# Service Migration Patterns - Decision Support and Best Practices for the Migration of Existing Service-based Applications to Cloud Environments

Christoph Fehling[1], Frank Leymann[1], Stefan T. Ruehl[2], Marc Rudek[3], Stephan Verclas[3]

[1]Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{fehling, leymann}@iaas.uni-stuttgart.de

[2]Clausthal University of Technology,
Clausthal-Zellerfeld, Germany
stefan.t.ruehl@tu-clausthal.de

[3]T-Systems International GmbH,
Frankfurt, Germany
{marc.rudek, stephan.verclas}@t-systems.com

BibTeX:

```
@inproceedings{FehlingLRR2013,
  author    = {Christoph Fehling and Frank Leymann and Stefan T. Ruehl and Marc
               Rudek and Stephan Verclas},
  title     = {Service Migration Patterns - Decision Support and Best Practices
               for the Migration of Existing Service-based Applications to
               Cloud Environments},
  booktitle = {Proceedings of the 6th IEEE International Conference on Service
               Oriented Computing and Application (SOCA)},
  year      = {2013},
  pages     = {9--16},
  doi       = {10.1109/SOCA.2013.41},
  publisher = {IEEE Computer Society}
}
```

# Service Migration Patterns

## Decision Support and Best Practices for the Migration of Existing Service-based Applications to Cloud Environments

Christoph Fehling, Frank Leymann
Institute of Architecture of Application Systems – University of Stuttgart
Stuttgart, Germany
{fehling, leymann}
@iaas.uni-stuttgart.de

Stefan T. Ruehl
Clausthal University of Technology
Clausthal-Zellerfeld, Germany
stefan.t.ruehl@tu-clausthal.de

Marc Rudek, Stephan Verclas
T-Systems International GmbH
Frankfurt, Germany
{marc.rudek, stephan.verclas}
@t-systems.com

*Abstract*— **In many ways cloud computing is an extension of the service-oriented computing (SOC) approach to create resilient and elastic hosting environments and applications. Service-oriented Architectures (SOA), thus, share many architectural properties with cloud environments and cloud applications, such as the distribution of application functionality among multiple application components (services) and their loosely coupled integration to form a distributed application. Existing service-based applications are, therefore, ideal candidates to be moved to cloud environments in order to benefit from the cloud properties, such as elasticity or pay-per-use pricing models. In order for such an application migration and the overall restructuring of an IT application landscape to be successful, decisions have to be made regarding (i) the portion of the application stack to be migrated and (ii) the process to follow during the migration in order to guarantee an acceptable service level to application users. In this paper, we present best practices how we addressed these challenges in form of service migration patterns as well as a methodology how these patterns should be applied during the migration of a service-based application or multiples thereof. Also, we present an implementation of the approach, which has been used to migrate a web-application stack from Amazon Web Services to the T-Systems cloud offering Dynamic Services for Infrastructure (DSI).**

***Keywords—SOA; cloud; migration; compliance***

## I. INTRODUCTION

Many companies evaluate the migration of existing applications to cloud environments for reasons such as the ability to scale resources flexibly [24] [18] and pay for resources on a pay-as-you-go basis [28] [20]. To benefit from a cloud, an application, however, has to respect the properties of this powerful environment in its architecture and its runtime operation. Newly developed cloud applications and existing applications to be migrated should, therefore, display certain architectural properties. In the following, we will cover the properties of cloud environments, derive architectural principles to be followed by cloud applications, and show that these *cloud architectural principles* are also predominant in service-based applications following the Service Oriented Computing (SOC) approach [5] [6] [9]. Therefore, we show the similarities between service-based applications and cloud-based applications by mapping their architectural principles to motivate that existing service-based applications are ideal candidates for a migration to a cloud environment. The remainder of this document is structured as follows. Section II describes the migration methodology we followed to move existing application to the cloud. Section III covers the mapping of architectural principles between SOC and cloud computing. Section IV covers best practices that we identified during the migration of existing applications in the form of patterns. A pattern in this scope is a document (or document section) following a certain format to capture a good solution to a reoccurring problem. Section V describes an evaluation scenario as customer applications landscapes from which the best practices have been deducted are mostly confidential. This scenario serves as a live demonstration showcase in the T-Systems Innovation Center[1] and describes the application of the migration methodology and migration patterns. Section VI then covers related work and Section VII concludes the paper by summarizing our findings and future work.

The work published in this article was partially funded by the Co.M.B. project of the Deutsche Forschungsgemeinschaft (DFG) under the promotional reference SP 448/27-1.

## II. MIGRATION METHODOLOGY

T-Systems‘ Cloud Readiness Services[2] provide a consulting service for the identification of cloud usage scenarios in enterprises, the evaluation of existing application landscapes, and the strategic restructuring of IT to use cloud computing. The overall process followed by this consulting service is displayed in Figure 1. In the initial *scoping* phase, the strategic motivation of customers to use cloud computing are evaluated. Motivation, constraints, and strategic goals are collected in workshops and discussions. In the following *CMO (current mode of operation) survey*, data about existing applications and their management is collected to identify applications suitable for a cloud migration. This analysis includes a detailed evaluation of legal and corporate regulative restrictions as well as an architectural analysis to estimate necessary changes in the application. In this paper, we provide *architectural properties* in Section III that proved relevant for a successful migration of applications to a cloud environment. Especially, we found that service-based applications share key architectural principles with cloud applications making them ideal migration candidates. Finally, the cloud readiness services include a

[1] http://www.t-systems.com/solutions/dynamic-services-for-infrastructure-computing-power-at-the-push-of-a-button/998132
[2] http://www.t-systems.com/solutions/analyze-your-start-in-the-cloud-t-systems/760004

strategic *FMO (future mode of operation) design phase* to determine how a company using cloud computing operates IT in the future. In this paper, we provide best practices that we followed to transform the current mode of operation to the future mode of operation using cloud computing. We capture these best practices in the form of patterns, which are documents following the well-defined pattern format introduced in [12] [4] [3].

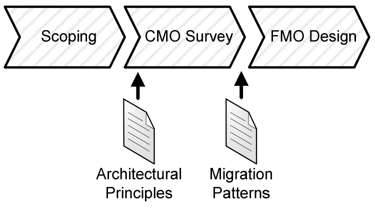

**Figure 1: Phases of the Migration Methodology and Supporting Content provided by this Paper**

## III. Architectural Principles of SOC and Clouds

When following the service-oriented computing (SOC) paradigm, resulting service-oriented architectures (SOA) display architectural properties that show a significant overlap to architectural properties required by cloud applications. We first cover the architectural properties of these two computing paradigms and then show that the evident overlap makes service-based applications suitable for the migration to clouds.

### *A. Architectural Principles of Service-based Applications*

According to Krafzig [6], a service-oriented Architecture is based on four abstractions: **1. Application Frontends**: graphical user interfaces or business processes [11] that control the service-based applications. These components may use services, thus, orchestrating provided functionality to support a business task, such as, handling loan approvals or stock trades. **2. Services**: services are functionality of an organizational unit of a company [9] [21] offered to other companies or departments. A service may be accessed through multiple interfaces. In addition to interfaces and an implementation, a service is also constituted by a service contract describing its function and behavior. **3. Service Repository**: service-consumers may discover services from the service-repository based on service-contracts as well as additional information, such as the physical location of the service, usage fees, and service levels. **4. Service Bus**: connectivity between the service-consumer and the enacted service is realized by an (enterprise) service bus. It serves as an intermediary to reduce the assumptions communication partners have to make about one another, such as location, availability, or used data format.

### *B. Properties of Cloud Environments*

The NIST cloud definition [20] is a widely accepted description of cloud environments. It gives four properties that a cloud commonly displays. These environment properties then lead to architectural properties of a cloud application. **1. On-demand self-service**: customers can access offered services on their own without the help of a human sales agent etc. This is often realized through a Web-based user interface or an application programming interface (API). **2. Broad network access**: the cloud services are connected to the customer networks with a significantly powerful network making the performance perceived by customers independent of the physical location of data centers. **3. Resource pooling**: IT resources used by a cloud provider are shared between customers to leverage economies of scale. This sharing also enables a flexible use of the service as resources that are no longer needed by one customer can be used to serve different customers. **4. Rapid elasticity**: through resource pooling and self-service interfaces, the flexible use of the shared cloud environment enables customers to provision and decommission resources very quickly. **5. Measured service**: the use of a cloud is measured by the provider to enable a transparent billing for customers often purely based on the actual use.

The NIST also defines three service models – *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)*, and *Software as a Service (SaaS)*. Cloud deployment models are described as *private clouds*, *public clouds*, *community clouds*, and *hybrid clouds*. As these characteristics of a cloud environment have fewer impact on the architectural principles of cloud-based applications, we do not cover them in detail. Refer to [20] and [4] for additional information.

### *C. Architectural Principles of Cloud Applications*

In [4], we identified properties in existing cloud applications that enable them to benefit from clouds. **1. Isolated state:** a cloud application should handle *session state* – state of the interaction with human users and *application state* – data handled by the application in as few application components as possible. Most cloud providers suggest handling state in communication offerings, i.e., messages or provider-supplied storage making application components stateless[3] [16]. **2. Distribution**: clouds are large distributed systems. Cloud applications should respect this by distributing functionality among multiple components. This enables the application to scale components independently and to rely on multiple distributed resources for resiliency. **3. Elasticity**: the cloud application has to support that cloud resources may be provisioned and decommissioned flexibly. The isolation of state is closely related to this property as the addition and removal of resources is significantly simplified if no state information has to be extracted or synchronized. **4. Automated Management**: manual changes to resource numbers are commonly not reactive enough to effectively benefit from usage-based billing supported by clouds. Also, cloud providers often do not assure availability for individual resources[4] suggesting automated failure handling. **5. Loose Coupling**: the dependencies among distributed application components constituting a cloud application should be reduced. This also eases the elastic scaling of the cloud application and simplifies coping with failures.

### *D. Mapping of SOC and Cloud Architectural Principles*

While *state isolation* is not explicitly required in a SOA, we found that it is often enabled as services expect *session state* to

[3] http://www.windowsazure.com/en-us/develop/net/fundamentals/intro-to-windows-azure/

[4] http://aws.amazon.com/ec2-sla/; http://www.windowsazure.com/en-us/support/legal/sla/

be provided with each request and encapsulate the *application state* of the service-based application. Modularity and the resulting *distribution* of application functionality among multiple components (services) is also inherent to every SOA, as services form encapsulated entities. The application frontend of a SOA is an additional special component that orchestrates other components. This form of application decomposition has been described as *process-based decomposition* in [4]. *Loose coupling* is an architectural property that is native in both service-based architectures and cloud application architectures. It especially shows the strong cohesion of both paradigms and is the reason why cloud computing is often perceived as having evolved from SOC [10] [7]. The two remaining cloud application properties, *automated management* and *elasticity* are not directly visible in every SOA. Automated management is often part of a service implementation as it operates independently and has to display an always-on behavior, thus, creating the need for automatic scaling and failure resiliency. The service registry is related to the elasticity of cloud applications as it may serve as a coordinator for multiple instances of a service that are provisioned and decommissioned when an application is scaled elastically. To summarize, the application components (services) and how they handle state are clearly defined in a service-based application. Loose coupling is inherent to both architectural paradigms and the basis for automated management and elasticity of a cloud application is also realized in a SOA. Due to these similarities, service-based applications have been targeted for the migration to clouds in multiple projects of T-Systems. As the discussed migration methodology can also be applied to other applications that do not have a service-oriented architecture, we use the term "component" in the following to denote a part of the application that shall be migrated. In scope of a SOA, these application components take the form of services.

## IV. Migration Patterns

After the current mode of operation survey has found applications that are suitable for a cloud migration, it is time to perform this migration. We captured the following patterns describing best practices to follow during this migration. Each section is formatted equally: a pattern has a name and an icon to be used in architectural diagrams (see evaluation in Section 6 for a demonstration). A pattern starts by summarizing its complete intent. Then, the question answered by the pattern is given followed by a description of the context in which the problem is observed. A solution is given describing how the problem is solved. It is supported by a sketch or an abstract process in BPMN [23]. More detail and problems possibly arising after the application of a pattern are given in the result section. Each pattern is concluded by a list of known uses.

### *A. Migration Target*

Applications are migrated at a layer of the application stack that is completely controlled by the migrating company. It should be the highest possible layer supported by the cloud provider and the highest common denominator when migrating multiple applications.

*How can the optimal portion of the application stack to be migrated be determined?*

**Context**: considering dependencies of an application on its hosting environment is critical for a successful migration. Dependencies mostly arise from the hosting infrastructure and other services that an application interacts with. A hypervisor [26] [25], for example, hosts an application as a virtual server. It may then require certain drivers and software to be installed in the virtual server that are incompatible with the target environment. Regarding other services that an application interacts with, one differentiates between operating support services (OSS) and business support services (BSS). OSS are necessary for the correct functioning of an application, for example, by providing operating system patches or anti-virus software. BSS are used to integrate an application in billing processes, reporting processes etc., for example, if use of the application shall be charged to departments or customers. **Solution**: the application is migrated at a level of the application stack that is controlled completely by the migrating company as seen in Figure 2. The remainder of the stack is recreated in the target environment.

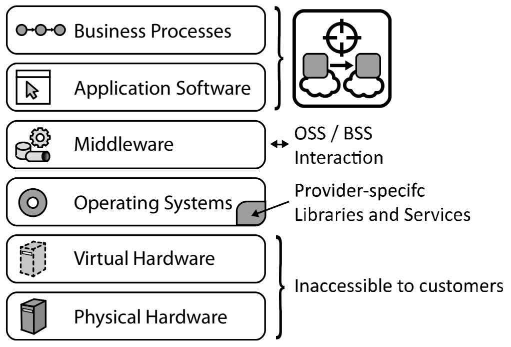

**Figure 2: Migration Target in an Exemplary Stack**

**Result**: the application stack is analyzed to determine dependencies on OSS and BSS functionality as well as on provider-supplied infrastructure and middleware. For each dependency, it is determined whether it is needed in the target environment and if so whether it can be migrated or be made accessible. The application stack is, therefore, likely to change during the migration. Even though recent industry standards such as OVF [8] have introduced standardization to the format of server images, migrations based on server images will, thus, have to consider the installed software and the application functionality. If servers hosting applications are inaccessible, this becomes especially apparent, as applications rely on provider-supplied functionality and this functionality cannot be extracted from the origin environment at all. **Known Uses**: Savvis [15] suggests the bottom-up consideration of layers in the OSI model [14] to determine the impact of the migration. This is used to identify a suitable *migration target*. Menzel and Ranjan [19] describe a decision support system for the selection of cloud providers based on the dependencies on provider-supplied services. Tran et al. [30] compute the complexity of migrations using a metric for connectivity, code adjustments, installation, configuration, and database adjustments. This metric can be computed to compare the complexity of different *migration targets*. Frey and Hasselbring [27] introduce a model for cloud environment constraints in order to describe dependencies of applications on the provider estimating migration complexity.

## *B. Forklift Migration*

Access to an application component is stopped. The component is then extracted from the origin environment and afterwards deployed in the target environment. During the transition, the component is, thus, temporarily unavailable.

*How can applications or application components that may experience some downtime be migrated?*

**Context**: applications and their components are considered to be hosted on an *elastic platform* [4] providing a managed hosting environment or an *elastic infrastructure* [4] providing virtual servers managed by customers. These provider-supplied cloud offerings provide a runtime environment to which applications may be deployed. Applications may also access provider-supplied middleware services for communication and data storage. According to the cloud service models employed by these environments – *Infrastructure as a Service* and *Platform as a Service*, respectively – the functionality used to manage the application is provided through a self-service interface or an API. Such an *elastic platform* or *elastic infrastructure* may pose the origin and target environment in scope of the migration of applications and their components. Alternatively, environments that do not display cloud computing properties may be target or origin of the migrated application. In this case, the migration process may include manual tasks. A critical aspect of migrating application components is that the migration itself can mostly not occur instantly but will take some time. This migration time subsumes the time it takes for the required middleware and runtime environment used by the application to be provisioned in the target environment as well as the time it takes the application itself to be deployed. Ensuring availability of the application during the migration time can be problematic and complex to ensure, because it means that the application has to be kept available in the origin environment while it is being extracted and then provisioned in the target environment. Therefore, if the application is accessed during the migration its *session state* – the state of the interaction with users and *application state* – the data handled by the application may change in the origin environment. This change will then have to be reflected in the target environment. However, it may be acceptable to have a certain time of unavailability.

**Solution**: state changes during the migration time are avoided by disabling access to the application during the migration, thus, making it unavailable for that period. The application is then extracted from the origin environment and provisioned in the target environment after which access is re-enabled.

**Result**: the application components are migrated from the origin to the target environment through interaction with the management interfaces offered by the *elastic infrastructure* or *elastic platform* as depicted in Figure 3. This migration includes the following steps. First, access to the application component is stopped by reconfiguring provider-supplied functionality, for example, load balancers, name-resolution, or access rules. Then, a snapshot of the running application component is created. This snapshot captures the current state of the application component. In case of a stateless application component, this step may be unnecessary as component instances are often provisioned based on snapshots and the stateless component does not have an internal state that could change during runtime. After the extraction, the component in the origin environment is decommissioned. In parallel to these two steps, the application stack identified as *migration target* is recreated in the target environment. Finally, the extracted component image – the components implementation and internal state is deployed on this recreated application stack and access is re-enabled.

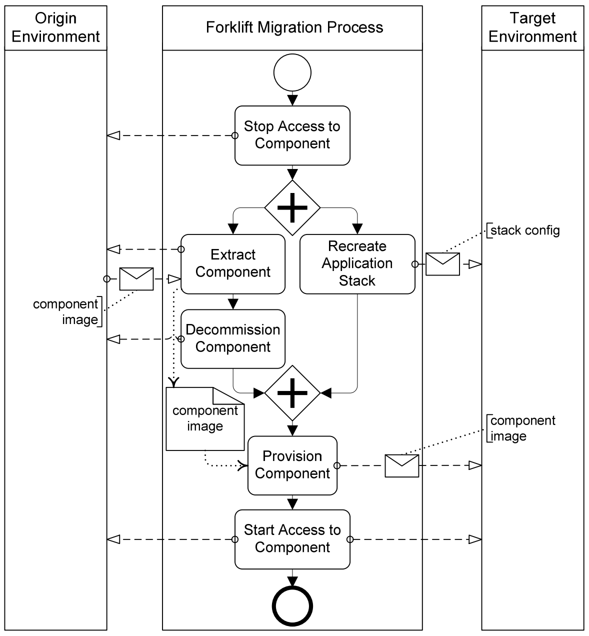

**Figure 3: Forklift Migration Process**

**Known Uses**: Savvis [15] describes the *forklift migration* of physical servers between different data center locations. Varia [17] covers *forklift migration* for self-contained web applications, applications whose components have a high level of interdependencies, and applications with few dependencies on other applications in the landscape, such as backup and archiving. Therefore, *forklift migration* is suggested for applications that do not need low latency interconnectivity with the remaining application landscape.

## *C. Stateless Component Swapping*

Stateless application components are extracted from one environment and deployed in another. They are active in both environments during the migration from the origin environment to the target environment, then, the old component instances are decommissioned.

*How can stateless application components that must not experience downtime be migrated?*

**Context**: in many business cases, the downtime of an application or one of its components is inacceptable. This may be the case for customer-facing websites or crucial company-internal applications. For example, a reduction in response time by the Amazon website of only 100 ms was found to result in 1% revenue loss [13]. A stateless application component shall, therefore, be migrated transparently to the accessing entity – human user or other applications. Again, "stateless" means that the application component does not handle an internal *session state* – state of the interaction with users or *application state* – data handled by the application. State is commonly provided with each requests or kept in provider-supplied storage. **Solution**: application component instances are first extracted from the origin then provisioned in the target environment. They are active concurrently. Access to the component instances is then switched immediately using a load balancer.

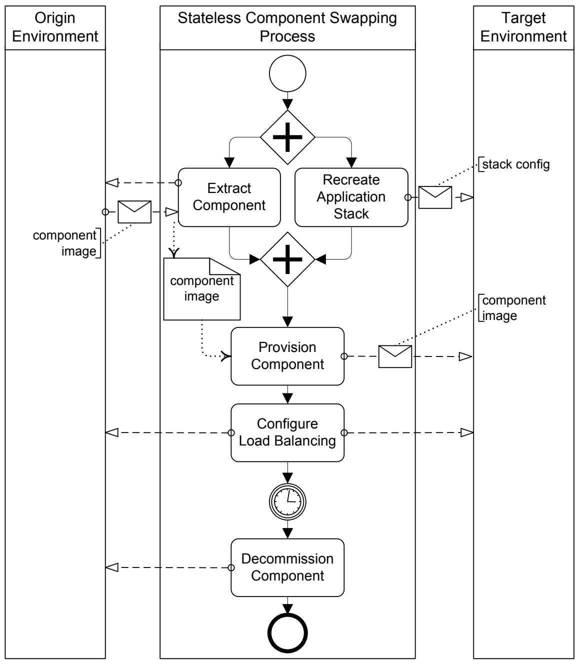

**Figure 4: Stateless Component Swapping Process**

**Result**: after extraction and provisioning the stateless component instances operate in both environments simultaneously as seen in Figure 4. The application stack is recreated according to the *migration target* pattern. Commonly, this is done by providing the target environment with the required configuration, for example, to provision a server image containing the required middleware. Application component instances in both environments rely on the same external storage offerings or other stateful application components to handle the *application state*. *Session state* is commonly sent with requests, thus, enabling component instances in both environments to handle requests in a unified fashion. The switch between component instances in the origin environment and the target environment is performed by reconfiguring the load balancer handling accesses to the instances. This could be a DNS-based load balancer, provider-supplied functionality, or special hardware.
**Known Uses**: Amazon describes the migration of an existing batch-processing application for media data to the Amazon AWS cloud in [1]. In this scenario, the application components handling media conversion are stateless as they retrieve a media file from storage, process it, and persist it again. The location of files to process is sent to these components using messaging. The *stateless component migration* is realized in this scope by provisioning media conversion components in both environments that rely on the same storage. Then, the component instances in the origin environment are decommissioned. Walberg describes a similar staged migration approach to move an existing Linux application to the Amazon cloud[5]. Again, multiple Web servers rely on the same external storage during and after the migration enabling them to be migrated transparently to application users.

### *D. Database Swapping*

Stateful databases are active in both origin and target environments during the migration. Handled data is kept in sync using hot-standby functionality provided by the database middleware.

*How can a database handling application state be migrated if it may not experience downtime?*

**Context**: as is the case for *stateless component swapping*, some application components may not experience a downtime during the migration for various reasons. However, if these application components handle state, a uniform behavior must be displayed by application component instances during the migration. Database management systems (DBMS) commonly provide means to replicate data to standby systems for redundancy and failure resiliency purposes. This functionality shall be used to enable a transparent migration. **Solution**: data handled by the database in the origin environment is extracted and provisioned in the target environment. Synchronization between these instances is enabled using DBMS hot standby functionality. After the database in the origin environment is no longer accessed it is decommissioned.

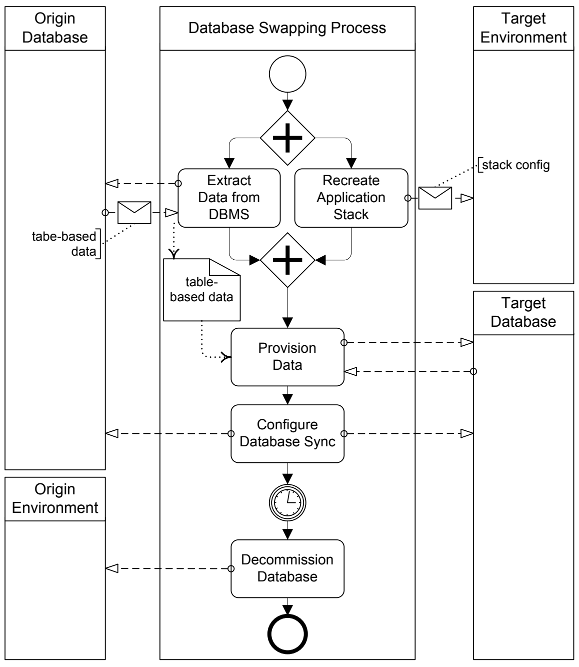

**Figure 5: Database Swapping Process**

**Result:** the extraction and recreation of the application stack depicted in Figure 5 is handled similar to the *stateless component swapping* pattern. However, note that the extraction

[5] http://www.ibm.com/developerworks/library/l-migrate2cloud-1/

accesses the database in the origin environment and not the environment itself. This data is then added to the newly provisioned database instance in the target environment. The standby functionality of the database management systems is configured which allows both database instances to be active simultaneously and display an equal behavior. The migration does not rely completely on the database sync as the initial data extraction and insertion is commonly quicker and less error-prone. Sync functionality is then only required to update the data that changed during the migration of the initial extraction. The switchover between the origin database and the target database is then performed by reconfiguring the application components accessing the databases. This is either done by changing the configuration of the accessing components directly or by provisioning new reconfigured component instances – possibly also in a different environment. This time required to reconfigure other application components is depicted as a timer event in Figure 5 for which the database migration process is inactive. It may be replaced by a message event in case the *database swapping* process shall be triggered explicitly after the migration of other application components.
**Know Uses**: many major database management systems, such as MySQL[6], ProstgreSQL[7], IBM DB2[8], and Oracle 11g[9] support the replication of handled data to hot standby systems, thus, the *database swapping* pattern may be realized using these products.

## *E. Hypervisor Swapping*

Stateful components are active in both origin and target environments during the migration. Handled data is kept in sync using storage area network (SAN) synchronization functionality, so that hypervisors can ensure the immediate switch of a virtual server.

*How can application components using virtual servers be migrated if they may not experience downtime?*

**Context**: as is the case for the two previous patterns some application components may not experience a downtime during the migration for various reasons. If state is not handled in databases but is stored on local file systems, the synchronization functionality of database management systems cannot be used to synchronize state as was the case for *database swapping*. Virtualization, however, enabled the abstraction from physical hard drives to virtual ones using hypervisor software [26] [25]. These hypervisors may use a storage area network (SAN) for keeping virtual server images. Synchronization and failover functionality is part of most SAN solutions.
**Solution**: virtual hard drives used by virtual servers are handled in a SAN in the origin environment. This storage is synchronized with the target environment using SAN functionality to enable the hypervisors to migrate a virtual server immediately as seen in Figure 6. Sometimes, the hypervisor may subsume SAN for synchronization as well.

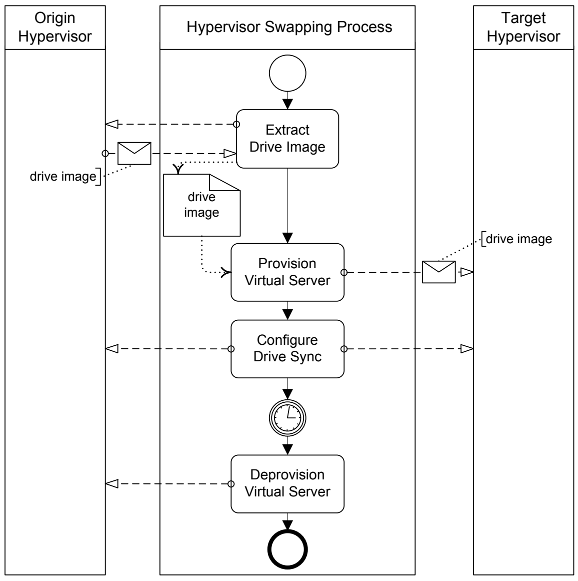

**Figure 6: Hypervisor Swapping Process**

**Result**: application components hosted in the origin environment use a virtual server that stores its hard drive in a SAN. This drive once extracted and moved to the target SAN is transparently kept in sync between environments. Depending on the used SAN, the initial drive image extraction may be optional. Due to the synchronization of the hard drive state, the hypervisor is enabled to perform the switch between the environments very quickly. After the synchronization has stabilized, the hypervisors may switch server instances by first moving the server in-memory state to the target environment and second by synchronizing this state just as the SAN synchronizes the hard drive. Then, the networking connection also controlled by the hypervisors are reconfigured for a near-real-time switch over. In difference to the *forklift migration* pattern or the *stateful component swapping* pattern, the application stack is not re-created in the target environment. The extracted drive image always contains the complete stack and provides the basis for the provisioned virtual server. Complications may, thus, occur due to differences in the hypervisors used in the origin and target environment as well as due to operating support services (OSS) and business support services (BSS), which is described in greater detail by the *migration target* pattern and the evaluation in the following section.
**Known Uses**: VMware supports *hypervisor swapping* as function of the VMware vMotion[10] product. OpenStack[11] supports a similar migration of SAN-based virtual servers.

## V. Evaluation

For confidentiality reasons of T-Systems' customers and their migrated application landscapes, we provide the following evaluation scenario as reference implementation for the migration methodology and the migration patterns. This

[6] http://www.mysql.com/
[7] http://www.postgresql.org/
[8] http://www.ibm.com/software/data/db2/
[9] http://www.oracle.com/products/database/

[10] http://vmware.com/products/datacenter-virtualization/vsphere/vmotion.html
[11] http://www.openstack.org/

scenario is used in the T-Systems Innovation Center[12] as live demonstration showcase. Most of the service-based applications we encountered at T-Systems' customers targeted for cloud migration used Web services as implementation technologies for a SOA. The evaluation scenario, therefore, also is a Web service-based application. To demonstrate the generality of the approach, we decided to use Amazon EC2[13] as the origin environment for the application and T-Systems Dynamic Services for Infrastructure (DSI) [14] as target environment to which the application was migrated. The approach is, however, also usable for applications hosted in non-cloud or other cloud environments than the two employed here. Figure 7 depicts the application stack and the pattern icons annotated to it indicating where patterns are applied to realize the migration. The user interface is based on PHP. It serves as an *application frontend* in scope of the architectural principles of service-based applications discussed in Section 2. It accesses Web services implemented using Jax-WS[15]. The Web services access table-centric data. The PHP user interface and the Web services are both hosted on Apache Tomcat[16]. The table data is hosted by a MySQL[17] database. The application is furthermore divided into two tiers by hosting the user interface on one virtual server while the Web services and database are hosted on a separate server.

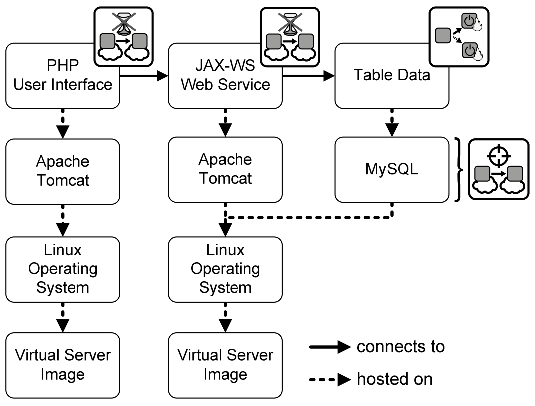

**Figure 7: Migration Patterns in the Evaluation Scenario**

The application of migration patterns to this application was done as follows. First, the *migration target* was identified. Test extractions of virtual machines hosted at Amazon EC2 were performed to evaluate the migration on the virtual server level. We used Linux-native tools – *dd* for direct access to the virtual drive and *ssh* to copy the content:

```
ssh -i <ec2sshcredentials> <ec2user>@<ec2hostname>
'sudo dd if=/dev/xvda1 bs=1M | gzip' | gunzip | dd
of=image.raw
```

This image was then converted to a VMware[18] disk image using qemu[19] as converter and tested locally on a desktop. The following dependencies on the Amazon environment using the XEN [20] hypervisor were found hindering the migration: (i) Virtual drives of guests cannot be booted independently – the extracted virtual drive image did not contain a boot sector as it relied on the hypervisor for this purpose. (ii) Kernel and library sharing – the guest Linux systems, depending on the concrete image provided by Amazon or third party providers, often shared the kernel and important system libraries with the host – a means to reduce the footprint of guest systems. After extraction, the virtual servers were, therefore, often left without these core operating system components. (iii) Provider-supplied BSS and OSS services were unavailable. Images that could be booted after kernel and boot sector were added to the extracted images displayed a very long boot time as many monitoring and billing services were tried to be accessed, which were unavailable. Due to these obstacles, the middleware layer of the application stack comprised of Apache Tomcat and MySQL was chosen as *migration target*. Most cloud providers, including the T-Systems DSI, provide ready to use images for this middleware. Regarding the application components, *stateless component swapping* was used for the user interface and the Web services. As these components do not hold *session state* or *application state*, they were extracted from the origin environment and deployed on the recreated middleware application stack in the T-Systems DSI. Access to these components was load balanced using DNS records. Entries in the DNS servers were, therefore, changed to the new component instances and after a certain time for DNS updates to traverse, the old component instances were decommissioned. *Database swapping* was used to migrate the database content. For this purpose, the new deployment of the MySQL database in the target environment was configured to serve as a hot standby for the database in the origin environment. This ensured that the user interface and Web service instances provided a uniform behavior while being active in both environments. After the DNS-based switchover, the MySQL hot standby became the main database and the MySQL instance was decommission in the origin environment. We performed this migration process once manually according to the abstract processes described by the migration patterns. However, the application stack is quite common and only standard interfaces of the Linux operating systems and the application middleware had been used to extract and migrate the application components. Therefore, the migration process itself has been automated as well to be offered as a service. It is based on the Activiti BPMN[21] engine that coordinates the order in which the separate migration processes handling individual application components have to be performed. These processes described by the migration patterns have, thus, been implemented as sub-processes to the overall migration process handling applications using the stack depicted in Figure 7. The human task manager of the Activiti Engine has been used to inquire necessary information from the user, such as EC2 credentials and folder locations of PHP files on the origin server etc. For space limitations, screenshots of the Activiti process have not been included. A demonstration video of the migration can be accessed online[22].

[12] http://www.t-systems.com/innovations/innovations-you-can-touch-t-systems/1054302
[13] http://aws.amazon.com/ec2
[14] http://www.t-systems.com/solutions/dynamic-services-for-infrastructure-computing-power-at-the-push-of-a-button/998132
[15] https://jax-ws.java.net/
[16] http://tomcat.apache.org/
[17] http://www.mysql.de/
[18] http://www.vmware.com/products/workstation/
[19] http://www.qemu.org

[20] http://www.xenproject.org/
[21] http://www.activiti.org/
[22] http://www.youtube.com/watch?v=b8NBkSAN0Iw

## VI. Related Work

According to the pattern format used in this paper and other publications [12] [4], we mention most related work in the known uses section of each migration pattern. Furthermore, the migration patterns may be integrated with other existing patterns. They may be linked with cloud computing patterns [4] [2] describing cloud application architecture in greater detail. During the migration of stateful components additional data patterns [29] may be considered to adjust data through obfuscation or anonymization in order to adhere to laws and establish privacy in the target environment. These data patterns also describe how the data-handling middleware, MySQL in our evaluation scenario, can be changed to different data-handling middleware. Other patterns may be relevant in case only parts of an application landscape are migrated, which will commonly be the case as a company will hardly ever replace its complete IT infrastructure with a cloud provider. Hohpe [12] describes patterns that can be used for the necessary integration of different enterprise applications. Regarding the pre-migration considerations, different online tools are available that mainly target the evaluation and prediction of costs. Amazon[23], Azure[24], and Rightscale[25] each provide cost calculators. Similar functionality is also provided generically by goCipher's Cloud Cost Calculator[26].

## VII. Summary and Outlook

The migration methodology introduced in this paper describes the different phases used by T-Systems in customer projects to evaluate and execute the migration of existing application landscapes to cloud environments. It has been shown that service-based applications have certain architectural properties making them ideal candidates to such migrations. Best practices for the migration that were used after the evaluation and identification of applications have been presented in the form of reusable migration patterns. The methodology and patterns were evaluated for service-based applications. The migration process for such applications was automated to reuse it for the migration of similar applications in the future. We used application model diagrams in Figure 7 to describe an application stack. This diagram and especially the used links between components made use of implicit semantic that has not been well defined. New industry standards target the standardization and well-defined modeling of application stacks and the management tasks related to them during their runtime. Amazon CloudFormation[27], VMware vFabric Application Director[28], and the OASIS standard TOSCA [22] describe models for such application stacks. In the future, the migration patterns could be integrated in such modeling tools. We find the TOSCA standard especially suitable for this purpose as it incorporates BPMN processes describing management tasks handled for applications. It, therefore, seems to ideally support the abstract processes of the migration patterns as well as the automated process used in the evaluation scenario.

[23] http://calculator.s3.amazonaws.com/calc5.html
[24] http://www.windowsazure.com/en-us/pricing/calculator/
[25] http://www.rightscale.com/cloud-cost-calculator/
[26] http://cloudpricecalculator.com/
[27] http://aws.amazon.com/cloudformation/
[28] http://vmware.com/products/application-platform/vfabric-application-director/

## References

[1] Amazon Web Services, "Migration scenario: migrating batch processes to the AWS cloud," White Paper, 2010.

[2] B. Wilder, Cloud Architecture Patterns, O'Reilly, 2013.

[3] C. Fehling, F. Leymann, R. Retter, D. Schumm, W. Schupeck, "An architectural pattern language of Cloud-based applications," Pattern Languages of Programs (PLoP), 2011.

[4] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, Cloud Computing Patterns, Springer, 2013 (in production).

[5] D. Georgakopoulos, M. P. Papazoglou, Service-Oriented Computing, MIT Press, 2008.

[6] D. Krafzig, K. Banke, D. Slama, Enterprise SOA, Prentice Hall, 2005.

[7] D. S. Linthicum, Cloud Computing and SOA Convergence in Your Enterprise, Addison-Wesley, 2009.

[8] DMTF, Open Virtualization Format (OVF) 2.0, 2013.

[9] F. Curbera, F. Leymann, T. Storey, D. Ferguson, S. Weerawarana, Web Services Platform Architecture, Prentice Hall, 2005.

[10] F. Leymann, "Cloud Computing: The Next Revolution in IT," in Proceedings of the 52th Photogrammetric Week, 2009.

[11] F. Leymann, D. Roller, Production Workflow: Concepts and Techniques, Prentice Hall, 2000.

[12] G. Hohpe, B. Woolf, Enterprise Integration Patterns, Addison-Wesley, 2004.

[13] G. Linden, "Make data useful", Talk at Standford University, 2006.

[14] ISO, Open System Interconnection (OSI), Standard 7498, 1984.

[15] J. Piazza, "Computing migration strategies," Savvis White Paper, '09.

[16] J. Varia, "Architecting for the cloud: best practices," Amazon White Paper, 2010.

[17] J. Varia, "Migrating your existing applications to the AWS cloud," Amazon White Paper, 2010.

[18] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I.Stoica, M. Zaharia, "Above the clouds: a Berkeley view of cloud computing," technical report, 2009.

[19] M. Menzel, R. Ranjan, "CloudGenius: decision support for web server cloud migration, " International Conference on World Wide Web, 2012.

[20] National Institute of Standards and Technology, "The NIST definition of cloud computing," 2009.

[21] O. Zimmermann, An Architectural Decision Modeling Framework for Service-oriented Architecture Design, Thesis University of Stuttgart, 2009.

[22] OASIS, Topology and Orchestration Specification for Cloud Applications, 2013.

[23] OMG, Business Process Model and Notation (BPMN) 2.0, 2011.

[24] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering Computing as the 5th Utility," Future Generation Computer Systems, 2009.

[25] R. P. Goldberg, "Architecture of virtual machines," Proceedings of the Workshop on Virtual Computer Systems, 1973.

[26] R. P. Goldberg, "Virtual machines: semantics and examples," Proceedings IEEE International Computer Society Conference, 1971.

[27] S. Frey, W. Hasselbring, "The cloudmig approach: model-based migration of software systems to cloud-optimized applications," International Journal on Advances in Software, 2011.

[28] S. Jha, A. Merzky, G. Fox, "Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes," Concurrency and Computation: Practice and Experience, 2009.

[29] S. Strauch, V. Andrikopoulos, T. Bachmann, F. Leymann, "Migrating application data to the cloud using cloud data patterns," International Conference on Cloud Computing and Service Science (CLOSER), 2013.

[30] V. T. K. Tran, K. Lee, A. Fekete, A. Liu, J. Keung, " Size estimation of cloud migration projects with cloud migration point (CMP)," Empirical Software Engineering and Measurement (ESEM), 2011.