# Development of Adaptive Multi-cloud Applications
## *A Model-Driven Approach*

Javier Miranda[1], Joaquín Guillén[2], Juan Manuel Murillo[1] and Carlos Canal[3]
[1]*Department of Information Technology and Telematic Systems Engineering, University of Extremadura, Badajoz, Spain*
[2]*GloIn, Calle de las Ocas 2, Cáceres, Spain*
[3]*Department of Computer Science, University of Malaga, Málaga, Spain*

Keywords: Service, Component, Cloud, Adaptation, MDE, Interoperability, Vendor Lock-in.

Abstract: Cloud computing is a new paradigm that allows users to access computing resources in a dynamic, flexible and scalable manner. It has drawn the interest of multiple users, and in a short period of time it has experienced a notorious hype. However, its numerous strengths are mitigated by the lack of standardization which the technology suffers from. Different cloud vendors provide and manage similar resources in a different manner, thereby coupling the application to its targeted cloud. Companies that consume cloud services are locked-in to a single cloud vendor due to the high costs of migrating software in the cloud, preventing them from changing their cloud provider or having multiple providers. In this paper we explore a solution to the cloud vendor lock-in problem based on the use of model-driven engineering and software adaptation techniques. The proposed solution is both cloud vendor and user friendly as it allows the former to freely define their own cloud policies, whilst users continue to be free to choose a cloud provider, even after the application has been developed.

## 1 INTRODUCTION

Cloud computing is a new paradigm that allows users to access computing resources in a dynamic, flexible and scalable manner. The underlying technology and its business model have drawn the interest of multiple users, and in a short period of time it has experienced a notorious hype (Leavitt, 2009). However, its numerous strengths are mitigated by the lack of standardization which the technology suffers from (Armbrust et al., 2010). Cloud vendors provide and manage similar resources in a different manner, thereby coupling the application to its targeted cloud. This is known as the vendor lock-in problem (Petcu et al., 2012), and has immediate consequences on companies that consume cloud services. Considering a catalogue of cloud users composed on one end by companies that deploy complex services and architectures which they want to keep under a strict control, and on the other end by companies that deploy smaller public services where availability and performance is a critical factor due to the high number of users, different standardization interests may be identified for each. The former will rarely fully deploy their applications in a public cloud, and will be mainly interested in standardization for enabling the interoperability between their private and public cloud infrastructures. The latter would look into standardization seeking to freely scale and also migrate their services from one cloud provider to another and/or to distribute them among several clouds at a time.

In this scenario different initiatives, such as OVF (Open Virtualization Format), OCCI (Open Cloud Computing Interface), OGF (OpenGrid Forum), or OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications), are emerging in order to solve the problems derived from the aforementioned shortcomings, taking a first step towards defining standards that would homogenize the existent cloud services at different levels (Rochwerger et al., 2009). However, these proposals are currently at a very early stage; no generalized consensus has been reached and neither of them have been adopted massively (Celesti et al., 2010). Application interoperability and migratability between cloud providers was never a concern when the technology was conceived. In fact, cloud vendors have their own implementation and specification of

Miranda J., Guillén J., Murillo J. and Canal C..
Development of Adaptive Multi-cloud Applications - A Model-Driven Approach.
DOI: 10.5220/0004370603210330
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODA-2013), pages 321-330
ISBN: 978-989-8565-42-6

the services that they provide, locking users into their solutions and preventing the portability of cloud applications to other providers. This situation threatens the success of Cloud Computing as a universal service, where users can switch between providers as they need (Loutas et al., 2011). The current lack of support for these standardization attempts by the existent vendors suggests that an agreement is not going to be reached, at least in the short term. Even if it is so, more immediate alternatives could take place before an agreement on the use of standards is reached. Furthermore, the possible consolidation of more than one of these proposals would provoke interoperability problems between vendors adopting different standards.

Alternatives to standardization are mostly based on the use of middleware layers (see for instance, (Di Martino et al., 2010); (Tsai et al., 2010); (Maximilien et al., 2009) which isolate the application from vendor specific services. However, middleware solutions are often quite complex and heavyweight. Considering that they have to be deployed in conjunction with the application, they will clearly penalize the performance of the software components attached to them. Further yet, the source code of middleware-dependent components will be tightly coupled to the specification of the middleware, thereby moving the lock-in effect from vendors to middleware.

Instead, our approach is based on the integration of Model-Driven Engineering (MDE) and Software Adaptation (SA) techniques. Developers are requested to tag the components indicating which cloud they will be deployed in; MDE techniques are then applied to generate a XML based cloud deployment plan. The source code and the XML deployment plan are processed to generate cloud compliant artefacts in order to access the underlying cloud services. Then, on-the-fly migration of cloud components, allow service developers to easily change the underlying cloud depending on vendor offers, or their own market and evolving business perspectives at any given time. Migration can be achieved by means of SA techniques, generating the adapters required for allowing a component to move to a cloud it was not originally conceived for. The most outstanding benefit of using adapters in cloud environments is the ability to automatically generate loosely coupled applications with a reduced impact on their deployment, and at the same time favouring cloud interoperability.

This paper presents ongoing work in our proposal, which was originally presented in (Guillén et al., Oct. 2012). There, the concept of using a software development framework for building cloud applications was introduced. The framework allows developers to separate all possible dependencies between the software being developed and the cloud from the source code through the generation of software adapters.

In (Guillén et al., Oct. 2012), adaptation was briefly presented as the candidate technique to be used in the framework in order to keep the application source as cloud-agnostic as possible. In a subsequent work (Guillén et al., Sept. 2012), we analyzed in more depth every situation in which mismatch could be produced, and how SA techniques would of help in solving these situations.

We expect that this combined MDE/SA approach provides several advantages in comparison with existing alternatives, mainly because it consists on a lightweight solution which effectively deals with cloud interoperability and migratability issues.

The remainder of this paper is organized as follows. Section 2 presents the motivation of our work, describing an scenario of migratable multi-cloud application development. Section 3, presents our approach for developing and deploying cloud agnostic software. Then, Section 4 identifies the scenarios in which the use of adapters is required, and describes our adaptation approach, pointing out the main notation of interest and applicable techniques to achieve adaptation at any interaction level. Next, Section 5 contains a description of the related work. Finally, Section 6 presents our conclusions and future lines of work.

## 2 MIGRATABLE MULTI-CLOUD DEVELOPMENT

In the recent years, cloud computing has become overwhelmingly popular. It provides a virtually unlimited amount of computational infrastructure to its users at an accessible cost, as no technology has ever done before. Its popularity has also grown due to the possibilities that it provides for outsourcing maintenance tasks, allowing organizations to concentrate on their core competencies.

The great potential of this technology and its growing acceptance have resulted in the appearance of multiple cloud vendors which provide similar services. The variability found at different levels of these services has resulted in the vendor lock-in effect. However, the ultimate goal of cloud computing is being able to build and deployed Service-Based Applications (SBAs) by combining

conveniently the adequate resources available in the cloud, even though they are supplied by different third-party providers. For that purpose, there is a need of techniques and tools addressing a wide range of interaction mismatch issues, caused by the heterogeneous origin of the services one may be interested in composing. Only the flexibility of cloud computing provides the fabric through which SBAs can be constructed and deployed (Nguyen et al., 2011).

Indeed, interoperability and migratability between cloud providers was never a concern when the technology was conceived. As we have shown, cloud vendors have their own implementation and specification of the services they provide, locking their users into specific solutions and preventing the portability of the applications. Current cloud services are provided as a one-size-fits-all solution, usually with preconfigured and monolithic IaaS/PaaS/SaaS combinations. This situation threatens the success of cloud computing as a universal service where users can switch between providers as they need (Loutas et al., 2011. The choice of a given cloud provider may prevent the use of certain data formats demanded by users due to incompatibilities with the underlying infrastructure and platform.

These problem become even bigger when facing the development of multi-cloud SBAs, trying to combine the functionality of services hosted in different clouds. Then, the inconsistency of cloud resource descriptions and the use of proprietary technologies by cloud vendors are the main barriers that must be confronted for communicating services deployed on clouds by different providers.

For all these reasons, cloud application migratability and interoperability are currently hot issues that are being strongly questioned (Armbrust et al., 2010), due to the inability to migrate software which has been coupled to a specific cloud, to a different environment. Additionally, little or no effort has been made on behalf of cloud vendors to favour interoperability with services and applications hosted by different clouds.

In order to illustrate the current scenario, let us consider a generic application (see Figure 1), designed as the composition of several service-based components, partly hosted in-house and partly in the cloud, as the *Cloud Component* in the figure. Suppose also that *Cloud Component* behaves as a service consumer for a SaaS application deployed in a maybe different cloud environment. This scenario is a variation of the one we previously presented in (Guillén et al., Sept. 2012), which is retaken and summarized here for motivating our proposal.

The dotted lines in Figure 1 represent the different dependencies between *Cloud Component* and the rest of the system:

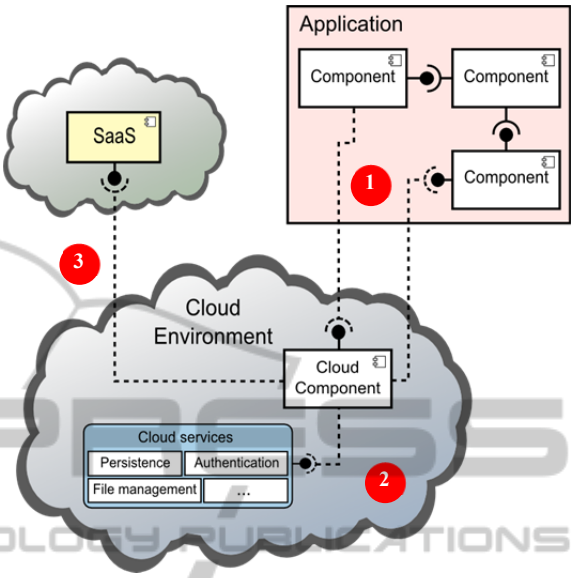


Figure 1: Current scenario for SBA deployment in the cloud.

1. *Cloud Component* communicates with the remaining components in the SBA application. In order to allow service provision and consumption to and from the cloud, this communication has to be compliant with the specifications of the cloud provider in which the component is hosted.
2. *Cloud Component* presents also dependency to services (e.g., persistence or authentication) supplied by the cloud it is hosted in. These services present specific interfaces and features, which are defined by the cloud provider.
3. *Cloud Component* consumes external services, that may be hosted in one or several different clouds. In order for this to be done, the communication must align with the constraint imposed by the external cloud and service providers.

Consider now that we decided to migrate *Cloud Component* to a different cloud. The reasons for that change could be multiple; on the one hand, in the immature market of cloud computing, providers are likely to change their hosting or changing policies, or they may simply disappear. On the other hand, a new cloud vendor may become more interesting for hosting our component, for instance ensuring a SLA which fits better the application's QoS requirements.

Under these circumstances, migrating *Cloud Component* will necessarily affect the aforementioned dependencies, that would need to be reworked in order to make the component compliant with the constraints of the new cloud environment.

However, little effort is currently being put into generating tools, techniques, procedures and standard data formats with enough potential to solve these issues, and probably we will decide to keep *Cloud Component* hosted in its original cloud, or either redevelop it from scratch, considering its new target cloud.

To sum up, the current scenario regarding SBA development for the cloud has the following implications:

- Communication between components and services is strictly conditioned by the technology supported by each cloud and service provider.
- The invocation mechanisms and technologies supported by each cloud provider must be taken into account for invoking external components and services provided by third-parties.
- The use of vendor-specific technologies and services provoke migration and portability problems that have to be taken into account during the design stage.

Hence, multi-cloud SBA interoperability and cloud migratability require a number of conditions to be fulfilled:

- The modelling and development of the different artefacts that compose a cloud application and their requirements should be done in a vendor-independent and cloud-agnostic manner. This way, the application constituent components are not constrained by the technical requirements of any cloud or service provider. Only later in the development process it will be evaluated whether the application requirements for each component are supported by a specific cloud provider, or if this is not the case, adapters can be generated to solve the existing mismatch.
- The assignment of an application component to a given cloud, or to in-house hosting is a decision that may be reverted at any time in the lifecycle of an SBA. Hence, its design and implementation must require no additional effort depending on which cloud platform it is finally hosted.
- A multi-cloud SBA must be able to integrate external services, developed by third parties and hosted in different clouds. Interface information on the requirements and behaviour of these services will be used to link the SBA with these services adequately.

According to the conditions above, the development process for cloud SBAs should be quite similar to the one carried out for in-house service-based applications. Software developers should be able to describe all of the components involved in their applications through the use of the same set of techniques and methodologies. Additionally, they may choose to tag the components with information about the specific cloud where they will be deployed. This information will be interpreted during the development process in order to determine which components need to be adapted as well as the type of adaptation that they require.

The following sections describe ongoing work on our proposal for building multi-cloud SBAs in which components could easily migrate from one cloud to another one, even at runtime.

# 3 DEVELOPING CLOUD-AGNOSTIC APPLICATIONS

Cloud SBA interoperability arouses a series of concerns that can be successfully solved combining MDE and SA techniques. The variability between the API and service specifications of each cloud provider can be analyzed and defined, resulting on a feature model describing cloud platform variability. Then, cloud applications, as well as the requirements of the different components that make up the developed application, can be modelled in a cloud-agnostic way, and MDE techniques are used to generate components tailored to the particular features of a given cloud. Finally, component-to-cloud adapters would be generated for solving interoperability problems, and for allowing component migration between different clouds, even at runtime.

This section presents our approach based on MDE and SA techniques for developing SBAs that can be migrated freely from one cloud to another, hence overcoming the vendor lock-in effect. The development process is divided into three phases: (i) application modelling, (ii) coding and deployment configuration, and (iii) cloud artefact and adapter generation. Each phase will be detailed in the following subsections.

## 3.1 Application Modelling

During this phase developers will model a cloud

application for which the source code and a cloud deployment plan will be generated through model-to-text transformations. For that, a cloud application metamodel has been defined. The metamodel is based on a UML profile, which ensures that users would not need to change their engineering process, and that standard UML tools can be used along with our proposal. Based on this profile, cloud applications will be modelled as groups of components, further referenced as cloud artefacts, taking into account the following considerations:

- Each cloud artefact must be considered as an atomic software entity that must be deployed in a single cloud platform.
- Components that belong to the same cloud artefact will communicate with one another locally; i.e. no mediation will be required to allow communication between these components.
- Components that belong to different cloud artefacts will communicate with one another remotely; i.e. mediation will be required to allow communication between these components.
- Components may consume external services provided by other applications.

Our cloud application metamodel is shown in Figure 2, and it defines cloud applications as built by the composition of cloud elements or artefacts hosted in one or more clouds. Cloud artefacts are tagged with the stereotype *CloudElement*, and several elements can be assigned to a specific cloud (stereotype *CloudAssignment*). Assignments are associated with the *QoSParameter* stereotype, which allows QoS properties to be defined for the assignment. Finally, the *CloudInterface* stereotype describes the interfaces used by cloud artefacts to interact with each other and with the services provided by the cloud they are hosted in, and indicates if these interfaces are provided by the artefact, or required to the cloud provider. In the latter, the artefact cannot be deployed in the cloud it is assigned to unless it offers the services required by the artefact or an adapter is generated to solve the existing mismatch. That will be explained in the following subsections.

A model-to-text transformation will be then applied on the models created during this stage in order to generate class skeletons for establishing the basic structure of the application. These skeletons are cloud agnostic; i.e. all cloud related information will be generated separately in a XML formatted cloud deployment plan. Round-trip engineering techniques will be used to synchronize the application model with the actual code added to the skeletons during the following development process, avoiding inconsistence between these two levels.

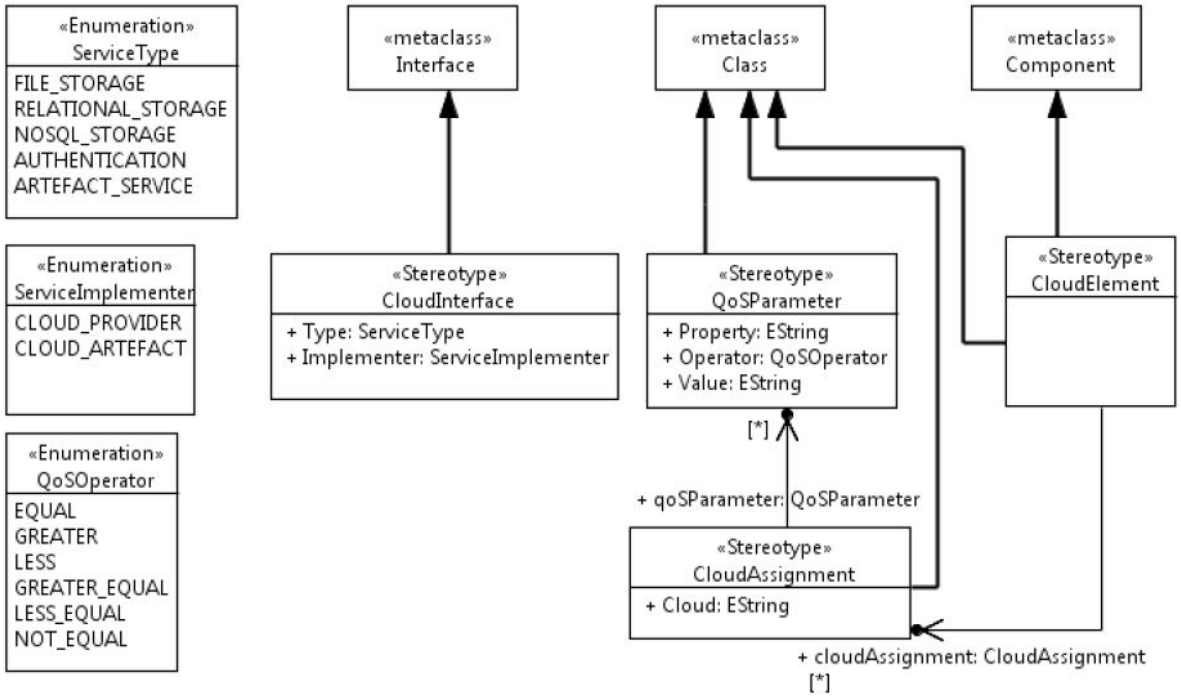


Figure 2: UML profile for cloud application modelling.

### 3.2 Coding and Deployment Configuration

During this phase developers will code the application's functionality starting from the skeleton classes generated in the previous phase. Additionally, tools will be integrated in the development environment in order to allow them to configure the cloud deployment plan.

This configuration will imply the assignment of each artefact to a specific cloud platform based on a catalogue of supported platforms. Hence, cloud specific information regarding the following points will automatically be included in the deployment plan per cloud artefact:

- Services provided and consumed by the cloud artefact for interoperability with other components that will be deployed in different clouds
- Vendor specific core services consumed by the cloud artefact.

The technological restrictions of each cloud platform regarding these issues will also be automatically generated and included in the deployment plan in order to generate cloud compliant software components during this phase. This is possible because the approach encloses a feature model (an engineering paradigm frequently used in the scope of Software Product Lines) that documents the variability of each cloud platform. This feature model provides a knowledge base containing all the specific features of each cloud platform; it documents the supported service protocols, the required configuration parameters, the cloud specific services that are provided, etc.

Notice that we have chosen to include this intermediate step instead of directly generating the cloud compliant source code in order to make it easier for developers to code the application's functionality. Directly generating the source ode used for component-to-component or component-to-cloud interoperability would hinder the development process by forcing developers to integrate their source code into complex generated code. Furthermore, it would also be detrimental for round-trip engineering and application's maintainability.

### 3.3 Cloud Artefact and Adapter Generation

The source code and the deployment plan generated in the previous phase will be processed during this stage to produce cloud compliant artefacts. Considering that each cloud platform may impose a different structure to its software projects, the cloud artefacts generated in this phase will be enclosed in predefined source code projects that also contain a series of software adapters.

As a part of this phase, we propose the use of Software Adaptation techniques in order to allow a flexible solution for overcoming variability and interoperability issues in clouds. SA techniques are aimed at developing mediator elements, called adapters, which are automatically built from their correspondent specifications and granting interoperability between mismatching software elements.

Adaptation will be performed at any of the different levels of interoperability (Becker et al., 2004), depending on the needs of each cloud artefact. The following section discusses in more detail the different sources of mismatch we have found in migratable multicloud SBAs. The detection of mismatch, and the automatic generation of the required adapters will help to achieve interoperability between cloud artefacts deployed through heterogeneous cloud providers.

## 4 ADAPTATION IN THE CLOUD

Software Adaptation is a field within Software Engineering which aims at providing the abstractions and non-intrusive techniques for composing mismatching black-box components by automatically generating adapters able to reconcile interoperability problems among them. We may distinguish different levels of interoperability at which mismatch may occur (Canal et al., 2006), and different adaptation techniques are applied to each of these levels:

- *Signature level.* Deals with the static aspects of interface interoperability, including operation names, type of arguments and return values, and exception types.
- *Behavioural level.* Describes the interactive behaviour that an artefact follows and expects from its environment, (i.e., the order in which the operations available on an interface should be invoked). Behavioural descriptions are required for stateful artefacts since operation availability depends on the internal state of the component.
- *Service level.* Deals with the description of QoS properties like temporal requirements, security, cost, etc.

- *Conceptual level.* This level concerns semantic specifications (i.e., what the artefacts actually do). Even in the absence of mismatch at any of the preceding levels, we must ensure that the artefacts are going to behave as expected during their interactions.

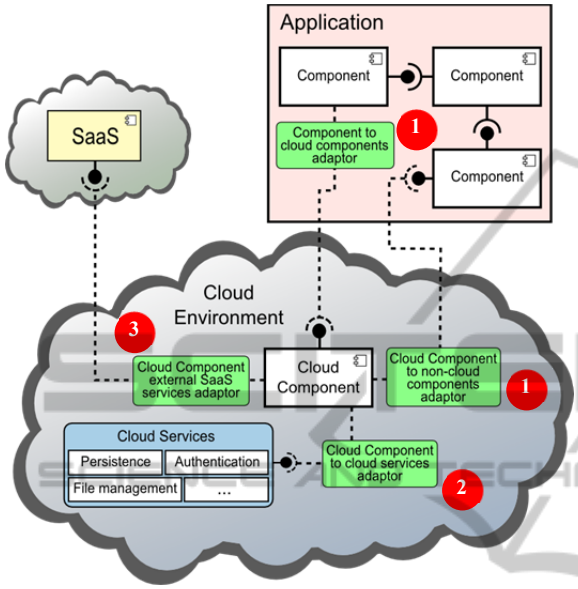


Figure 3: Cloud deployment scenario with adapters.

As we presented in (Guillén et al., Sept 2012), three different situations where adaptation would take place have been considered in our approach, corresponding to the numbered tags in Figure 3:

Case 1. *Adaptation between components.* The way in which a component deployed in a cloud environment communicates with another cloud component or a component deployed in a non-cloud environment differs substantially from a fully in-house approach.

Case 2. *Adaptation between components and cloud services.* Cloud environments usually provide specific services which can be consumed by components deployed on their platforms. The most common examples of these services are those related with persistence, security/authentication, file management, etc.

Case 3. *Adaptation using third-party components or services (SaaS).* Cloud deployed components consume external services differently depending on the cloud provider in which they have been deployed. The services being invoked and their location also determine how they must be consumed.

In the following subsections, we further develop the sources of mismatch presented in (Guillén et al., Sept 2012) , classifying them into four adaptation situations, each one related to one of the levels of interoperability, and propose the use of specific interface description and adaptation techniques for addressing them.

## 4.1 Signature Adaptation

Signature adaptation deals with mismatching service and operation names, argument types and ordering, and naming conventions imposed by the underlying technologies. This is the most frequent mismatch issue that appears when trying to combine software artefacts independently developed by third parties, and consequently it may appear in any of the three adaptation cases shown in the cloud scenario in Figure 3. Either when migrating a component from in-house to cloud hosting, or from a cloud platform to a different one, both the location and name of its services and operations will change, when they are invoked by the remaining components in the application, and also the way it invokes services and operations from the rest of the system will be affected. Furthermore, even though the most common services that the migrated component may require are provided by every cloud provider, their names and arguments largely differ from one cloud platform to another. Finally, the decision of hosting a component in a particular cloud platform, may also impose limitations on the distributed component technologies available (e.g. SOAP, Rest, Java RMI, etc.) which impose variations in the way these services are invoked.

Among the different kinds of mismatch, signature mismatch is the easiest to solve, and it has been customarily addressed by non-invasive techniques, like wrappers, or proxies. More specific proposals (see as an example (Canal et al., 2008)), advocate for the use of adaptation contracts or mappings, abstract specifications of how mismatch can be solved, based on the description of component interfaces and service APIs. These mappings establish correspondences between dissimilar names of operations in the interfaces of the artefacts to be adapted, allowing also reordering and synthesis of operation arguments when required. From them, the corresponding adapters can be generated (Canal et al., 2008), allowing successful interaction of the counterparts despite the different naming conventions. For instance, the adapter can easily transform what for the invoking component seems to be a local call to its target component or

cloud service, while solving any kind of signature mismatch by translating the name of the operation, redirecting to a remote location if required, and switching between different naming conventions and invocation technologies.

## 4.2 Behavioural Adaptation

Behavioural mismatch refers to different granularity of services and operations, and to incompatible interaction protocols, in which the partial ordering of operations does not fit among the counterparts engaged on a service transaction. This kind of mismatch can only be present in stateful software artefacts, as many complex cloud services are. Again, behavioural mismatch may be detected in any of the three adaptation cases in Figure 3. For instance, a frequent scenario of behavioural mismatch appears when a stateful service is deployed in a cloud environment that only supports SOAP invocations. Here, WS-Resource and WS-Addressing mechanisms can be used to perform changes in the communication schema in order to continue supporting a stateful behaviour.

Even more typically, the operation granularity and ordering of equivalent cloud services may differ between vendors. For instance, the same generic 'get' operation of a persistence service could imply several different operation invocations and/or orderings from one provider to another one. The same occurs when comparing equivalent external services.

Several recent research efforts (see as an example (Canal et al., 2008); (Seguel et al., 2010) concentrate on behavioural interoperability, extending interfaces with a description of the protocol followed during interactions, using either automata-based notations or industrial standards like WS-BPEL, and ensuring their correctness and termination. The works in this category explain how to generate adapters able to capture, store, remember and reorder messages and operation calls and their arguments, that are transmitted to their destinations only at the point they are prepared to receive them. For that, and starting from an empty or null behaviour, the adapter is extended with input/output actions reflecting those of the counterparts to be adapted, and that serve to remove a deadlock in their interaction, while at the same time ensuring other properties of interest which can be specified by the designer. The process continues until all deadlocks have been removed, returning a full adaptor able to ensure successful and correct interaction among the participants.

## 4.3 QoS Adaptation

Service mismatch refers to interoperability issues related to non-functional properties, such as client QoS requirements and SLAs offered by cloud platforms Thus, this kind of mismatch is likely to appear in both component-to-cloud and component-to-SaaS interactions. Indeed, each cloud provider defines a specific API for cloud intrinsic characteristics. Again, migrating a component to a different cloud will affect the way in which QoS and other non-functional properties are queried and established. Similarly, providers enforce very different SLAs for their services. Hence, switching between service providers may cause mismatch between the SLA imposed by the cloud and the QoS required by the application.

Service adaptation is probably the least explored adaptation level. QoS description models and their related notations, such as the QoS Modeling Language (QLM), are usually highly customizable, and the possible specifications include mean values, standard deviations and a set of quantiles characterizing the distribution of any self-defined quality metric. This ensures that QoS mismatch can be easily detected, but once the mismatch is found not so many actions can be performed to solve it, especially if the mismatch comes from the SLA policies enforced by a cloud provider. In that case, only changing to a different cloud or service provider, or replicating a component among different clouds may help in solving the problem.

## 4.4 Conceptual Adaptation

Conceptual mismatch is produced by differences in the functionality of the services offered and requested. This kind of mismatch is mainly related to the use of external services, and hence, it may appear in the third adaptation case in Figure 3.

Indeed, in order to avoid external dependencies, the consumption of external services must be specified without determining the identity and location of the targeted service. For example, a component may indicate that it requires a translation service, without specifying the exact service it is going to consume. During the adaptation process, the most convenient matching service will be selected according to the requirements.

In the service-oriented computing field, there are several notations for providing semantic information about services using ontology-based notations such as OWL-S or WSMO, which are particularly interesting for service discovery. Once a semantic

match is selected, adaptation will be performed at the remaining levels, in order to ensure correct interaction.

# 5 RELATED WORK

Considering that no standardization initiative has been consolidated, and that the outcome of the existent initiatives is yet unknown, different proposals have been made to mitigate this absence. This section describes the cloud application migratability and interoperability proposals that are most closely related to ours.

In the scope of MDE for the cloud, (Hamdaqa et al., 2011) proposes a meta-model for modelling cloud applications focused in the definition of cloud tasks as composable units, each one consisting of a set of actions that make use of services to provide a specific functionality. The approach detaches the application modelling process from specific cloud platforms; nevertheless, model transformations will generate code that will be coupled to a specific cloud platform. Our approach models cloud applications from a different perspective since it allows us to generate source code that is not coupled to the cloud. Instead, all cloud related data is included in a separate deployment plan, thereby favouring the code's cloud agnosticism and maintainability.

Another proposal based on MDE techniques is presented by (Frey and Hasselbring, 2010) as a means of mapping models of existent cloud environments to legacy software models and transforming the result to cloud-specific code through a series of iterations and result evaluations. This approach is fully oriented towards legacy software; additionally, any subsequent changes will have to be integrated into the generated software, which may result more difficult to work with and understand by the developers. In our work a different approach for modelling the variability of cloud platforms, based on feature models, is used based. Both newly developed and legacy software can be migrated easily to the cloud without coupling the application's source code to a cloud platform.

Other proposals for combating the vendor lock-in effect are based on the use of middleware and intermediate software layers intended to create an abstraction between cloud platforms and the generated software. One of the closest to our work is mOSAIC, a reference initiative carried out as a Europe funded project. Its perspective of how cloud development should be accomplished matches our criteria. Cloud migratability, interoperability and the deployment of applications across more than one cloud is tackled in mOSAIC through a robust solution based on an API and a middleware platform for cloud brokering (Di Martino et al., 2011). Our approach deals with these issues through a different perspective based on the use of SA techniques, which present beneficial results in alternative scenarios where lightweight software components may be required.

A Service Oriented Cloud Computing Architecture (SOCCA) is presented in (Tsai et al., 2010). An architecture is provided where cloud computing resources are componentized, standardized and combined in order to build a "cross-platform virtual computer" which operates upon an ontology mapping layer that is used to abstract the differences between cloud providers. SOCCA applications can be developed using the standard interfaces provided by the architecture or the platform unique APIs of a cloud provider. In both cases the developed applications will be coupled to a specific platform, thereby hindering their migration to alternate scenarios. In our approach the applications are not coupled to any platform; the use of SA allows us to easily migrate components between clouds without having to modify their source code.

# 6 CONCLUSIONS AND FUTURE WORK

In this paper we have presented an alternative approach to the existent standardization initiatives and middleware-based solutions for achieving cloud application migratability and interoperability. This solution has been conceived with cloud providers and customers in mind, trying to offer them maximum flexibility.

The underlying technology and tools for putting our approach into practice is currently under development, as a proof-of-concept development framework. Our future work includes the extension of its current capabilities, supporting more cloud platforms and programming languages, as well as exploring the use of dynamic adapters in order to support run-time changes in cloud artefact compositions.

# ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish

Government under Projects TIN2012-34945, and TIN2012-35669.

## REFERENCES


N. Leavitt, "Is cloud computing really ready for prime time?" *Computer*, vol. 42, no. 1, pp. 15–20, Jan. 2009.

M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

D. Petcu, G. Macariu, S. Panica, and C. Craciun, "Portable cloud applications - from theory to practice," *Future Generation Computer Systems*, Jan. 2012.

B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The reservoir model and architecture for open federated cloud computing," *IBM J. Res. Dev.*, vol. 53, no. 4, pp. 535–545, Jul. 2009.

A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to enhance cloud architectures to enable cross-federation," *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 337 – 345.

N. Loutas, E. Kamateri, F. Bosi, and K. Tarabanis, "Cloud computing interoperability: The state of play," *Cloud Computing Technology and Science, IEEE International Conference on*, pp. 752–757, 2011.

B. Di Martino, D. Petcu, R. Cossu, P. Goncalves, T. M´ahr, and M. Loichate, "Building a mosaic of clouds," *Euro-Par 2010 Parallel Processing Workshops, ser.* Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, vol. 6586, pp. 571–578.

W. Tsai, X. Sun, J. Balasooriya. "Service-Oriented Cloud Computing Architecture," *ITNG10 7th International Conference on Information Tech-nology: New Generations*, pp. 684-689, 2010.

E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson. "Toward cloud-agnostic middle-wares," *OOPSLA09: 14th conference companion on Object Oriented Programming Systems Languages and Applications*, pp. 619–626, 2009.

J. Guillén, J. Miranda, and J. M. Murillo. "Decoupling Cloud Applications From The Source - A Framework for Developing Cloud Agnostic Software," *Proceedings of CLOSER 2012*, Oct. 2012.

J. Guillén, J. Miranda, J. M. Murillo, and C. Canal. "Identifying Adaptation Needs to Avoid the Vendor Lock-in Effect in the Deployment of Cloud SBAs," *Proceedings of WAS4FI 2012*, Sept. 2012.

D. K. Nguyen, F. Lelli, Y. Taher et al.. "Blueprint template support for engineering cloud-based services," *Proceedings of ServiceWave'11*, pp. 26–37, 2011.

S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky, and M. Tivoli. "Towards an engineering approach to component adaptation," R.H. Reussner, J.A. Stafford, and C.A. Szyperski, editors, *Architecting Systems with Trustworthy Components*, vol. 3938 of Lecture Notes in Computer Science, pp. 193–215. Springer, 2004.

C. Canal, J. M. Murillo, and P. Poizat. Software adaptation. Special Issue on Coordination and Adaptation Techniques for Software Entities. *L'Objet*, 12(1):9–31, Hermes-Lavoisier, 2006.

C. Canal, P. Poizat, and G. Salaün. Model-based adaptation of behavioural mismatching components. *IEEE Transactions on Software Engineering*, 4(34):546–563, 2008.

R. Seguel, R. Eshuis, and P. Grefen, "Generating minimal protocol adaptors for loosely coupled services," *Web Services, IEEE International Conference on*, pp. 417–424, 2010.

M. Hamdaqa, T. Livogiannis, and L. Tahvildari, "A reference model for developing cloud applications." *CLOSER*, pp. 98–103. SciTePress, 2011.

S. Frey and W. Hasselbring, "Model-Based Migration of Legacy Software Systems into the Cloud: The CloudMIG Approach," *Proceedings of the 12th Workshop Software-Reengineering (WSR 2010)*, May 2010, pp. 59–60.