, Color.LIGHT_GRAY));
      [...]
  }
}
```

Listing 6.6: Source code location excerpt of CEC violation V-117.

Manual Inspection of Violation V-2081

CEC Violation ID:	V-2081
CEC Type:	SocketOpeningConstraint
Violation Severity:	Critical
Source Location:	com.mysql.jdbc.NamedPipeSocketFactory.NamedPipeSocket
Violation Context:	In the Java MySQL Connector Library, a connection to the MySQL Server is initialized by instantiating a *Socket* directly to the server.
Validation Correctness:	Correct
Justification / Appraisal:	Manual code review (see Listing 6.7) verifies the instantiation of a *java.net.Socket*. Instantiating a socket is not allowed in the App Engine.

Table 6.10: Inspection of CEC violation V-2081.

```java
package com.mysql.jdbc; [...]
public class NamedPipeSocketFactory implements SocketFactory { [...]
  private Socket namedPipeSocket;
  public Socket connect(String host, int portNumber /* ignored */,
     Properties props) throws SocketException, IOException {
    String namedPipePath = props.getProperty(NAMED_PIPE_PROP_NAME);
    if (namedPipePath == null) {
      namedPipePath = "\\\\.\\pipe\\MySQL"; //$NON-NLS-1$
    } else if (namedPipePath.length() == 0) {
      throw new SocketException(Messages
          .getString("NamedPipeSocketFactory.2") //$NON-NLS-1$
          + NAMED_PIPE_PROP_NAME
          + Messages.
             getString("NamedPipeSocketFactory.3")); //$NON-NLS-1$
    }
    this.namedPipeSocket = new NamedPipeSocket(namedPipePath);
    return this.namedPipeSocket;
  }
}
```

Listing 6.7: Source code location excerpt of CEC violation V-2081.

6.2 Evaluation of CloudMIG's Violation Detection Capabilities

The automatic violation detection mechanism is an important core component of the CloudMIG approach. This procedure relies on models which are described in Chapter 3. The precise configuration of the cloud profile in particular, where constraints are defined, is a decisive factor influencing the output of the violation detection phase. Based upon these constraint definitions, the validators check the architectural model of the considered application for existing violations. When a manual migration including conformance tests is completed successfully, two sets of violations exist: the statically analyzed violation set produced by CloudMIG and the set of violations that were discovered during the migration and tests. These sets have to be compared in terms of coverage and structure of violations contained in order to identify reasons for potential differences, such as insufficiencies of the CEM constraint classes (e.g., regarding the constraint validators).

Since a constraint either holds or fails, the validation process is considered a binary classification test. Hence, a reasonable approach to evaluate the quality of CloudMIG's detection mechanism is to calculate measures that describe the performance of a binary classification test.

Precision

As described before (see Section 6.1), a full migration has not been executed for the reasons mentioned. Due to this fact, the analysis will not cover further qualitative indicators such as *miss rate*, *sensitivity (recall)*, *specificity* or prevalent combined measures, which rely on the former results.

We will focus on the determination of the *precision*, also referred to as *positive predictive value* (*PPV*), restricted to the violations that have been identified in JForum's own sources. A complete list of all 250 violations in JForum including their underlying CEC type, the severity and a short description is given in Appendix B.

The result of the violation detection can be divided into two disjoint sets of violations. Violations that are both identified by the detection mechanism as well as verified to be correct by manual inspection are referred to as *true positives* (t_p). Those violations that have been identified by mistake and proven to be wrong by manual inspection constitute the set of *false positives* (f_p). Combined, the two sets form the complete set of violations.

The *precision* represents the quota of violations that are detected correctly (t_p) compared to the complete set of detected violations (t_p+f_p), hence:

$$Precision = \frac{t_p}{t_p + f_p}.$$

Apart from one exception, which will be considered in particular, the manual inspection of the complete set of violations that CloudMIG Xpress has detected in JForum's own sources proved to be identified throughout correctly. The manual inspection process including several exemplary inspections of different violation types is described in Section 6.1.3. The violation V-109 (see page 65), which is based on the underlying ReflectionConstraint, is the only violation that is identified as *False*. This is rather due to an incorrect interpretation of the vague information about restrictions of the App Engine's JVM that is incorporated in the constraint definition of the cloud profile than to an erroneous validation mechanism. The description of the sandbox environment included in the on-line documentation of the App Engine[4] indicates that reflection is only allowed for "own" classes. Both the deployment as well as run-time tests showed that reflection of classes that belong to deployed libraries is also feasible. Thus, assuming that an adjusted constraint definition in the cloud profile would correct those reflection violations, the precision of CloudMIG's violation detection mechanism regarding JForum's own sources shows a percentage of 100%.

Improvement Potentials

Besides calculable measures, several findings that result from the manual inspection are presented here. Two areas of potential improvement and the motivations behind the manual inspection of CloudMIG and the proposed tool CloudMIG Xpress are pointed out in the following:

- Until now, solely static violation detection has been provided by CloudMIG Xpress. In order to improve both the accuracy and the coverage of its violation detection mechanism, dynamic analyses need to be appended. For example, potential violations that reside in *dead code* and hence will not raise an error in the target environment are not

[4]GAE documentation: http://code.google.com/intl/de-DE/appengine/docs/java/runtime.html#The_Sandbox (October 14, 2011)

distinguished from those that actually cause violations at target run-time. Another aspect affects the level of quality that the violation detection offers for reengineers in locating and fixing the identified violations. For example, calling a restricted method that indicates a violation has to be traced back in a debugging activity to find out which component is the responsible caller of the method. A dynamic analysis could identify possible scenarios, control flows, and stack traces leading to the violation. Further, violations based on time-relevant constraints, such as maximal response times for web requests, are not identifiable by exclusively static analyses.

- It is recommended, that the responsible element or call that causes a violation be supplied with enhanced additional information (e.g., explicit line in the source code, variable name, and calling component) to the violation indication (besides its underlying CEC type and containing class) in order to help the reengineer to locate the exact code location more efficiently. The incorporation of this improvement in CloudMIG Xpress is already intended in the near future.

- Violations are detected by different independent validators. Under specific circumstances multiple violations are raised for the same element. For example, the statement `<import java.io.FileOutputStream;>` raises one *FilesystemAccess (Write)* violation and also one *TypesWhiteList (Import)* violation. In order to yield more precise quantitative measures of detected violations and to provide better support for reengineers, those violations should be combined. They could be counted as one violation, as they arise from the same statement, but attached with additional information that represents the different violation causes.

Another issue concerning a potential source for *false negatives* was noticed during the inspection of violations. The instantiation of types outside of methods or constructors (of a Java class) is, in contrast to those inside, assigned to the external model (as API and library elements) of the application's KDM model during the model extraction process of MoDisco. This leads to undiscovered violations, unless particular attention is given by the violation detection component.

7 Related Work

When a software system is considered for migration to a cloud environment - for whatever reason - the required effort and cost are generally important items affecting the decision of whether the migration project is both feasible and rational. Traditional approaches for software effort estimation are not quite appropriate because of the different circumstances of software development and reengineering regarding the cloud computing domain.

Tran et al. (2011a) investigate the migration of a sample application (*.Net PetShop*, the .Net counterpart of *Java PetStore*) to Microsoft's Azure cloud platform. They performed a manual migration for the application as a case study to define a taxonomy of the examined migration tasks with additional attention to distinguishing the cost-effective factors between different task categories. The identification and definition of typical migration tasks, including a Function Point-based analysis of estimated efforts and recorded overheads, gives an overview of potential migration challenges. Although this case study is an interesting experience report, it provides no systematical methodology for supporting a reengineer with the migration process.

A new approach, especially for size estimation of cloud migration projects is also presented by Tran et al. (2011b). Their methodology is called *Cloud Migration Point* (CMP) and is based on the established software size estimation model *Function Point*. They showed the CMP model to provide reliable size estimations. Therefore, they evaluated their model both theoretically by means of size metric validation as well as empirically by cross-validating several small-scale projects. Though CMP delivers significant information about a migration project's effort, it is not appropriate for supporting a reengineer who plans and practically conducts the migration in identifying and handling the technical reasons for the effort.

Khajeh-Hosseini et al. (2011b) present two tools: the first tool supports the estimation of costs that would result from a software migration to the cloud, and the second tool helps to investigate potential benefits and risks regarding a migration to an IaaS-based cloud. Case studies showed that both tools were able to assist the user in obtaining useful information to contemplate the migration decision.

The related paper Khajeh-Hosseini et al. (2011a) discusses the challenges of organizations' decision makers when trading-off a cloud migration. They consider not only the cost estimation, but also further aspects (referred to as socio-technical factors) that influence the decision. Furthermore, they present a collection of tools and a framework for organizing the decision process (*Cloud Adoption Toolkit*) where the provided tools are matched to the according concerns of the focused activity.

Zhou et al. (2010) present an ontology-based approach to support enterprise software migration to the cloud. It aims at understanding and decomposing a considered software system into cloud-conform service candidates by an ontology development process. Therefore, an initial ontology of the legacy software system is built by means of reverse engineering and model transformation techniques. This ontology, representing concepts and relations of the system, is used to identify potential service candidates that are applicable for cloud environments.

8 Conclusion

This chapter presents the conclusion of this thesis. First, a summary of the previous chapters is given in Section 8.1. The discussion in Section 8.2 emphasizes main issues and findings of this thesis before Section 8.3 proposes future work.

8.1 Summary

Cloud computing, software migration, and the CloudMIG approach were the main foundations required for our investigations regarding the migration of software systems to PaaS-based cloud environments and have been described in the beginning of this thesis. In addition, the involved technologies including different cloud platforms (Eucalyptus, Google App Engine, and AppScale), migration candidates (ADempiere and JForum), and supporting tools, for example, MoDisco for model extraction and CloudMIG Xpress for violation detection, have been presented. We constructed the essential models proposed by the CloudMIG approach and performed a critical analysis of the KDM extraction process. With the help of CloudMIG Xpress, these models were investigated for CEC violations. The resulting set of violations detected by CloudMIG Xpress has been presented and saved for further analyses. In addition to the theoretical examination, we conducted the migration of JForum to some extent for pointing out significant migration challenges that occurred in this specific case. Based upon the set of violations provided by CloudMIG Xpress, several quantitative analyses of the detected violations, grouped according to violation severity or underlying constraint type are presented. In order to evaluate the violation detection capabilities of CloudMIG, and the correctness of CloudMIG's constraint validators in particular, the detected violations have been subject to a manual inspection. For this purpose, a template has been defined that allowed for a uniform examination of a selection of violations.

8.2 Discussion

This thesis aimed at investigating the migration of software systems to PaaS-based cloud environments. Since the CloudMIG approach was appointed to serve as the major foundation for our investigations, several essential components of CloudMIG have been subject to closer examinations and are discussed in the remainder of this section. Further, the major migration challenges that have been experienced in this context are reconsidered.

Overall, the goals stated in Section 1.2 could be achieved. Several obstacles, especially in the early stages of this thesis, required higher efforts than scheduled beforehand. For example, both the unsolvable model extraction problem with ADempiere and the huge amount and grave importance of existing violations in JForum impeded further evaluations of the completely migrated application as well as analyses of the violation detection.

Another finding that emerged during the entire period of this thesis was the rapid development of tools and technologies in the cloud migration context. For example, incorporated third-party products, such as the MoDisco framework, the AppScale platform, and Google's App Engine had new major version releases in the last few months. Furthermore, the CloudMIG approach and its proto-type tool CloudMIG Xpress are under continuing development and will hopefully obtain useful insights through this thesis.

Model Construction

The CloudMIG approach and its model-based violation detection and mapping from the application's actual architecture to the target architecture promises a reliable and flexible approach for supporting software reengineers in migrating applications to PaaS-based clouds. However, the model-extracting tool, MoDisco shows several shortcomings:

- The unsolved problem performing the extraction of ADempiere's architectural model indicates errors in MoDisco's model discoverer and ATL transformation.

- Since MoDisco utilizes the source code of an application for its model extraction, the level of detail of models of libraries without attached source is limited and might leave out relevant information. This is especially a disadvantage for proprietary libraries, which generally do not provide source code.

Either these drawbacks have to be resolved in cooperation with MoDisco's developers to match CloudMIG's requirements, or an alternative tool for the model extraction has to be established that allows for substituting MoDisco.

Violation Identification

The violation detection was executed with CloudMIG Xpress. Though it already offers helpful support, this tool is still in a prototype stage and exhibits several areas of potential improvement:

- Refined characterization of detected CEC violations, for example, with additional information about the location of the violation regarding the source code and responsible trigger of the violation.
- Sharpened definition of constraint types that provide more precise details of the underlying instance, for example, in order to distinguish between occurrences of violations as type imports, type instantiation, and return types.
- Improved presentation of detected violations, for example, by offering additional graphical views, providing basic statistics, and supporting different output formats.

Migration Challenges

One of the main problems concerning the specific migration considered in this thesis has been the different database paradigms. JForum, the application that was chosen as the migration candidate, utilizes a relational database (e.g., MySQL, PostgreSQL, Oracle), but the App Engine, representing the target cloud environment only provides Google's NoSQL-based DataStore. As discussed in Chapter 5, this circumstance requires an elaborated solution. At the end of the period of this thesis, Google announced that the App Engine for Java had been enhanced by an integrated service that allows applications to use a relational database, referred to as *Google Cloud SQL*[1]. This new service, assumed that it offers acceptable performance and scalability, highly increases the potential for migrating enterprise software systems, which often use relational databases, to the App Engine.

[1]http://code.google.com/intl/de-DE/apis/sql/ (October 13, 2011)

8.3 Future Work

A full migration, including comprehensive test coverage, is proposed as an essential future project. As described in Section 6.2, this would allow for measuring the performance of CloudMIG's violation detection by means of evaluating the quality of its validators as binary classifiers to a greater extent.

The successful migration is also the basis for performance analyses of a migrated application. The performance evaluation of migrated systems is an interesting field that represents the essential factor in decisions as to whether a migration is rational or not.

Regarding the violation detection capabilities of CloudMIG, a major goal will be to incorporate enhanced violation detection mechanisms. For example, adding dynamic analyses would extend the possibilities for implementing constraint validators that allow for identifying new violation categories, such as run-time specific violations.

Appendices

A Source Code of the Command Line Program for ATL Model-to-Model Transformation

```java
package java2kdm;
import [...]
public class Java2Kdm {

  // Absolute paths will be set as program's arguments
  private static String inModelPath = "blank.javaxmi";
  private static String outModelPath = "blank.kdm";

  public static void main(String[] args) {
    Java2Kdm main = new Java2Kdm();
    if (args.length != 2) {
      System.out
          .println("Please set input and output model"
              + "absolut filepaths as arguments");
    } else {
      // Set absolute path descriptors of input and
      // output models
      inModelPath = "file://" + args[0];
      outModelPath = "file://" + args[1];

      // Prompt starting time of the translation
      System.out.println("Transformation started at: "
          + getDateTime() + "\nPlease be patient!");

      // Display a progress bar at the console to
      // indicate that the translation is still executing
      ClassicalShellProgressBar myprogressbar =
          new ClassicalShellProgressBar();
      myprogressbar.start();

      // Set the level of the ATL Logger
      ATLLogger.getLogger().setLevel(Level.INFO);

      // Start the model to model translation with
      // path informations of input and output models
      main.transform(inModelPath, outModelPath);

      // Translation has finished, hence stop the
      // progress bar and prompt a finish message
      myprogressbar.showProgress = false;
      System.out.println("\nModel translation finished."
          + "Thank you for your patience!");
    }
  }

  /**
   * Process executing a ATL Model-to-model transformation
   *
   * @param inFilePath Absolute path to input model file
   * @param outFilePath Absolute path to output model file
   */
  public void transform(String inFilePath,
      String outFilePath) {
    try {
      // Get instances of model(de-)serializers
      IInjector injector = new EMFInjector();
      IExtractor extractor = new EMFExtractor();

      // Set up input and output meta-models for
      // translation with the specific meta-model
      // descriptions (java.ecore / kdm.ecore)
      ModelFactory factory = new EMFModelFactory();
      IReferenceModel javaMetaModel =
          factory.newReferenceModel();
      IReferenceModel kdmMetaModel =
          factory.newReferenceModel();
      injector.inject(javaMetaModel, Java2Kdm.class
          .getResource("/java.ecore").toString());
      injector.inject(kdmMetaModel, Java2Kdm.class
          .getResource("/kdm.ecore").toString());

      // Get instances of the input and output models used
      // during transformation and load (inject) input model
      IModel javaModel = factory.newModel(javaMetaModel);
      IModel kdmModel = factory.newModel(kdmMetaModel);
      injector.inject(javaModel, inFilePath);

      // Get an instance of the transformation launcher and
      // add models to the transformation context
      ILauncher launcher = new EMFVMLauncher();
      launcher.initialize(Collections
          .<String, Object> emptyMap());
      launcher.addInModel(javaModel, "IN", "java");
      launcher.addOutModel(kdmModel, "OUT", "kdm");

      // Set up options of the ATL-VM used for the
      // transformation process
      HashMap<String, Object> objects =
          new HashMap<String, Object>();
      objects.put("step", "false");
      objects.put("printExecutionTime", "true");
      objects.put("allowInterModelReferences", "true");
      objects.put("showSummary", "true");

      // Set up the resource URL of the asm definition file
      URL transformationResourceURL =
          Java2Kdm.class.getResource("/javaToKdm.asm");

      // Launch the transformation using given parameters
      launcher.launch(ILauncher.DEBUG_MODE,
          new NullProgressMonitor(), objects,
          transformationResourceURL.openStream());

      // Export the KDM formatted model to output file
      extractor.extract(kdmModel, outFilePath);

    } catch (ATLExecutionException atlEx) {
      atlEx.printStackTrace();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Get the date and time as a formatted String
   *
   * @return Returns the current date and time as a
   *         formatted String
   */
  private static String getDateTime() {
  [...]
  }
}

/**
 * A classical progress bar for displaying progressing
 * status in an ascii console
 *
 * @author sfe
 */
class ClassicalShellProgressBar extends Thread {
[...]
}
```

Listing A.1: Source code excerpt of the command line program for ATL model-to-model transformation (detailed).

B CEC Violations in JForum

ID	CEC	Severity	Description
1	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
2	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
3	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
4	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
5	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
6	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
7	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
8	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
9	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
10	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
11	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
12	FilesystemAccess (Write)	Critical	Instantiation of "java.io.FileOutputStream".
13	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
14	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
15	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
16	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
17	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
18	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
19	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
20	FilesystemAccess (Write)	Critical	Call of method "java.io.File.delete".
21	FilesystemAccess (Write)	Critical	Call of method "java.io.File.mkdir".
22	FilesystemAccess (Write)	Critical	Call of method "java.io.File.mkdirs".
23	FilesystemAccess (Write)	Critical	Call of method "java.io.File.mkdirs".
24	FilesystemAccess (Write)	Critical	Call of method "java.io.File.renameTo".
25	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
26	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
27	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
28	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
29	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
30	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
31	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
32	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
33	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
34	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
35	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
36	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
37	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
38	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
39	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
40	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
41	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
42	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
43	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
44	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
45	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileReader".

46	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
47	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
48	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
49	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileReader".
50	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
51	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
52	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
53	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
54	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
55	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
56	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
57	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
58	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
59	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
60	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
61	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
62	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
63	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
64	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
65	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
66	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileReader".
67	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
68	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
69	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
70	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
71	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
72	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
73	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
74	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
75	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
76	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
77	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
78	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
79	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
80	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
81	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
82	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
83	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileReader".
84	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
85	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
86	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
87	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
88	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
89	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
90	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
91	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
92	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
93	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
94	FilesystemAccess (Read)	Warning	Instantiation of "java.io.File".
95	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
96	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
97	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileReader".
98	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileReader".
99	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
100	FilesystemAccess (Read)	Warning	Instantiation of "java.io.FileInputStream".
101	MethodCall (SystemExit)	Warning	Call of method "java.lang.System.exit".
102	TypesInstantiation	Critical	Instantiation of "java.lang.Thread".
103	TypesInstantiation	Critical	Instantiation of "java.util.Timer".
104	TypesInstantiation	Critical	Instantiation of "java.util.Timer".
105	Reflection	Critical	Call of method "java.lang.reflect.Method.invoke".
106	Reflection	Critical	Call of method "java.lang.reflect.Method.invoke".

107	Reflection	Critical	Call of method "java.lang.reflect.Method".
108	Reflection	Critical	Call of method "java.lang.reflect.Method.invoke".
109	Reflection	Critical	Call of method "java.lang.reflect.Method.invoke".
110	Reflection	Critical	Call of method "java.lang.reflect.Method.getName".
111	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
112	TypesWhiteList	Critical	Usage of "javax.imageio.ImageIO". (Import).
113	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
114	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
115	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
116	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
117	TypesWhiteList	Critical	Usage of "java.awt.Color". (Import).
118	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
119	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
120	TypesWhiteList	Critical	Usage of "java.awt.Image". (Call).
121	TypesWhiteList	Critical	Usage of "java.awt.Image".
122	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Create).
123	TypesWhiteList	Critical	Usage of "javax.naming.Context".
124	TypesWhiteList	Critical	Usage of "java.awt.Image". (Call).
125	TypesWhiteList	Critical	Usage of "javax.naming.AuthenticationException". (Import).
126	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Import).
127	TypesWhiteList	Critical	Usage of "javax.imageio.stream.ImageInputStream". (Call).
128	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream".
129	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Create).
130	TypesWhiteList	Critical	Usage of "java.awt.Dimension". (Write).
131	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
132	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
133	TypesWhiteList	Critical	Usage of "java.awt.image.PixelGrabber". (Call).
134	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Create).
135	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
136	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
137	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
138	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream".
139	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
140	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Create).
141	TypesWhiteList	Critical	Usage of "javax.naming.directory.InitialDirContext". (Create).
142	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
143	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriter". (Call).
144	TypesWhiteList	Critical	Usage of "javax.imageio.ImageIO". (Call).
145	TypesWhiteList	Critical	Usage of "javax.naming.directory.Attribute". (Import).
146	TypesWhiteList	Critical	Usage of "javax.naming.Context". (Call).
147	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
148	TypesWhiteList	Critical	Usage of "java.awt.Image".
149	TypesWhiteList	Critical	Usage of "javax.naming.directory.InitialDirContext". (Import).
150	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Import).
151	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriteParam". (Call).
152	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Import).
153	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
154	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Read).
155	TypesWhiteList	Critical	Usage of "java.awt.Dimension". (Write).
156	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
157	TypesWhiteList	Critical	Usage of "javax.naming.AuthenticationException".
158	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
159	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
160	TypesWhiteList	Critical	Usage of "java.awt.image.ColorModel". (Call).
161	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
162	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Create).
163	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
164	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Import).
165	TypesWhiteList	Critical	Usage of "javax.imageio.ImageIO". (Import).
166	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
167	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriter". (Call).

168	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Import).
169	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
170	TypesWhiteList	Critical	Usage of "javax.imageio.stream.ImageOutputStream".
171	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Read).
172	TypesWhiteList	Critical	Usage of "javax.imageio.IIOImage". (Import).
173	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
174	TypesWhiteList	Critical	Usage of "java.util.Map.Entry". (Call).
175	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Import).
176	TypesWhiteList	Critical	Usage of "java.util.Map.Entry". (Call).
177	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
178	TypesWhiteList	Critical	Usage of "javax.imageio.IIOImage". (Call).
179	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
180	TypesWhiteList	Critical	Usage of "javax.naming.InitialContext". (Import).
181	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriteParam".
182	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
183	TypesWhiteList	Critical	Usage of "java.awt.Image". (Import).
184	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
185	TypesWhiteList	Critical	Usage of "javax.naming.directory.InitialDirContext". (Call).
186	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
187	TypesWhiteList	Critical	Usage of "javax.imageio.stream.ImageOutputStream". (Import).
188	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
189	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Import).
190	TypesWhiteList	Critical	Usage of "java.awt.Dimension".(Read).
191	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Read).
192	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
193	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Create).
194	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
195	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream".
196	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriteParam". (Import).
197	TypesWhiteList	Critical	Usage of "java.awt.Dimension".
198	TypesWhiteList	Critical	Usage of "java.awt.image.PixelGrabber".
199	TypesWhiteList	Critical	Usage of "javax.naming.directory.DirContext".
200	TypesWhiteList	Critical	Usage of "javax.naming.NamingException".
201	TypesWhiteList	Critical	Usage of "javax.naming.Context". (Import).
202	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriter". (UsesType).
203	TypesWhiteList	Critical	Usage of "java.awt.image.PixelGrabber". (Call).
204	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
205	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream".
206	TypesWhiteList	Critical	Usage of "java.util.Map.Entry". (Call).
207	TypesWhiteList	Critical	Usage of "javax.naming.directory.Attribute".
208	TypesWhiteList	Critical	Usage of "javax.imageio.stream.ImageInputStream". (Call).
209	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriter".
210	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriteParam". (Call).
211	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
212	TypesWhiteList	Critical	Usage of "javax.imageio.ImageIO". (Call).
213	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
214	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
215	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream".
216	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Import).
217	TypesWhiteList	Critical	Usage of "javax.naming.Context". (Import).
218	TypesWhiteList	Critical	Usage of "javax.naming.directory.DirContext". (Import).
219	TypesWhiteList	Critical	Usage of "java.awt.Dimension". (Import).
220	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Import).
221	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
222	TypesWhiteList	Critical	Usage of "java.awt.Image". (Import).
223	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriter". (Call).
224	TypesWhiteList	Critical	Usage of "javax.naming.NamingException". (Import).
225	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage".
226	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
227	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
228	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (UsesType).

229	TypesWhiteList	Critical	Usage of "java.util.Map.Entry". (Call).
230	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Read).
231	TypesWhiteList	Critical	Usage of "javax.naming.Context". (Call).
232	TypesWhiteList	Critical	Usage of "javax.imageio.plugins.jpeg.JPEGImageWriteParam". (Import).
233	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream".
234	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
235	TypesWhiteList	Critical	Usage of "javax.naming.NamingException".
236	TypesWhiteList	Critical	Usage of "javax.imageio.ImageIO". (Call).
237	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Create).
238	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
239	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
240	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Import).
241	TypesWhiteList	Critical	Usage of "java.awt.image.BufferedImage". (Call).
242	TypesWhiteList	Critical	Usage of "javax.imageio.ImageWriter". (Import).
243	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
244	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream".
245	TypesWhiteList	Critical	Usage of "javax.imageio.IIOImage". (Create).
246	TypesWhiteList	Critical	Usage of "java.io.FileOutputStream". (Call).
247	TypesWhiteList	Critical	Usage of "javax.imageio.ImageIO". (Import).
248	TypesWhiteList	Critical	Usage of "java.awt.Dimension". (Read).
249	TypesWhiteList	Critical	Usage of "java.awt.image.PixelGrabber". (Import).
250	TypesWhiteList	Critical	Usage of "java.util.Map.Entry". (Call).

Table B.1: CEC violations detected in JForum (w/o libs).

C Distribution Diagrams of CEC Violations in JForum and Libraries

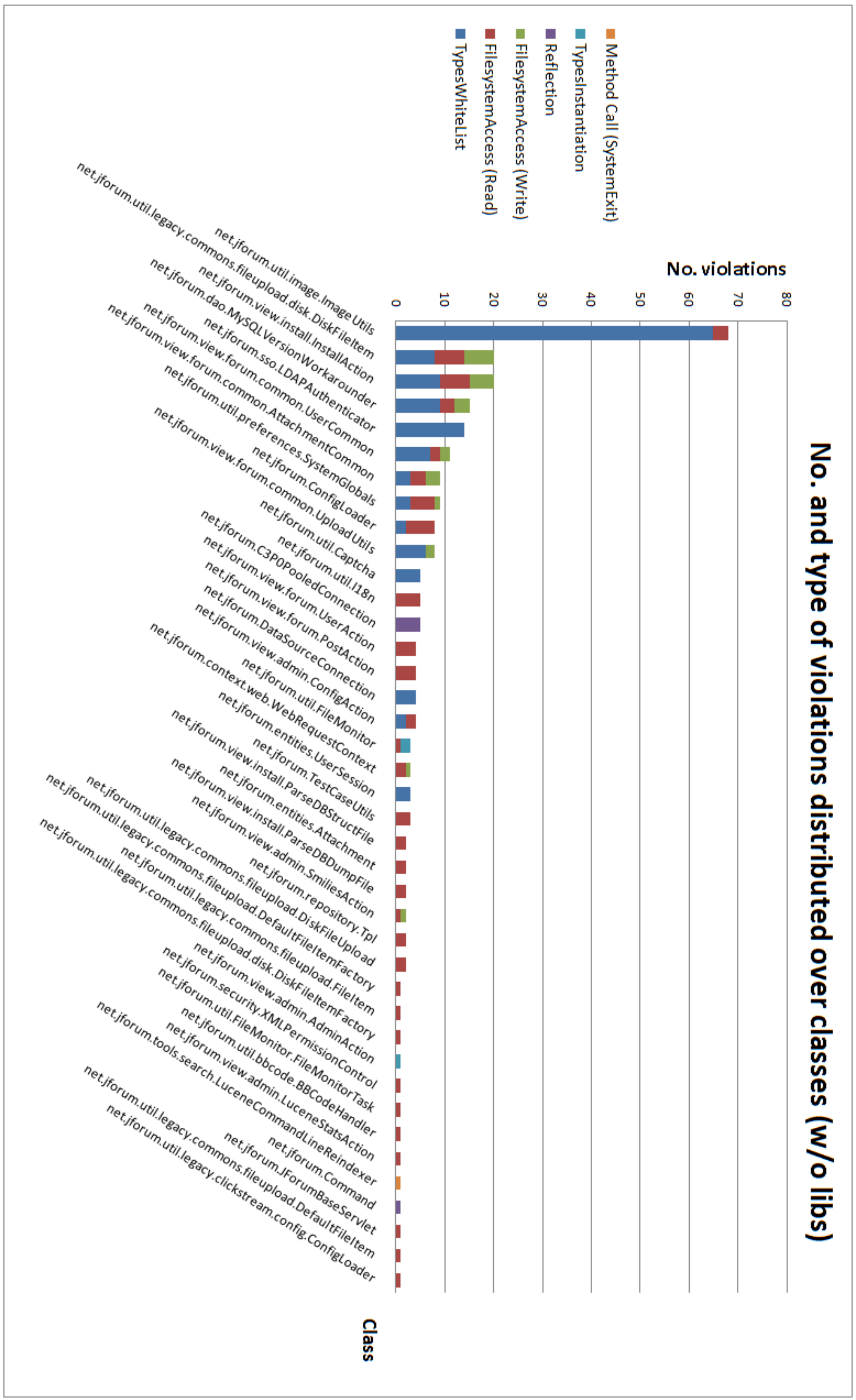


Figure C.1: Distribution diagram: CEC violations detected in JForum (w/o libs) per class.

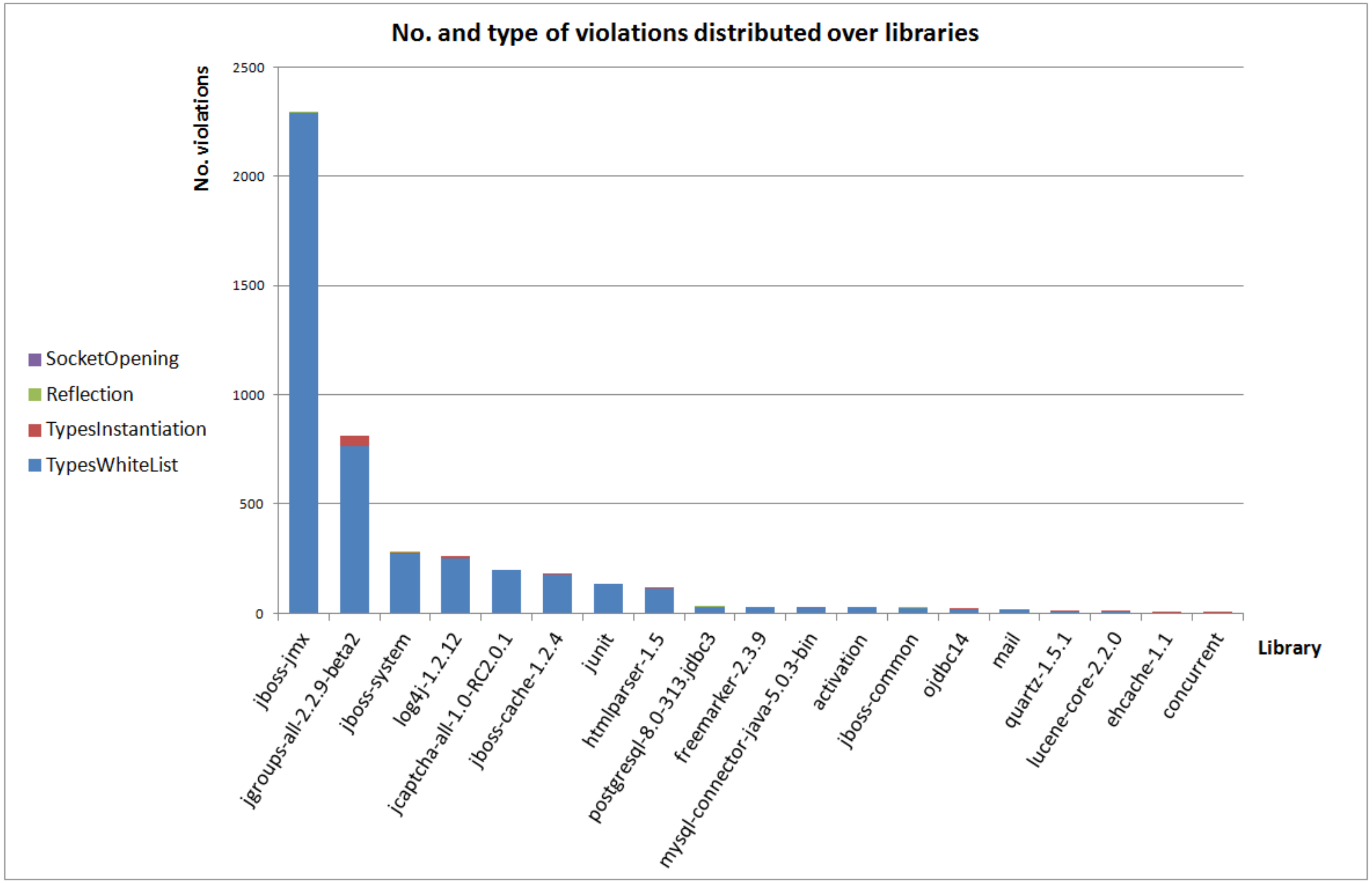


Figure C.2: Distribution diagram: CEC violations detected in JForum per library.

Bibliography

F. Allilaire, J. Bézivin, F. Jouault, and I. Kurtev. ATL - Eclipse Support for Model Transformation. In *Proc. of the Eclipse Technology eXchange Workshop (eTX) at ECOOP*, 2006.

A. A. Almonaies, J. R. Cordy, and T. R. Dean. Legacy System Evolution towards Service-Oriented Architecture. *International Workshop on SOA Migration and Evolution SOAME 2010*, pages 53–62, 2010.

M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A Berkeley View of Cloud Computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCBEECS-2009-28*, 2009.

J. Bisbal, D. Lawless, B. Wu, and J. Grimson. Legacy Information Systems: Issues and Directions. *Software, IEEE*, 16(5):103 –111, sep/oct 1999. ISSN 0740-7459. doi: 10.1109/52.795108.

M. L. Brodie and M. Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995. ISBN 1-55860-330-1.

H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. MoDisco: A Generic and Extensible Framework for Model Driven Reverse Engineering. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 173–174, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0116-9. doi: 10.1145/1858996.1859032.

C. Bunch, N. Chohan, S. Pang, M. Nagy, S. Sunil, R. Wolski, and C. Krintz. AppScale Design and Implementation. Technical report, University of California, Santa Barbara, Feb. 2009.

C. Bunch, N. Chohan, C. Krintz, J. Chohan, J. Kupferman, P. Lakhina, Y. Li, and Y. Nomura. An Evaluation of Distributed Datastores Using the AppScale Cloud Platform. *Cloud Computing, IEEE International Conference on*, 0:305–312, 2010. doi: 10.1109/CLOUD.2010.51.

C. Bunch, N. Chohan, and C. Krintz. AppScale: Open-Source Platform-As-A-Service. Technical report, University of California, Santa Barbara, Jan. 2011.

E. J. Chikofsky and J. H. C. II. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1):13–17, 1990.

N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. AppScale: Scalable and Open AppEngine Application Development and Deployment. In D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, E. Dekel, O. Akan, P. Bellavista, J. Cao, F. Dressler, D. Ferrari, M. Gerla, H. Kobayashi, S. Palazzo, S. Sahni, X. S. Shen, M. Stan, J. Xiaohua, A. Zomaya, and G. Coulson, editors, *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 57–70. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12636-9. doi: 10.1007/978-3-642-12636-9.

F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J.-M. Jézéquel. Model-Driven Engineering for Software Migration in a Large Industrial Context. In G. Engels, B. Opdyke, D. Schmidt, and F. Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 482–497. Springer Berlin / Heidelberg, 2007. doi: 10.1007/978-3-540-75209-7_33.

I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. *ArXiv e-prints*, 901, Dec. 2009.

S. Frey and W. Hasselbring. Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach. In *Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2010)*, pages 155–158, Lisbon, Portugal, Nov. 2010. ISBN 978-1-61208-001-7.

S. Frey and W. Hasselbring. An Extensible Architecture for Detecting Violations of a Cloud Environment's Constraints During Legacy Software System Migration. In T. Mens, Y. Kanellopoulos, and A. Winter, editors, *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, pages 269–278. IEEE Computer Society, Mar. 2011. ISBN 978-0-7695-4343-7. doi: 10.1109/CSMR.2011.33.

A. Fuhr, T. Horn, and A. Winter. Model-Driven Software Migration. In G. Engels, M. Luckey, and W. Schäfer, editors, *Software Engineering 2010: Fachtagung des GI-Fachbereichs Softwaretechnik 22.-26.02. 2010 in Paderborn*, volume P-159, pages 69–80, Bonn, 2010. Gesellschaft für Informatik.

W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, and S. Krieghoff. The Dublo Architecture Pattern for Smooth Migration of Business Information Systems: An Experience Report. In *In Proceedings of the 26rd International Conference on Software Engeneering (ICSE-04), Los Alamitos, California, May23-28 2004. IEEE Computer Society*, pages 117–126, 2004.

B. Hayes. Cloud Computing. *Commun. ACM*, 51:9–11, July 2008. ISSN 0001-0782. doi: 10.1145/1364782.1364786.

A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville. The Cloud Adoption Toolkit: Supporting Cloud Adoption Decisions in the Enterprise. *Software: Practice and Experience*, 2011a. ISSN 1097-024X. doi: 10.1002/spe.1072.

A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. B. Teregowda. Decision Support Tools for Cloud Migration in the Enterprise. *CoRR*, abs/1105.0149, 2011b.

M. M. Lehman. On Understanding Laws, Evolution, and Conservation in the Large Program Life Cycle. *Journal of Systems and Software*, 1:213 – 221, 1979-1980. ISSN 0164-1212. doi: 10.1016/0164-1212(79)90022-0.

P. Mell and T. Grance. Effectively and Securely Using the Cloud Computing Paradigm, Oct. 2009a.

P. Mell and T. Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 53(6):50, 2009b.

D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *CCGRID*, pages 124–131, 2009. doi: 10.1109/CCGRID.2009.93.

D. L. Parnas. Software Aging. In *Proceedings of the 16th international conference on Software engineering*, ICSE '94, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. ISBN 0-8186-5855-X.

B. P. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51, 2009. doi: 10.1109/NCM.2009.218.

R. C. Seacord, D. Plakosh, and G. A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 0321118847.

V. Tran, J. Keung, A. Liu, and A. Fekete. Application Migration to Cloud: A Taxonomy of Critical Factors. In *Proceeding of the 2nd international workshop on Software engineering for cloud computing*, SECLOUD '11, pages 22–28, New York, NY, USA, 2011a. ACM. ISBN 978-1-4503-0582-2. doi: 10.1145/1985500.1985505.

V. Tran, K. Lee, A. Fekete, A. Liu, and J. Keung. Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP). In *The Fifth International Symposium on Empirical Software Engineering and Measurement*, Banff, Alberta, Canada, September 2011b.

W. Ulrich. A Status on OMG Architecture-Driven Modernization Task Force. In *Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference.* IEEE Computer Society Digital Library, Sept. 2004.

L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009. ISSN 0146-4833. doi: 10.1145/1496091.1496100.

B. Wu, D. Lawless, J. Bisbal, and J. Grimson. Legacy System Migration : A Legacy Data Migration Engine. In *Proceedings of the 17th International Database Conference (DATASEM'97)*, pages 129–138, 1997a.

B. Wu, D. Lawless, J. Bisbal, and R. Richardson. The Butterfly Methodology : A Gateway-free Approach for Migrating Legacy Information Systems. In *Proceedings of the 3rd IEEE Conference on Engineering of Complex Computer Systems (ICECCS'97*, pages 200–205. IEEE Computer Society Press, 1997b.

L. Youseff, M. Butrico, and D. Da Silva. Toward a Unified Ontology of Cloud Computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov. 2008. doi: 10.1109/GCE.2008.4738443.

H. Zhou, H. Yang, and A. Hugill. An Ontology-Based Approach to Reengineering Enterprise Software for Cloud Computing. *Computer Software and Applications Conference, Annual International*, 0:383–388, 2010. ISSN 0730-3157. doi: 10.1109/COMPSAC.2010.46.

Acknowledgments

I'd like to thank:

- Sören Frey and the Software Engineering Group of Prof. Dr. W. Hasselbring,
- my wife, my children, my parents and my whole family (thx Berlin!),
- my boss during my former apprenticeship,
- Br^{35} Ba^{56}, Simon Posford & the Kalkbrenner Brothers,
- and both last and least the incredible Egusi Crew aka *Die Dünnbrettbohrer*.

Declaration

I hereby declare that I have completed the present thesis independently, making use only of the specified literature and aids. Sentences or parts of sentences quoted literally are marked as quotations; identification of other references with regard to the statement and scope of the work is quoted. The thesis in this form or in any other form has not been submitted to an examination body and has not been published.

Kiel, October 15, 2011

Sören Fenner