Developing migratable multicloud applications based on MDE and adaptation techniques

Joaquín Guillén
Javier Miranda
Gloin
Calle de las Ocas 2, Cáceres, Spain
{jguillen,jmiranda}@gloin.es

Juan Manuel Murillo
Department of Information Technology and Telematic Systems Engineering
University of Extremadura, Spain
juanmamu@unex.es

Carlos Canal
Department of Computer Science
University of Málaga, Spain
canal@lcc.uma.es

ABSTRACT

Developing software for the cloud usually implies using the tools and libraries supplied by cloud vendors for each of their platforms. This strongly couples the software to specific platforms and penalizes its migration or interoperability with external cloud services, in what is known as vendor lock-in. Under these circumstances multicloud applications become difficult to build and maintain since they require multidisciplinary teams with expertise on multiple platforms, and the redevelopment of some components if the cloud deployment scenario is altered. The MULTICLAPP framework described in this paper tackles these issues by presenting a three-stage development process that allows multicloud applications to be developed without being coupled to any concrete vendor. MDE and adaptation techniques are used throughout the software development stages in order to abstract the software from each vendor's service specifications. As a result of this, multicloud applications or their subcomponents can be reassigned to different cloud platforms without having to undergo a partial or complete redevelopment process.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: Model Development; D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.12 [**Software Engineering**]: Interoperability

Keywords

Cloud, Multicloud, UML Profile, Vendor lock-in, Framework

1. INTRODUCTION

Cloud Computing [15] is an emerging technology that has experienced a great industrial acceptance over a very short period of time. Even though it has been considered by some as the culmination of concepts such as Grid, the utility computing business model, service-oriented architectures and Web 2.0 technologies, the truth is that its current service models (SaaS, PaaS and IaaS) are barely a decade old. A great demand for this technology has created an industrial gap that many companies, further referenced as cloud vendors, have fleeted to cover in a quest for new business horizons that would allow them to increment their profits on one hand, and to pay off the investment made in large server farms on the other. The urge to win as much market share as possible and to tie users to their products has led vendors to define proprietary services without considering their interoperability or compatibility with those defined by their competitors.

The systems developed and used by cloud customers are constructed according to the proprietary services supplied by each vendor, such that they become fully dependent on the former and thereby run into what is known as vendor lock-in [13]. This is extremely beneficial for vendors since it discourages customers from migrating their products to different clouds. However its a great drawback for customers, who jeopardize the freedom to evolve their software and become vulnerable to price increases, reliability problems, or to the possibility of their vendors going out of business [4].

Nevertheless customers have to deal with other issues which are not as intrinsic to their vendors as the aforementioned problems. Software systems can be comprised of different components with different sets of requirements that may not be fulfilled by a single cloud and may therefore require a multicloud scenario where components are deployed in different cloud environments. Hybrid cloud environments are commonly used by firms to provide services to their customers through public cloud infrastructures and to manage their corporate information using private cloud infrastructures. The applications developed for these environments pose a clear example of the use of multicloud scenarios.

Developing those types of applications is currently a big challenge due to the lack of tools and IDEs designed for that purpose. It implies using the different tools and libraries supplied by each cloud vendor, and manually implementing the services that allow the distributed cloud components to interoperate with one another.

These shortcomings are being addressed by industry and academia in an attempt to provide an added value to the existent cloud services by allowing customers to freely develop applications that consume such services without being

NordiCloud 2013, 2-3 September, Oslo, Norway.

tied to a particular platform. Nevertheless, as we discuss in this paper, the existant initiatives are still at a very early stage and therefore do not accomplish solving all of the problems [17].

In this paper the development process followed by the MULTICLAPP (MULTICLoud migratable and interoperable APPlications) framework is presented as an alternative for mitigating these shortcomings. The framework follows a three-stage development process where applications can be modeled and coded without developers having to be familiar with the specification of any cloud platform. The applications, which are developed as for in-house environments, can be assigned to cloud platforms without this having any impact upon the platform-independent models or the source code. Thereafter, cloud compliant software projects are created and adaptation techniques are applied in order to automatically generate service adapters that allow the software to interoperate with specific cloud services as well as with dependent components deployed in remote environments. Migrating parts of an application to different clouds requires developers to reconfigure its deployment plan with the graphical editors integrated in the development environment, and regenerating the cloud compliant software projects and adapters, thereby not having to change the application's source code.

Here we provide a big picture of the MULTICLAPP framework by summarizing the work presented in [16] [18] [10] on one hand, and by extending it with the advances made upon the application modeling stage on the other hand. We focus on presenting the main ideas that lie behind the framework, and the development processes followed throughout each of its stages. The tools used by the framework for automating parts of the development processes in each of the development stages are currently under development.

The remainder of this paper is structured as follows. In Section 2 we present the motivations of our work and review the types of approaches that exist for combatting vendor lock-in in the development of migratable multicloud applications. Section 3 contains a description of the MULTICLAPP framework where we present each of the three development stages that it is based on. Additional work related to the framework is reviewed in Section 4. Finally the conclusions and future lines of work are commented in Section 5.

2. BACKGROUND AND MOTIVATIONS

Due to the short period of time that Cloud Computing, as we know it today, has been around, the development of software in this field is still highly dependent on the platform where it will be hosted. Providers define their own sets of tools, libraries and technological restrictions for building software for their platforms, thereby creating a strong dependency between the software that is developed and their technological infrastructure. Developing multicloud applications that can be migrated to different clouds becomes extremely difficult if only the vendors' tools are used, thereby making third party solutions indispensable.

Industry and academia have come up with initiatives for solving these shortcomings that can be classified into three groups: standardization initiatives [2, 9, 1], middleware-based initiatives [19, 22, 14], and Model-Driven Engineering (MDE) initiatives [6, 3, 11]. Each of them solve some of the aforementioned problems, but also present issues that are worth noting.

Standardization initiatives, which have been mostly proposed by the industry, seek to standardize the services provided by cloud vendors, such that common interfaces are declared by services of the same type. The main standardization initiatives [2, 1, 9] were summarized in [16], where we remarked that neither of them have yet been undertaken by any of the market's main cloud vendors. This is because they stand against their interest of keeping customers tied to their technology and limit their competitive advantage amongst other vendors. Furthermore, since several initiatives coexist, standardization runs the risk of more than one standard being accepted, in which case compatibility between standards would also become cumbersome. We thereby consider that standardization initiatives must not be taken as the only means of combating vendor lock-in.

An alternative to standardization consists on the use of middleware-based solutions that implement a software layer that lies between the applications and the cloud services in order to abstract developers from the peculiarities of each vendor's services. They provide a unified API that allows developers to consume services homogeneously, and internally manage the conversion of API invocations to cloud specific service invocations. Furthermore, some solutions include databrokers capable of matching the QoS requirements of the applications with the SLAs of each vendor in order to determine which platform is most suitable for the application, and to manage its deployment in the platform. Middleware-based approaches constitute robust solutions and are popular both in academia [19, 22, 14] as well as industrially [20, 12, 8], however they behave as virtual clouds and couple the software developed for them to their specification, similarly to how conventional platforms do. Furthermore, middlewares are usually complex software entities that may result exceedingly heavy to deploy for applications that do not require an extensive use of their functionality. It is also worth noting that middlewares require a constant maintenance in order to support the changes made by vendors upon their platforms, and this may be a problem for some projects once they finalize.

MDE techniques have also been used in different works [6, 3, 11] to mitigate the effects of vendor lock-in. These approaches rely on models as a means of abstracting the development process from the peculiarities of each cloud platform. Applications are modeled independently of the cloud platform that they will be deployed in, and transformations are applied upon them until platform specific source code is automatically generated. This keeps developers from having to be familiar with the technical details of each platform's services, however these solutions do not generate the complete source code of an application and thereby require developers to be able to work on the automatically generated code to fully implement the system's functional behaviour.

All of these approaches attempt to mitigate the difficulties associated with the development of multicloud applications following techniques that are applied during different phases of the development process: middleware-based and standardization initiatives are useful during the application's coding phase since they provide a unified low level definition of the services, whereas MDE based initiatives can be applied during the application's design since they homogenize cloud services at a higher abstraction level. However most of these approaches do not cover a muticloud application's complete development cycle.

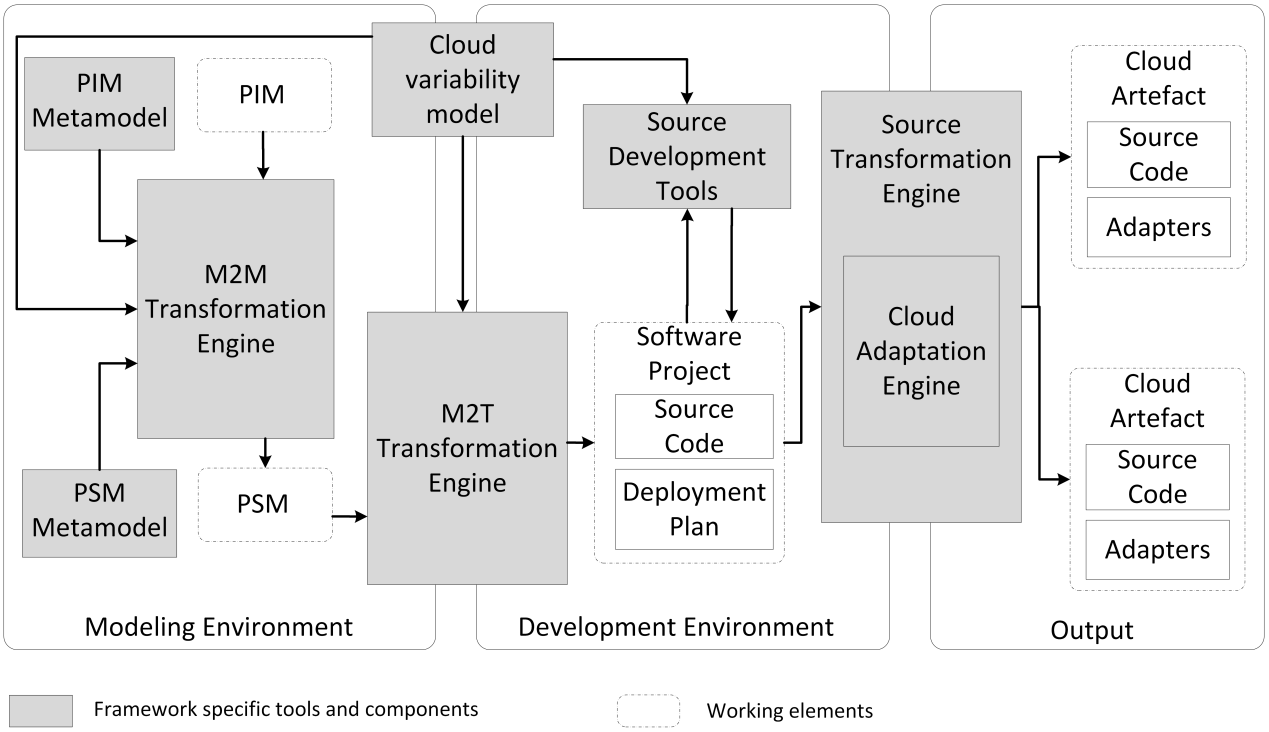

Figure 1: Cloud application model excerpt

These shortcomings have motivated the design of the MULTICLAPP framework, which assists developers in the construction of multicloud applications during their design, their development and final deployment in the cloud/s.

3. MULTICLOUD APPLICATION DEVELOPMENT BASED ON MDE AND ADAPTATION

In this section we describe the MULTICLAPP framework and analyse each of the development stages that it covers. In Section 3.1 the framework's high level design is analysed and its main definitions are given. Section 3.2 describes how multicloud applications are modeled. The model transformation towards cloud-agnostic source code and a cloud deployment plan, and the coding stage are described in Section 3.3. Finally, the transformations applied upon these elements to generate cloud compliant projects and service adapters are detailed in Section 3.4.

3.1 High-Level Design and Definitions

MULTICLAPP allows analysts and developers to construct migratable multicloud applications, and relieves them from tasks related with the integration of their components with vendor specific services or their interoperability with components located in different clouds. MULTICLAPP defines a three-stage development process [16] consisting on an initial modeling stage, a functional behaviour coding stage, and a final cloud artefact generation and deployment stage. Each of these stages are represented in Fig. 1 where the framework's high level design is illustrated.

The framework interprets multicloud applications as a composition of software components that can be deployed in different platforms, such that each component may interoperate with others found in remote platforms as well as with the vendor specific services of its platform [18]: we call these groups of components *cloud-artefacts*.

Cloud-artefacts can be assigned to a specific cloud platform during the modeling and coding stages. The assignment of an artefact to a specific platform requires the framework to have a catalog of the existent platforms and to gather knowledge about each of their services and requirements. This information is gathered in the Cloud Variability Model included in Fig. 1. It has been implemented as a feature model[1], and it contains information about the features, services, SLAs and requirements of each of the supported platforms. The feature model is described thoroughly in [10] where we explain that the model has been constructed hierarchically, such that the top level is used to differentiate each of the supported platforms and the underlying levels describe specific features of each cloud platform, including their service definitions categorized by service type. Currently the feature model supports the Microsoft Azure and Amazon Beanstalk platforms, for which a subset of their services have been modeled.

By keeping all the knowledge relative to the supported platforms in the variability model, MULTICLAPP manages to isolate architects and developers from the technical requirements and specifications of each platform. Furthermore, the elements with which they work with (models and source code), are also kept clean from the specifics of each platform. That is, cloud-artefacts contained in the platform-independent model are not assigned to a specific platform,

[1]Feature models are commonly used in Product Line Engineering. They provide a representation of all the products in a Product Line.

and the source code that implements the application's functional behaviour is always cloud-agnostic. All cloud-related information managed during the coding stage is kept in a separate deployment plan.

3.2 Cloud Application Models

The first stage of the development process supported by the MULTICLAPP framework allows architects and designers to model multicloud applications in order to easily and graphically identify the cloud-artefacts that comprise a multicloud application before engaging in the functional coding stage.

Multicloud applications are modeled without taking into consideration the specifics of any platform. Instead, they define a set of cloud-artefacts that interoperate with one another as well as with the services of the platform where they are deployed. A simple UML profile has been implemented for this purpose by the platform-independent metamodel (PIM) illustrated in Fig. 2.

As illustrated in the metamodel, classes and components that implement the application's functionality are tagged with the *CloudArtefactElement* stereotype that allows them to be assigned to one or more cloud-artefacts. Belonging to more than one cloud-artefact allows *CloudArtefactElement*s to finally be deployed in more than one platform, which is useful when common software elements have to be used in multiple artefacts. An example of this would be an application that requires user authentication in all of its artefacts, such that each of them validate the invoker's credentials. If all the artefacts use the same authentication procedure, a unique *Authentication* component can be designed and assigned to both cloud-artefacts.

The interoperability needs of artefacts with their accessible cloud services and with remote artefacts deployed in different platforms are modeled using the *CloudArtefactInterface* stereotype for UML interfaces. Each *CloudArtefactInterface* contains tagged values to indicate whether it is an interface for interoperability with remote artefacts implemented by a cloud-artefact (CLOUD_ARTEFACT) or an interface to interoperate with a local cloud service (CLOUD_PROVIDER). In both cases the system architect is free to design the interface without having to consider the specifications of any service vendor. The adaptation of these interfaces to each vendor's specification is tackled during the third stage of development process described in Section 3.4.

Finally, multicloud PIMs also model the non-functional requirements of each artefact. This is done through the 1-to-n association that exists between the *CloudArtefact* stereotype and the *QoSParameter* stereotype, which associates a set of QoS expressions to an artefact. Each expression is constructed using the three tagged values (Property, Operator and Value) of the *QoSParameter* stereotype: [Property = 'PCT UPTIME', Operator = GREATER EQUAL, Value = '90'] is an example expression that determines that the cloud-artefact requires a 90% plus uptime.

Once an application's PIM has been modeled it is then transformed to a platform-specific model (PSM) by the model-to-model (M2M) transformation engine. This implies taking a decision about the platforms where each of the artefacts should be deployed. The PSM metamodel is an extension of the UML profile illustrated in Fig. 1 that allows models to incorporate the architect's decisions about the cloud that each artefact is deployed in, and how it interoperates with its required cloud services and remote artefacts. The following points summarise the decisions that must be taken by architects to assign an artefact to a cloud platform:

- **Service types:** The type of services provided by artefacts for interoperability with remote artefacts have to be chosen. MULTICLAPP currently supports SOAP and REST services.
- **Service matches:** Some platforms provide different services of the same type. If this is the case architects must choose the final cloud-specific services that each artefact interoperates with through the interfaces that are specified for such purpose.
- **Adaptation contracts:** The interfaces defined for interoperating with cloud services will naturally differ from those exposed by each vendor. These differences are known as mismatches. Their resolution requires architects to define mappings [7, 21] that solve them by establishing correspondences between their operations and parameters, thereby defining an adaptation contract for each interface.
- **Configuration parameters:** Each platform requires different sets of parameters that must be configured by the architect.

These decisions are taken by architects during the execution of the M2M transformation engine, which provides a graphical decision support system for assigning cloud artefacts to cloud platforms. The engine matches the non-functional requirements of each artefact with the SLAs of the platforms included in the variability model and makes recommendations to the architect about the most suitable platform for its deployment. Once the deployment platform/s are chosen for each artefact, architects are prompted to take each of the aforementioned decisions following a user friendly process that uses the tools integrated by the framework into the Eclipse IDE. This information allows the engine to execute the ATL[2] transformations required to generate a multicloud PSM.

3.3 Coding the Application's Functional Behaviour

In its current state of development MULTICLAPP only supports the construction of Java coded applications, nevertheless the framework's design is extensible to new programming languages. Java was chosen as a starting point for the framework since it is supported by most cloud platforms. Hence, MULTICLAPP software projects are Maven[3] projects based on the *webapp* archetype, that contain the application's source code and an XML-coded deployment plan where all cloud-related information is kept. The project's POM[4] file is preconfigured in order to integrate the basic libraries and tools used by the framework. Amongst these tools lies the Source Transformation Engine, which has been implemented as a Maven plug-in in order to allow cloud-artefacts to be generated as part of the source code's compilation process.

[2]Eclipse ATL - http://www.eclipse.org/atl/
[3]Maven - http://maven.apache.org/
[4]A Project Object Model (POM) is an XML representation of a Maven project

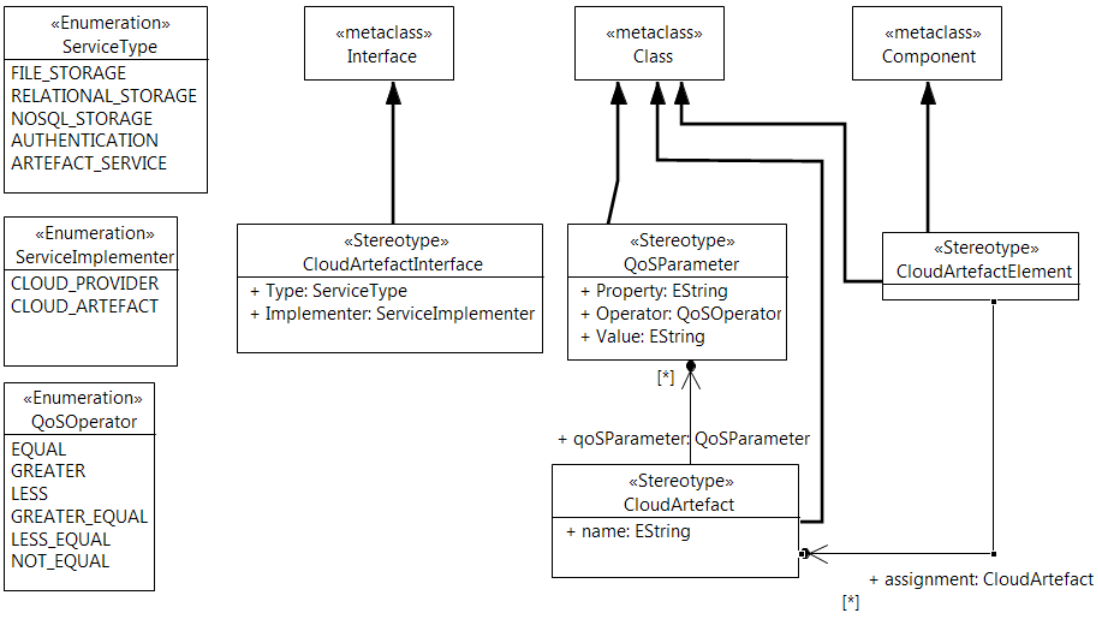

Figure 2: PIM Metamodel

In [10] we describe the framework's coding and cloud-artefact generation stages in depth, and we present a schema for defining multicloud application deployment plans. Deployment plans contain all the cloud-related information required by the framework to generate the final artefacts that can be deployed in each platform. That is, the architect's artefact assignments to specific platforms, their corresponding service mappings, the adaptation contracts used to resolve mismatches between the artefact interfaces and the platform-specific service implementations, and the configuration parameters required by each platform. This is a subset of the information found in the multicloud PSMs described in section 3.2 and is therefore automatically generated from them without human intervention through model-to-text transformations.

The model-to-text transformations are carried out by the M2T transformation engine integrated into the Eclipse IDE, which transforms an application's PSM to a software project where developers can start to code its functional behaviour. These transformations are implemented by Java Emitter Templates[5] (JET) that generate class skeletons for each of the classes included in the models and, most importantly, the application's deployment plan. PSM-to-text transformations generate full deployment plans that can be processed by the Source Transformation Engine described in Section 3.4.

Other than generating the software projects used by developers for the application's coding stage, the tools integrated into the Eclipse IDE also allow developers to modify the contents of their deployment plan without having to manually edit its XML code. Hence, new cloud-artefacts can be defined by selecting the set of classes that they are composed of and the interfaces used by each to interoperate with remote artefacts and with cloud services. Artefacts can be assigned or reassigned to any of the platforms found in the cloud variability model, in which case the assignment decisions listed in 3.2 have to be taken and the required information has to be inserted into the deployment plan's graphical editor.

By following this approach developers become free to code the functional behaviour of the applications as they wish and only have to be aware that the software will be deployed in the cloud if they choose to modify its deployment plan using the tools provided for this.

The way in which a multicloud application's coding stage is initiated depends on whether it has previously been modeled. Amongst the tools that have been integrated into the Eclipse IDE, a project creation assistant is supplied to create MULTICLAPP compliant projects with no source code and a basic deployment plan where no cloud-artefacts are included. This allows developers to easily start to code the application's functional behaviour without having to previously model the applications, or manually set up a new project and initialize a deployment plan.

3.4 Cloud-Artefact Generation and Deployment

Cloud-artefacts are generated by the Source Transformation Engine illustrated in Fig. 1, which is implemented as a Maven plug-in that is automatically invoked during a MULTICLAPP project's compilation, thereby allowing artefacts to be automatically generated without any additional intervention from developers.

The engine processes the deployment plan and generates and configures the cloud-artefacts. This is followed by the execution of the Cloud Adaptation Engine, which is responsible for automatically generating the adapters that allow cloud-artefacts to be integrated into their corresponding cloud platforms. As we described in [18], two types of adapters have to be generated: adapters that allow remote artefacts to interoperate with one another, and adapters that allow artefacts to interoperate with the services provided by their cloud environment. Both adapters require signature and behaviour level adaptation to be applied, nevertheless gener-

[5]http://www.eclipse.org/modeling/m2t/?project=jet#jet

ating the former is a lot simpler since remote cloud artefacts use the same interface specification to interoperate with one another. In consequence, the adaptation engine's goal consists on providing the interface implementation as a SOAP or REST service and generating its corresponding client adapter. On the other hand generating the adapters required by artefacts to interoperate with platform-specific cloud services is slightly more complex since the interfaces used by artefacts to access the platform's services may differ greatly from the actual interfaces exposed by the platform. Generating these adapters requires the engine to process the adaptation contracts generated by architects and developers during previous development stages. These contracts map the operations and parameters of the artefacts' interfaces with those exposed by the services they have been mapped with.

The adapters generated by the adaptation engine follow the *Proxy* design pattern, and are generated in a similar way as in other adaptation approaches [5]. They are injected into each cloud-artefact's source code using the inversion-of-control (IoC) container provided by the Spring framework. This allows MULTICLAPP to keep classes that were invoking the cloud-agnostic interfaces unaltered thereby facilitating the replacement of adapters.

The final step of the engine's execution consists on copying each artefact's source code from the software project used during the functional coding stage into the newly created projects, and configuring the Spring IoC container. From that moment the automatically generated cloud-artefacts can be deployed in their corresponding platforms using each vendors' tools.

4. RELATED WORK

Globalization of the IT industry has aroused an interest for developing delocalized software systems that exploit the benefits of the cloud. Such interest coexists with many organizations' purpose of not being tied to any specific vendor, and having different parts of a system deployed across different cloud platforms. However these interests stand against vendor lock-in and therefore require the adoption of third party solutions on behalf of software developers.

In the scope of middleware-based solutions mOSAIC [19] must be mentioned as it is a reference initiative carried out as a Europe funded project that shares many of MULTICLAPP's motivations. Like MULTICLAPP, mOSAIC aims to provide a platform for developing multicloud applications that can interoperate with one another and with the services provided by each cloud. Their solution is based on an API and a middleware platform that creates an abstraction layer between the developed applications and the cloud in which it will be deployed. The technology is also complemented with cloud brokers that manage SLA negotiations with each cloud platform as well as the deployment of each application. The provision of cloud brokers that carry out the dynamic deployment of the applications is not contemplated by our framework, which unlike mOSAIC, does address their modeling stage.

Cloud4SOA[6] is a similar Europe-funded project focused on providing an homogeneous ecosystem of PaaS vendors. The unification of such ecosystem is mainly done by a semantic interoperability framework for matching application requirements with the supported resources offered by cloud vendors, and a harmonized API with most of the diversities of existing PaaS platforms. Adaptation tasks are also done to allow cloud services to be invoked in a uniform manner, nevertheless Cloud4SOA defines the API that must be implemented by all adapters. This differs from our approach where architects and developers are free to define the interface used to interoperate with cloud services.

Another middleware-based approach where an API is specified to access cloud resources homogeneously is presented in [22]. Here a Service Oriented Cloud Computing Architecture (SOCCA) is proposed where cloud computing resources are componentized, standardized and combined in order to build a cross-platform virtual computer. An ontology mapping layer is configured over these services as a means of masking the differences between cloud vendors. Similarly to mOSAIC, cloud brokers interact with the ontology mapping layer for deploying applications in one cloud or another depending on a series of parameters, such as the budget, SLAs and QoS requirements that are negotiated with each vendor. SOCCA applications can be developed using the standard interfaces provided by the architecture or the platform specific APIs defined by vendors.

These projects can be categorized in the group of middleware-based approaches. They allow multicloud applications to be developed by abstracting their implementation from the final platforms in which they are deployed. However in all cases this is achieved by defining their own APIs upon which applications are developed, such that invocations to the APIs are internally transformed to platform-specific service calls. This shifts the lock-in issue from the cloud vendors to the middleware technology, whereas in MULTICLAPP the applications' source code is not coupled to any specific APIs. Middleware-based approaches are often considered as virtual cloud platforms and can therefore also be supported by the MULTICLAPP framework. Hence it would be possible for multicloud applications to be constructed and deployed using mOSAIC if it were included in our cloud variability model.

Other projects related with the MULTICLAPP framework rely on MDE techniques for developing multicloud applications. MODAClouds [3] is another European-funded project which shares some of the motivations of the MULTICLAPP framework. The project is currently at an early stage of development and its goal consists on allowing system developers to design software in a cloud agnostic manner, allowing it to be instantiated and deployed across multiple clouds. MODAClouds uses three different levels of abstraction to model multicloud applications: a Computation Independent Model (CIM) in which non-functional requirements are modeled, a Cloud-Provider Independent Model (CPIM) where cloud concepts are introduced into the model but kept away from any specific platform, and a Cloud-Provider Specific Model (CPSM) in which the artefacts required by each platform are introduced into the models. Additionally a decision support system is integrated in order to determine the platform or platforms in which an application should preferably be hosted, and a run-time adaptation component is used to monitor and manage the execution of applications across several clouds.

Another MDE-based approach is presented in [6] under

[6]Cloud4SOA - http://www.cloud4soa.eu

the scope of the FP7 REMICS project[7]. CloudMl is presented as an extension of SoaML[8] for integrating into models information about the hardware and network resources that are required by applications, thereby allowing cloud instances to be generated by a CloudML engine with the modeled configurations. Systems are defined as a set of nodes with different hardware requirements (RAM, cores, disk space) that can be connected to one cloud or another through the use of specific connectors. Their work focuses on describing the virtual instances that have to be provided in each platform, and relies on the use of the JClouds [12] API to abstract the applications' generated source from the vendor specific services. This differs from our approach, which focuses more on modeling the cloud service and interoperability requirements of each artefact and does not contemplate the hardware characteristics of the virtual instances.

The use of models is also present in [11] where a metamodel for cloud applications is defined, centralized in the definition of a cloud task as a "composable unit, which consists of a set of actions that utilize services to provide a specific functionality". The metamodel aims to decouple the design process from specific cloud platforms, however the possibility of having different components of an application deployed in different clouds is not contemplated.

The use of MDE techniques in these projects intends to abstract the applications from the cloud platforms in which they are deployed. They rely on model transformations to generate software artefacts that can be deployed in their supported platforms and consume the services provided by each platform. Since applications can belong to any domain, in their current state of work none of these proposals generate the complete source code of an application directly from the models. They require developers to code the applications and integrate this source code into the artefacts generated from the model transformations. Managing this source code requires developers to be familiar with the peculiarities of the targeted cloud platforms. A possible solution to this is given by the MULTICLAPP framework by separating all cloud related information from the source code into an XML coded deployment plan, making the generated source code cloud agnostic and easier to interpret and maintain.

5. CONCLUSIONS

In this paper a general view of the approach followed by MULTICLAPP for developing multicloud applications has been presented. The development stages that it covers have been described and special emphasis has been put on explaining how the framework manages to keep applications decoupled from the environments where they are deployed. The separation of all cloud-related information from the applications' source code, and the use of adaptation techniques for integrating cloud-artefacts into each platform provides a non-intrusive development methodology. This allows software developers to construct their applications freely without being tied to any cloud vendor, and also relieves pressure from vendors so that they can freely evolve their services without loosing any competitive advantage.

This gains a high importance in the current scenario where developing cloud applications is strongly conditioned by vendors' interest of locking their customers in. We believe that third party frameworks and tools like MULTICLAPP should contribute to make cloud technologies more attractive to their potential customers by mitigating the risks that they assume with their adoption.

The tools and technologies described in this paper are currently under development and work is being carried out in different fields to enhance the framework's capabilities. At the modeling level the QFTP[9] UML profile defined by the OMG for modeling QoS requirements is being integrated into the framework in order to standardize how non-functional requirements are modeled for the cloud-artefacts. Furthermore, the variability model is being enhanced in order to support a wider range of cloud platforms. The current adaptation approach is also being reviewed and research is being done to support run-time adaptation in order to make it easier for artefacts to dynamically be redeployed in different platforms. The possibility of using quality attributes during the adapter generation process is also being explored in order to increase the adaptive capabilities of the artefacts.

6. ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish Government under Projects TIN2011-24278, TIN2012-34945 and TIN2012-35669. It has also been funded by the Government of Extremadura and FEDER funds.

7. REFERENCES

[1] Distributed Management Task Force. http://dmtf.org/standards/cloud. Last accessed 20/03/2013.

[2] Storage networking industry association - Cloud Data Management Interface (CDMI). http://www.snia.org/cdmi. Last accessed 20/03/2013.

[3] D. Ardagna, E. D. Nitto, P. Milano, D. Petcu, C. Sheridan, C. Ballagny, F. D. Andria, and P. Matthews. MODACLOUDS : A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. pages 50–56, 2012.

[4] 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*, 53(4):50–58, Apr. 2010.

[5] M. Autili, P. Inverardi, A. Navarra, and M. Tivoli. Synthesis: A tool for automatically assembling correct and distributed component-based systems. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 784–787, 2007.

[6] E. Brandtzaeg and S. Mosser. Towards CloudML , a Model-based Approach to Provision Resources in the Clouds. *Proceedings of the Model-Driven Engineering for and on the Cloud workshop (co-located with ECMFA'12)(CloudMDE'12)*, (257793), 2012.

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

[8] CloudBees. Cloudbees. http://www.cloudbees.com.

[9] A. Edmonds, T. Metsch, A. Papaspyrou, and A. Richardson. Toward an open cloud standard. *IEEE Internet Computing*, 16(4):15–25, Jul 2012.

[7]REMICS - http://www.remics.eu/

[8]SoaML - http://www.omg.org/spec/SoaML/

[9]Quality of Service and Fault Tolerance Characteristics and Mechanisms - http://www.omg.org/spec/QFTP/1.1/

[10] J. Guillén, J. Miranda, J. M. Murillo, and C. Canal. A service-oriented framework for developing cross cloud migratable software. *Journal of Systems and Software*, 2013. Currently in print.

[11] M. Hamdaqa, T. Livogiannis, and L. Tahvildari. a reference model for developing cloud applications. *Proceedings of the 1st International Conference on Cloud Computing and Services Science*, pages 98–103, 2011.

[12] JClouds. JClouds. http://www.jclouds.org/, 2011.

[13] N. Leavitt. Is cloud computing really ready for prime time? *Computer*, 42(1):15–20, Jan. 2009.

[14] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson. Toward cloud-agnostic middlewares. *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA 09*, page 619, 2009.

[15] P. Mell and T. Grance. The NIST Definition of Cloud Computing (Draft). *National Institute of Standards and Technology*, page 7, Jan. 2010.

[16] J. Miranda, J. Guillén, J. M. Murillo, and C. Canal. Enough about standardization, let's build cloud applications. In *Proceedings of the WICSA/ECSA 2012 Companion Volume*, WICSA/ECSA '12, pages 74–77, New York, NY, USA, 2012. ACM.

[17] J. Miranda, J. Guillén, J. M. Murillo, and C. Canal. Development of adaptive multi-cloud applications - a model-driven approach. In *MODELSWARD*, 2013.

[18] J. Miranda, J. M. Murillo, J. Guillén, and C. Canal. Identifying adaptation needs to avoid the vendor lock-in effect in the deployment of cloud sbas. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, WAS4FI-Mashups '12, pages 12–19, New York, NY, USA, 2012. ACM.

[19] D. Petcu, G. Macariu, S. Panica, and C. Craciun. Portable cloud applications - from theory to practice. *Future Generation Computer Systems*, (0):–, 2012.

[20] Rightscale. Multi-cloud platform, 2006.

[21] R. Seguel, R. Eshuis, and P. Grefen. Generating minimal protocol adaptors for loosely coupled services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 417–424, 2010.

[22] W.-T. Tsai, X. Sun, and J. Balasooriya. Service-oriented cloud computing architecture. *2010 Seventh International Conference on Information Technology: New Generations*, pages 684–689, 2010.