Institute of Architecture of Application Systems

CMotion: A Framework for Migration of Applications into and between Clouds

Tobias Binz, Frank Leymann, David Schumm

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

BibTeX:

```
@inproceedings {INPROC-2011-75,
    author = {Tobias Binz and Frank Leymann and David Schumm},
    title = {{CMotion: A Framework for Migration of Applications into
            and between Clouds}},
    booktitle = {2011 IEEE International Conference on Service-Oriented
            Computing and Applications (SOCA)},
    publisher = {IEEE Computer Society},
    month = {December},
    year = {2011},
    doi={10.1109/SOCA.2011.6166250},
 }
```


Universität Stuttgart
Germany

CMotion: A Framework for Migration of Applications into and between Clouds

Tobias Binz, Frank Leymann, David Schumm
Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

***Abstract*—The number of applications and services hosted in the cloud grows steadily, because of significant advantages in cost, flexibility, and scale compared to traditional IT. However, major difficulties in this field are (i) the migration of existing applications into the cloud and (ii) the increasing vendor lock-in which denotes the inability to leave a certain cloud provider without significant effort. Current approaches do not offer a holistic solution: Either they require the user to provide the application in a certain standardized way or they are only able to migrate one specific type of component. As a consequence, the migration of composite applications with different types of components is not supported. To overcome this limitation we propose the *Cloud Motion Framework* (CMotion) which leverages existing application models and provides support to migrate composite applications into and between clouds. Based on the application model, the framework evaluates alternative ways to host each component. CMotion assumes that the dependencies of components are modeled explicitly and the components are self-contained.**

***Keywords*-application migration; service management; cloud computing; composite applications**

I. Introduction

Many businesses want to move their existing applications into the cloud in order to benefit from the elasticity [1] [2] [3], payments based on actual usage (*pay-as-you-go*) [1] [2] [4], dramatically reduced cost [1] [4], flexibility in terms of on-demand and self-service [2] [4], ease of deployment and management [4], and energy efficiency [4]. However, it must be ensured that existing investments in applications could be taken to the cloud without high effort or complex code changes, which would reduce the financial benefit of the migration. Application migration is still a mostly unsolved problem in research [5] and vendor lock-in a high risk [6] for cloud computing. As a report of the *European Network of Information Security* states [6, p.25, R.1]: "There is currently little on offer in the way of tools, procedures or standard data formats or service interfaces that could guarantee data and service portability [...]". With the Cloud Motion Framework (CMotion) we try to address the problem of migrating applications into and between different clouds. CMotion uses adapters to make previously incompatible technologies able to work together. It processes the complete application stack and generates alternatives to deploy the application, while ensuring it still provides the same functionality.

The main contribution of this paper is a generic approach and procedure (see Figure 1) to generate alternative, cloud-based realizations of applications. We implemented our approach prototypically in Java and are able to generate and evaluate alternatives based on the provided application model.

The remaining paper is structured as follows: The three classes of migration we identified are described in Section II. Section III gives an overview of the framework and explains the running example. Section IV defines the entities CMotion is based on. The framework's architecture and operation is detailed in Section V. Section VI discusses related works in the area of portable applications and frameworks for their deployment. The paper concludes with a critical discussion of current limitations and future challenges (Section VII).

II. Cloud Application Migration

We performed a literature study on how migration of applications is realized nowadays and identified three classes:

Standardized Format Migration. In a Standardized Format Migration a component implemented in a standardized, self-contained format is migrated. Components are moved between instances of the same software or to other software solutions supporting this format. There are many widely used

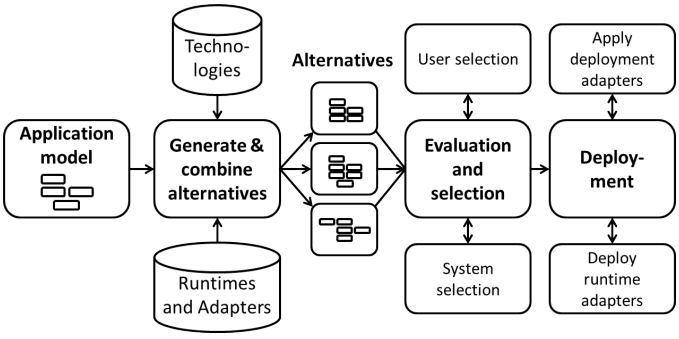

Figure 1. Procedure for Application Migration: Depicts the steps taking place in CMotion to do a migration of an application.

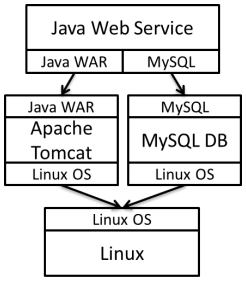

Figure 2. Application model of the composite sample application

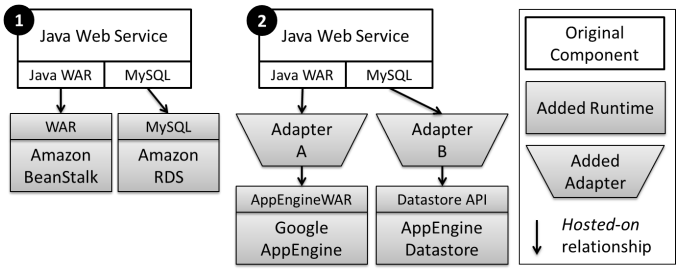

Figure 3. Different generated alternatives to host example application. Components needed to run the application in the cloud which did not exist in the original application model have been highlighted in gray color.

formats to facilitate migrations of this class, for example, migrating VMware or *Open Virtualization Format* (OVF) [7] images. Java [8] defines the standardized component framework *Enterprise Java* (EJB) [9] and portable application bundles like *Java Web Archives* (WAR) [10]. Both formats are executable on containers of different vendors (e.g., Apache Tomcat, Eclipse Jetty, and many commercial products).

Component Format Migration. In this class the format of the respective component is transformed into another format. Examples are transforming a virtual machine image or enable the execution of scripting languages on PaaS offerings. For example, there are projects to run Ruby on Google AppEngine[1] or PHP on Microsoft Azure[2].

Holistic Migration, the third class, is what CMotion aims to enable. It is the migration of a complete application, built out of multiple components, by migrating each component separately. In contrast to the component format migration it addresses composite applications consisting out of components on software, platform, and infrastructure level [2]. The related work in Section VI shows that current research focuses on the definition of standardized formats to enable portability.

Based on self-experience, we identified three requirements for a framework implementing holistic migration: (R1) The framework must be independent of a specific application model format. (R2) Enable migration of components without access to the component source (i.e., black boxes). (R3) Extensibility and openness to reuse previously implemented transformations in the framework.

III. Framework Overview and Procedure

CMotion processes the application model of the application which should be migrated. While not bound to a specific kind of application model (Requirement R1), we expect it to contain all components of the composite application, their dependencies, and implementations. Figure 2 shows the application model we will use throughout this paper to illustrate our approach. The top element is a Web service implemented in Java and packaged as *Java Web Archive* (WAR). It requires a runtime for WAR files and a MySQL database, consequently the Web service is deployed on an *Apache Tomcat* server and connects to a *MySQL DBMS*. Both, Tomcat and MySQL, are installed on *Linux*. The CMotion procedure depicted in Figure 1 goes on with processing the application model and generating new hosting alternatives by exploiting the available adapters and runtimes, as described in Section V in detail. To explain this step of the procedure, we will illustrate it with two exemplary alternatives: (1) The first alternative in Figure 3 depicts that the *Java Web Service* is hosted on *Amazon Beanstalk*, which is an Amazon-managed cluster of Tomcat servers, and uses *Amazon Relation Database Service* (RDS) as database, which allows us using cloud-enabled and managed MySQL without changing the application logic. (2) The second alternative in Figure 3 uses the respective services of Google. Slight changes of the WAR file are needed to host the *Java Web service* on *Google AppEngine*. This is realized by an adapter (*Adapter A*) which generates and adds the required XML file to the WAR before the deployment on *Google AppEngine*. The *AppEngine Datastore* is no relational database, which requires complex changes to the Java Web service. A human developer will do the required code changes, which is represented by a so called manual adapter.

The generated alternatives provide the same functionality, but depend on different building blocks. In the next step the alternatives are evaluated and the application model to deploy is chosen based on the cost. We chose a simple example for better illustration of the applied concepts. However, the framework can scale up and handle more complex applications. We make not claim that the framework is able to handle all types of applications.

IV. Conceptual Model and Framework Entities

This section defines the entities CMotion is built on. Figure 4 depicts the entities and their relation to each other.

Technologies are a classification of standards and APIs. To depict different variations or dialects, the technologies are structured as trees, for instance, the technology *SQL* has a child *PostgreSQL*, *MySQL*, and so on. Figure 5 depicts the

[1]http://code.google.com/p/appengine-jruby/

[2]http://www.php-compiler.net

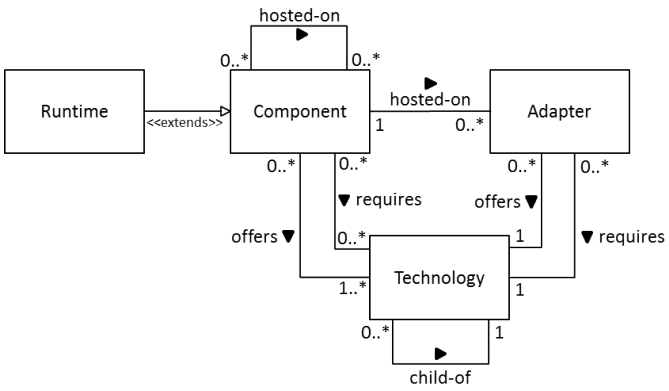

Figure 4. Conceptual Model of the framework entities and their relations.

current CMotion technology tree which can be extended by the user as needed.

Components are the nodes in the application model representing the building blocks of the application. In the running example (Figure 2) the components would be the *Java Web Service*, *Apache Tomcat*, *MySQL DB*, and *Linux*. The conceptual model in Figure 4 states that components may host other components which require one of the offered technologies. Components themselves may also require certain technologies to run.

Runtimes offer hosting for one or more technology and may require technologies themselves. *Apache Tomcat* in the sample application in Figure 2 is a runtime for the technology *WAR* and in turn requires an operating system. Runtimes may be offered as a service, like Amazon Beanstalk. In contrast to components, runtimes are pre-defined in the framework before the processing starts.

Adapters are components which are able to mediate between two different technologies. They provide the functionality to transform one format into another. CMotion uses adapters to generate hosting alternatives for applications, e.g., if for a given technology no suitable runtime was found, one or more adapters can be used to adapt the component's technology to a different technology. In Figure 3, *Adapter A* enables that the *Java Web Service* can run on *Google App Engine*.

We classify adapters into two dimensions: *when* and *how*: The *when* dimension denotes at which time in the application lifecycle [11] the adapter is applied: Either the transformation is done before the deployment of the application, for example, transforming a file into another format, or at runtime, for example, by adding an additional component doing the needed transformation on the fly. The other dimension represents two ways *how* adapters apply their changes: Automatic adapters are executed by the framework without human intervention, whereas manual adapters represent a task done by a human developer and are included into CMotion to consider their cost in the optimization.

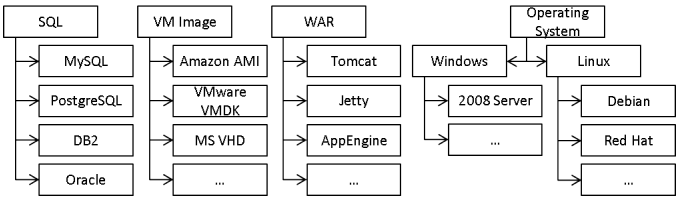

Figure 5. CMotion technology tree which can be extended by the user.

Users are able to develop additional runtimes and adapters or load them from a repository. Both are reusable (see Requirement R3), because they are only tied to technologies and not to the specific application which is migrated.

V. The CMotion Framework

This section briefly lays out how the CMotion framework operates, following the procedure depicted in Figure 1.

Prerequisites for applications processed by CMotion. The framework operates on the supplied application model which must depict all outside dependencies of its components. Additionally, it references the application files and if available the respective source code (see Requirement R2), which are processed by the adapters in a later step.

Step 1: Generation of Alternatives. Our approach starts from the top of the application, because these components deliver most of the distinguishing functionality, which is the *Java Web service* in our running example. From this component down towards the infrastructure we find more and more standardized and exchangeable components which can be realized in alternative ways. For each required technology of a component, one hosting component or runtime is needed. CMotion determines all runtimes and adapters offering the required technology and creates one new alternative for each possibility. This process builds a tree with a branch for each alternative hosting possibility in a depth-first way.

Step 2: Evaluation and Selection of Alternatives. For optimization and evaluation purposes each adapter, runtime, and component is associated with a specific *cost*. The cost model may include optimization criteria like response time, labor cost, licensing cost, or energy efficiency. Based on this cost metric the alternatives can be evaluated and selected.

Step 3: Deployment. The deployment is delegated to an existing, specialized deployment framework. Therefore, the deployment highly depends on the used application model and deployment framework. CMotion consumes the application model as *cafe application description* [12] and returns the chosen alternative in the same format. Our implementation can be adapted to other application models (Requirement R1 in Section II) by mapping the application model of choice to the conceptual model defined in Section IV.

VI. Related Work

The related work presents approaches to build portable applications and frameworks for their deployment.

SAGA[3], the *Simple API for Grid Applications*, provides operations frequently used for scientific grid applications like replication and job handling. [13] describes how SAGA applications can run on a cloud by providing an adapter implementation of the SAGA API. This approach and similar ones are in the class of *standardized format migration* as defined in Section II. In contrast, CMotion does not require the usage of a specific interface or application format. Arbitrary technologies, architectures, and models can be used. Whether a migration is possible or not fully depends on the availability of the respective adapters.

The *composite application framework* (cafe) [12] provides tooling to model so-called *application descriptions*, as well as means to customize, offer, and deploy the described applications. Based on cafe, [14] uses enterprise service bus (ESB) concepts for provisioning. Applications are developed against *virtual host components* which are replaced by real components during deployment. This approach can exchange implementations with the same interface (represented by technologies in this paper), similar to what ESBs do with services. CMotion goes on step further by adapting the interface offered by the respective components to match the required interface. With this, [14] could be extended to use components not matching the required interface.

VII. Future Challenges and Current Limitations

CMotion shows how applications can be migrated by using adapters and runtimes. In the following we discuss current limitations of CMotion and which challenges need to be addressed in future research.

Environment dependencies. Many components have dependencies onto the environment, for example, differences in folder separators between Windows and Linux or shared libraries in application servers. Therefore, each dependency must be declared explicitly in the application model to be processed during the migration.

Policy support. Policies regarding provider selection, security and compliance [5] [6], or ecological implications [15] must be considered to increase cloud adoption.

Application data migration. Data lock-in and data transfer are two major obstacles of cloud computing [16] and on the European union's list of research topics [4, p.3, Rec.1]. When migrating an application, the data processed by the application must be transferred together with the application.

Application's cloud characteristics. CMotion aims to support the migration of applications into the cloud and to reduce vendor lock-in for applications already hosted in the cloud. The programming model and non-functional requirements of the application stay the same. Future work may consider optimizations towards an elastic or multi-tenant aware design, exploiting essential cloud characteristics [2].

[3]http://saga.cct.lsu.edu/

Acknowledgment

The research leading to these results has partially received funding from the 4CaaSt project (http://www.4caast.eu) from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258862.

References

[1] F. Leymann, "Cloud Computing: The Next Revolution in IT," in *Proc. 52th Photogrammetric Week*, September 2009.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," *Information Technology Laboratory*, July 2009.

[3] OpenCloudManifesto.org, "Open cloud manifesto - dedicated to the belief that the cloud should be open," 2009.

[4] L. Schubert, K. Jeffery, and B. Neidecker-Lutz, "The future of cloud computing," European Commission - Information Society and Media, Tech. Rep., 2010.

[5] A. Khajeh-Hosseini, I. Sommerville, and I. Sriram, "Research challenges for enterprise cloud computing," *Information Security*, no. 1960, 2010.

[6] D. Catteddu and G. Hogben, "Cloud computing - benefits, risks and recommendations for information security," European Network and Information Security Agency, Tech. Rep., 2009.

[7] System Virtualization, Partitioning and Clustering Working Group, "Open Virtualization Format Specification (DSP0243)," *Distributed Management Task Force*, February 2009.

[8] J. Gosling, B. Joy, G. Steele, and G. Bracha, *Java Language Specification, Third Edition*. Addison Wesley, 2005.

[9] Sun Microsystems, "JSR 220: Enterprise JavaBeans 3.0," 2007. [Online]. Available: http://jcp.org/en/jsr/detail?id=220

[10] ——, "JSR 154: Java Servlet Specification, Version 2.5," 2007. [Online]. Available: http://jcp.org/en/jsr/detail?id=154

[11] G. Breiter and M. Behrendt, "Life cycle and characteristics of services in the world of cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, 2009.

[12] R. Mietzner and F. Leymann, "A self-service portal for service-based applications," in *SOCA*, 2010.

[13] A. Merzky, K. Stamou, and S. Jha, "Application level interoperability between clouds and grids," *Workshops at the Grid and Pervasive Computing Conference*, pp. 143–150, 2009.

[14] R. Mietzner, C. Fehling, D. Karastoyanova, and F. Leymann, "Combining horizontal and vertical composition of services," in *SOCA*, 2010, pp. 1–8.

[15] A. Nowak, F. Leymann, and R. Mietzner, "Towards Green Business Process Reengineering," in *Proceedings of the Workshop on Services, Energy, & Ecosystem: SEE*, 2010.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," Feb 2009.