Towards Multi-Cloud Configurations Using Feature Models and Ontologies

Clément Quinton, Nicolas Haderer, Romain Rouvoy, Laurence Duchien
INRIA Lille - Nord Europe
LIFL UMR CNRS 8022
University Lille 1, France
{firstname.lastname}@inria.fr

ABSTRACT

Configuration and customization choices arise due to the heterogeneous and scalable aspect of the cloud computing paradigm. To avoid being restricted to a given cloud and ensure application requirements, using several clouds to deploy a multi-cloud configuration is recommended but introduces several challenges due to the amount of providers and their intrinsic variability. In this paper, we present a model-driven approach based on Feature Models (FMs) originating from Software Product Lines (SPL) to handle cloud variability and then manage and create cloud configurations. We combine it with ontologies, used to model the various semantics of cloud systems. The approach takes into consideration application technical requirements as well as non-functional ones to provide a set of valid cloud or multi-cloud configurations and is implemented in a framework named SALOON.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Methodologies, Representation*; D.2.13 [**Software Engineering**]: Reusable Software—*Domain engineering, Reuse models*

General Terms

Cloud Computing; Model-Driven; Configuration

Keywords

Cloud; Ontology; Feature Model; Multi-Cloud; Variability

1. INTRODUCTION

In cloud computing, resources are accessed on demand by customers and are delivered as services by cloud providers in a *pay-per-use* approach [1, 5]. This service provisioning model brings flexibility to companies that rely on cloud providers' infrastructure to run their applications. The whole software stack required to host the application can be either configured by hand in a *Virtual Machine* (VM) [21] or directly provided by cloud PaaS providers.

When deploying an application into the cloud, companies have to deal with a wide range of resources at different levels of functionality among available cloud solutions. This leads to complex choices which are usually made in an *ad hoc* manner. Dealing with this cloud variability and resources dimensions, *e.g.*, database size, ends up in selecting a cloud solution by adopting *ad hoc* criteria. It is all the more true when considering a multi-cloud configuration, *i.e.*, the deployment of an application over several cloud providers [14], *e.g.*, when a cloud provider does not provide the whole functionalities required by the application.

We argue that this selection can be partly automated using *Feature Models* (FMs) combined with ontology engineering [15]. Cloud variability can thus be described using FMs and cloud resources specification, defined in this paper as *dimensions*, can be done using feature attributes extending the FM [2]. In addition, we propose to use ontologies to capture cloud knowledge and deal with the range of different terms used to define the same cloud concept among cloud providers. We give an overview of the SPLE *application engineering* process by explaining how cloud features are selected and dimensions specified. Therefore, we present an approach based on the combination of ontologies with FMs to identify cloud or multi-cloud configurations whose functionalities and resources fit the application's technical requirements and their resource dimensions. The approach is implemented in a framework named SALOON, and we use as a running example the APISENSE platform, that contains up to 130 different configurations. Nonetheless, the approach proposed here is not specific to this application and can cover different domains.

The remainder of this paper is organized as follows. In SEC. 2, we discuss the motivation behind the presented approach and describe the challenges to tackle. In SEC. 3 we describe the approach and the way by which ontologies and FMs can be combined to support cloud or multi-cloud configurations. SEC. 4 presents the implementation and the first results obtained from applying the approach. Finally, SEC. 5 compares our approach with close-related work in the literature and SEC. 6 concludes the paper.

2. MOTIVATIONS & CHALLENGES

In this section, we analyze the difficulties of selecting one or several clouds from application's requirements, we discuss the reasons that lead to select a multi-cloud configuration and we identify several related challenges. We then present APISENSE, a motivating example for our approach.

2.1 Selecting Among Cloud Providers

With the cloud computing paradigm, computing resources are delivered as services. Such a model is usually described as *Anything as a Service* (XaaS or *aaS), where *anything* is divided into layers from *Infrastructure* to *Software* including *Platform*. At IaaS level, configuring the cloud environment means configuring the whole software stack running inside the VM as well as the infrastructure concerns: number of VMs, bandwidth, input/output activities, number of nodes, of hard drives, database configuration, etc. Regarding platforms provided by PaaS clouds, the configuration part focuses on software that compose this platform: which application server(s), database(s), compilation tool, libraries, etc. In a multi-cloud configuration perspective, parts of the application can be deployed either at PaaS, IaaS or both levels. The wide range of cloud providers [7] likely to host the application makes the choice difficult, and there is a lack of visibility among them to select one that matches the application technical requirements. To fit these requirements and dimensions, we find several possibilities. First, in the case depicted by FIG. 1 `a)`, the whole application is deployed on one given cloud. Second, a multi-cloud configuration can be used in case of *b)* privacy reasons, *c)* dimension reasons, in this case it's less expensive to store data on `Cloud A` or *d)* when a required element is not provided.

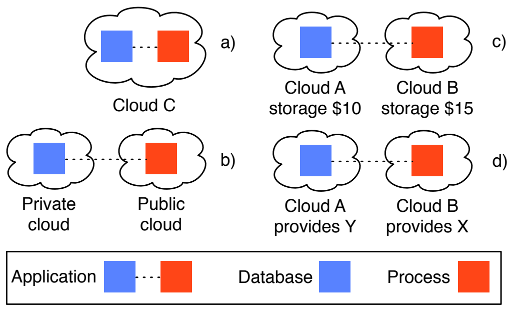

Figure 1: Cases of multi-cloud configurations

To the best of our knowledge, no approach based on configuration making tool has been proposed up to now enabling fine-grained selection of the ideal cloud or multi-cloud configuration, or at least the most suitable with the application's technical and non-functional environment. Nowadays, this choice relies on cloud computing experts' knowledge and raises issues about reliability and exhaustiveness of such a knowledge [21]. The approach described in this paper is *one* solution to help the user identifying and selecting among cloud providers.

2.2 Challenges

This contribution aims at selecting a cloud or multi-cloud configuration for the application to be deployed and faces the following challenges:

$\mathcal{C}_1$: *Support clouds variability.* The main difficulty is to deal with the wide range of clouds resources and functionalities at different layers to find clouds whose compatibility with the application's architecture and requirements is ensured.

$\mathcal{C}_2$: *Support configuration dimensions.* Among potential cloud providers identified with $\mathcal{C}_1$, a more accurate choice can be done by specifying resource dimensions related to the cloud configuration, *e.g.*, database size.

$\mathcal{C}_3$: *Support multi-cloud configurations.* Challenges $\mathcal{C}_1$ and $\mathcal{C}_2$ can be applied on multi-cloud based applications to find a valid configuration distributed over several cloud providers.

2.3 Case Study

To illustrate our proposition, we introduce APISENSE, a configurable software platform for developing crowd-sourcing applications divided into two main components: a server-side infrastructure and a mobile phone application [16]. A participant uses the mobile application to download experiments, execute them and upload the collected datasets on the server-side Java-based application, built as an assembly of services. Three of these services are mandatory. The `Collector` retrieves datasets uploaded by participants. The `Publisher` makes the experiment available to participants once defined by the scientist. Finally, the `Export` service extracts data in a computable format. Collected datasets are stored in a `Database`, either `BaseX` or `MongoDB`. The whole APISENSE FM represents up to 130 different configurations. Configuring it to be deployed in the cloud is a good way to avoid the server crashing when peak loads arise. Indeed, scientists cannot foresee how many participants are going to install their experiments on their mobile phones and send data to the server. In a multi-cloud configuration scenario, the APISENSE database and the services are deployed on different clouds. Another scenario could be the deployment of different services on several clouds (nodes).

3. PROPOSAL

To face the challenges identified in SEC. 2.2, we define a model-based approach used to *(i)* capture cloud providers' offer knowledge and *(ii)* bridge the gap between an application requirements and cloud providers available configurations.

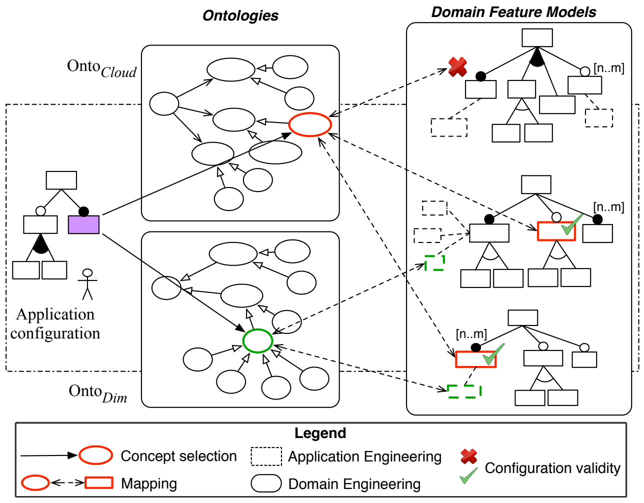

Figure 2: Approach overview

We propose in this contribution to combine ontologies and FMs into a single solution that gives strong support to all stakeholders involved in cloud deployment. The approach allows its user to *(i)* define a technical requirements configuration for the cloud providers likely to host the application considering the application configuration and *(ii)* add resource dimensions to this configuration. Ontologies and FMs form the core architecture of the approach (Fig. 2). We propose in a first step to tackle challenge $\mathcal{C}_1$ by *(i)* defining cloud providers FMs, more precisely one FM per cloud provider and *(ii)* mapping cloud ontology $Onto_{Cloud}$ concepts to cloud providers FMs' features. In a second step, we handle challenge $\mathcal{C}_2$ by proposing a dimension ontology $Onto_{Dim}$ whose concepts represent resource dimensions and are mapped to cloud providers FMs' attributes. At the end, the application configuration is mapped with each cloud provider FM, and each FM configuration validity is checked. Finally, challenge $\mathcal{C}_3$ can be tackled by combining several valid cloud configurations to fit the application's requirements.

The architecture distinguishes between two roles, the *domain experts* and the *user*. Cloud computing experts are involved in the domain description. They describe their cloud variability and commonality points, thus providing the corresponding FM to the architecture. They interact with other cloud experts to formalize the domain semantics and model the ontologies and they establish mapping between ontologies and FMs. The users are all stakeholders involved in cloud deployment, such as an application developer, a software architect or even a cloud provider, *e.g.* to test its own SaaS. Using such an approach only requires to have necessary knowledge to properly configure the application's requirements.

3.1 Cloud Systems Variability Modelling

The architecture relies on two distinct parts, FMs on one hand and ontologies on the other hand. FMs define the commonalities and variabilities of cloud providers while ontologies represents the *scope*, *i.e.*, the set of cloud providers. The definition of the commonality, the variability and the scope is part of the *Domain Engineering* process in a SPL approach [20].

3.1.1 Feature Models

FMs constitute the reasoning part of the architecture and are used to specify the functionalities provided by a given cloud. They focus on modeling and describing a cloud's commonalities and variabilities and its valid combinations. Each FM represents a cloud provider and the set of configurations related to this cloud.

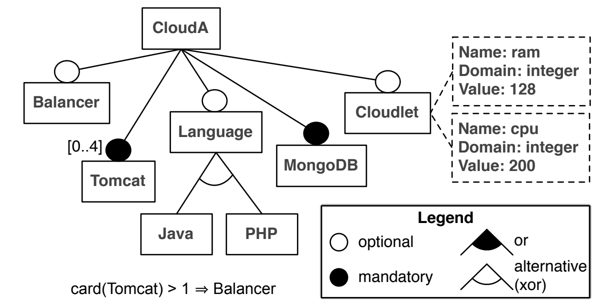

Figure 3: Excerpt of the CloudA FM (FM_{CloudA})

A *Feature Diagram* (FD) (see Fig. 3 that depicts an excerpt FM of a cloud `CloudA`) consists of a hierarchy of features (typically a tree), which may be mandatory (commonality) or optional (variability) and may form Xor or Or-groups. Constraints, *e.g.*, implies or excludes, can also be specified using propositional logic to express inter-feature dependencies. In the above example, the `Tomcat` application server is mandatory while the `Balancer` feature is optional. Configuring the `CloudA` to have more than one configured Tomcat implies such a cloud configuration to have a load `Balancer`. Such a relation is described as a constraint between features and is associated to the FD. We consider that a FM consists of a FD and the associated set of constraints.

In our approach, we defined a FM metamodel based on the concepts proposed in [20]. We extend those concepts to include more information about features. First, we extend the FM with feature cardinalities [9]. This kind of FMs is said *cardinality-based* FMs. A feature cardinality is an interval $[m..n]$ with m as lower bound and n as upper bound of this interval. This interval determines the number of instances of the feature allowed in the product configuration. For example, one possible configuration of the FM_{CloudA} allows up to 4 `Tomcat` instances. The second extension is done by adding attributes related to features, as proposed in [2]. These attributes are used to fill the lack of information in the basic FM notation. FMs with additional information are called *extended* FM*s*. As these proposals, we consider a feature attribute as a triplet <*name, domain, value*>. Thus, the `CPU` feature attribute in the FM_{CloudA} specifies the frequency provided by this feature. Constraints related to FMs can also be cardinality-based, as described by the `card(Tomcat) > 1` $\rightarrow$ `Balancer` constraint: if the number of `Tomcat` instances is upper than *one*, then a load `Balancer` must be selected. By means of constraints, we assume that the resulting configuration is fully functional.

3.1.2 Ontologies

Each cloud FM differs syntactically and semantically from each other, since defined by different experts and representing different clouds. The heterogeneity of the different underlying clouds needs to be abstracted to model the domain knowledge and define the semantics of features and their relationships. This abstract model is defined using ontologies. In computer science, ontologies were first introduced by [15] back in 1993. An ontology is *"a formal, explicit specification of a shared conceptualization"* [8]. They are used to describe the concepts and the relationships between these concepts, thus providing a vocabulary [13] and a knowledge representation of a domain, here the cloud.

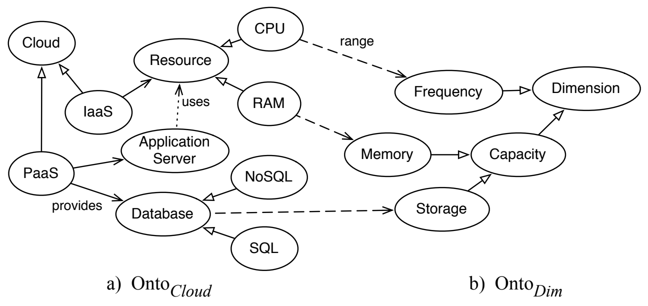

Figure 4: $Onto_{Cloud}$ and $Onto_{Dim}$ ontologies (excerpt)

We propose to use ontologies to semantically bridge the gap between the application's requirements and the architecture reasoning part, the FMs. In our approach, we define two ontologies as depicted by FIG. 4 that conforms to an ontology metamodel we defined based on the concepts proposed in [8]. The first one, $Onto_{Cloud}$, models technical requirements supported by cloud providers, *e.g.*, application server or database, and the relationships between these concepts. The second ontology, $Onto_{Dim}$, describes the dimension properties the user can specify and associate to the technical requirements selected in $Onto_{Cloud}$, *e.g.*, database size or CPU frequency.

3.1.3 Mapping Ontologies with FMs

We define two kinds of mapping to link these two ontologies with the cloud providers' FMs as depicted by FIG. 5. On the one hand, $Onto_{Cloud}$ `Concepts` are mapped to FMs `Features`. Two syntactically different `Features` in different `Feature Models` can be semantically equivalent and thus mapped with the same $Onto_{Cloud}$ `Concept`. On the other hand, regarding $Onto_{Dim}$, `Concepts` are mapped to feature `Attributes`.

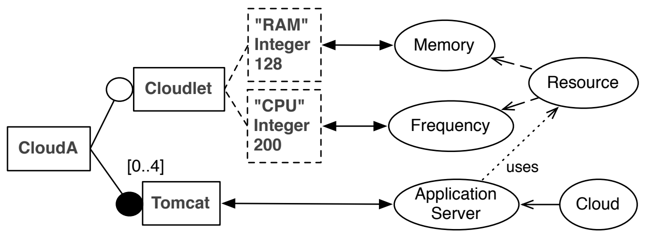

Figure 5: Example of mapping regarding FM_{CloudA}

More precisely, `Concept`'s `properties` are mapped to feature `Attributes` the following way: *(i)* `Property's domain name` (*i.e.*, `Concept's name` attribute) with `Attribute's name`, *(ii)* `Property's value` with `Attribute's value` and *(iii)* `Property's range` with `Attribute's domain`. These two mappings bridge the gap between application's requirements and the range of cloud FMs.

3.2 Clouds Configuration & Selection

Thanks to the previously described mappings, the user makes only the selection of ontologies concepts and avoids a tedious and error-prone task involving the selection of features for each cloud provider's FM. The selection of a valid combination of features identified in the domain engineering process is part of the *Application Engineering* process [20]. The user selects by hand among ontologies concepts those necessary for her/his application requirements. She/He also specifies related dimension's value when required. Let us now consider the following APISENSE configuration: $conf_1 = \{$`Database, MongoDB, Collector, Publisher, Export`$\}$. Based on the description given in SEC. 2.3, the APISENSE architect defines the set of $conf_1$ requirements as: req_1 = `[Java, MongoDB {capacity, real, 5}]`, the last number being the wished amount of GigaBytes, here given as an example. The `Storage` concept is selected and defines a property with the required information, `MongoDB` as *domain*, *real* as *range* and 5 as *value*. Once the concepts selection done, features are automatically selected in the different FMs thanks to the mapping between ontologies and FMs previously described: *(i)* $Onto_{Cloud}$ concepts are mapped with features and *(ii)* $Onto_{Dim}$ concepts are mapped with feature attributes (roughly).

The ALGO. 1 informally describes the process of feature selection for a given cloud provider FM. Running ALGO. 1

Algorithm 1 $configureFM(concepts, fm)$

```
Require: a set of $Onto_{Cloud}$ and $Onto_{Dim}$ concepts
Ensure: a $fm$ configuration $config$
 1: for all Concept $c$ in $concepts$ do
 2:   for all Feature $f$ in $fm$ do
 3:     if findMapping($c$, $f$) then
 4:       select $f$
 5:       if $c.properties \neq \emptyset$ then
 6:         for all Property $p$ in $c.properties$ do
 7:           $f.attribute \leftarrow p$
 8:         end for
 9:       end if
10:     end if
11:   end for
12: end for
```

with *concepts* related to req_1 and FM_{CloudA} as parameters results in the following FM_{CloudA} configuration: `[Database, MongoDB {capacity, real, 5}, Language, Java]`. This feature selection process is applied on each cloud FM. The validity of a cloud configuration is then checked using related FMs constraints.

3.3 Multi-Cloud Configurations

The previously described approach can be used in a multi-cloud configuration perspective and tackles challenge $\mathcal{C}_3$. Indeed, a FM configuration can be valid even if not fully covering application's requirements. In this case, it can be associated with another cloud FM valid configuration for a multi-cloud application to be deployed. Let us now consider the following `[Java, MySQL]` set of requirements req_2. $conf_2$ = `[Language, Java]` is a valid configuration for FM_{CloudA} where `CloudA` supports `Java`-based application but does not provide the `MySQL` database. $conf_2$ does not fulfill req_2. Regarding FIG. 1 case d), there could be another `CloudB` providing the `MySQL` support. Thus, the multi-cloud configuration required to deploy the application is $conf_{multi}$ = $\{$`[Language, Java]`$_{CloudA}$, `[Database, MySQL]`$_{CloudB}\}$ with `[Language, Java]` and `[Database, MySQL]` valid configurations for $_{CloudA}$ and $_{CloudB}$ respectively.

In this section, we described how our approach is used to deal with cloud variability and provide valid cloud configurations. It faces challenges $\mathcal{C}_1$ and $\mathcal{C}_2$ by defining a mapping between ontologies concepts and features and/or feature attributes. Challenge $\mathcal{C}_3$ can be tackled the same way by combining several configurations obtained from handling $\mathcal{C}_1$ and $\mathcal{C}_2$. Each FM configuration validity is then checked to determine wether the corresponding cloud(s) fits user's technical requirements and resources dimensions choices or not.

4. PRELIMINARY VALIDATION

In this section, we present the details of the SALOON framework we developed to implement the approach described in SEC. 3, as well as the results obtained from first experimentations we led to handle the challenges identified in SEC. 2.2.

4.1 Implementation

Regarding the implementation, the SALOON framework relies on *Eclipse Modeling Framework* (EMF)[1] metamodels,

[1]`http://www.eclipse.org/modeling/emf/`

which is one of the most widely accepted metamodeling technologies. Each metamodel is described as an *ecore* file and allows us to create a dynamic instance as XMI model. These models represent either an ontology, either a FM or even a mapping model. During the cloud FMs configuration process, ontology models are loaded and the user selects the required concepts. Then, mapping and FMs models are loaded. The former is used while looping on the latter ones to select the corresponding features. This choice brings high flexibility to the SALOON framework. Indeed, one can target a new cloud provider by adding corresponding FM model that conforms to the FM metamodel. Once features are selected, FMs are translated to propositional logic and constraints are loaded and checked against each cloud FM. The translation of FMs to propositional logic is well known [18] and off-the-shelf SAT solvers such as Sat4j [3] can be used to check FMs configuration validity.

4.2 Experimentation

To validate our approach, we used two different APISENSE configurations as SALOON inputs: $conf_1$, described in the previous section and $conf_2$ = `[Database, MongoDB, Collector, Publisher, Export, DataAnalysis]`, where `DataAnalysis` is a PHP-based algorithm used to compute collected datasets. Regarding these configurations, the associated sets of requirements are req_1 = `[Java, MongoDB]` and req_2 = `[Java, MongoDB, PHP]` respectively. We ran ALGO. 1 with four different PaaS FM describing Cloudbees[2], Cloud Foundry[3], dotCloud[4] and Heroku[5] PaaS providers. SALOON gave us as output the results reported in TABLE. 1.

PaaS	Req. set	Req. cover.	Missing	Multi
`Cloudbees`	req_1	✓	-	`MongoDB`
	req_2	✓	-	
`Cloud Foundry`	req_1	✓	-	
	req_2	×	`PHP`	
`dotCloud`	req_1	✓	-	
	req_2	✓	-	
`Heroku`	req_1	×	`MongoDB`	`Java`
	req_2	×	`MongoDB`	

Table 1: PaaS FM configuration using Saloon

Cloudbees and dotCloud are the only PaaS to provide both `Java` and `PHP` support. As they also provide a configured environment to deploy a `MongoDB` instance, they cover entirely the application's requirements and are able to host the two APISENSE configurations corresponding to req_1 and req_2. Cloud Foundry does not provide a `PHP` support, and cannot be configured to host APISENSE in its $conf_2$ configuration. Finally, Heroku can be configured to host `Java`-based applications, but only PostgreSQL database instances are provided. We then checked these results by deploying the two APISENSE configurations on the four real cloud PaaS named before. As described by TABLE. 1, we were able to deploy $conf_1$ on Cloudbees, Cloud Foundry, dotCloud and $conf_2$ on Cloudbees and dotCloud. Although successful, we achieved these deployments not without difficulty. Indeed, some modifications can be required before the application upload, *e.g.*, setting a correct database connection URL.

Used as a multi-cloud configurator, SALOON proposes several solutions. One possibility, described in TABLE. 1, is to configure the `MongoDB` database on the `Cloudbees` cloud, that provides fully managed Databases-as-a-Service and in particular a large `MongoDB` instance with 5GB storage, while the rest of the application is hosted on `Heroku`.

5. RELATED WORK

Multi-Cloud & Configuration. Some recent works were proposed to deal with the problem of multi-cloud and configuration. In [22], the authors propose a model-based approach that helps to model, deploy and configure complex applications in multiple IaaS. The application to be deployed is modeled and configured as an OVF appliance to be run in VMs whereas we configure the cloud(s) likely to host the application considering its requirements. In [4], a DSL is used to model the application to be deployed in the clouds while an interpreter is provided to identify which resources have to be used in the infrastructure to fulfill application's requirements. These authors pursue the same goal than us but they do not manage the cloud offer heterogeneity and can not check the validity of the obtained cloud configuration. Leusse *et al.* [11] propose in their vision paper an architecture to facilitate the deployment of different components of a same application onto different clouds. They point out the lack of visibility among cloud providers that is one of the challenges we face in this paper. In [19], the authors present a federated multi-cloud PaaS infrastructure deployed on top of several existing IaaS/PaaS. This infrastructure is based on an open service model used to design both the multi-cloud PaaS and the SaaS applications running on top of it. Contrarily to our approach, they don't need to configure the multi-cloud platform since both SaaS and PaaS are implemented using the same service model.

Ontologies, FMs & Cloud. Several related work propose an ontology or FM-based approach to discover a cloud provider. In [17, 10], ontologies are used to describe services available at IaaS and PaaS level. They both propose to reason on these ontologies to select one service that fits best user's requirements using similarity reasoning. Our approach goes in the same direction but add support to manage cloud heterogeneity and variability and check the configuration of several services thanks to FMs. Other authors such as [21, 12] present an FM-based approach to select and manage software configurations and deploy virtual appliances on IaaS providers. They consider virtual appliances as SPL products and rely on FMs to describe and select configurations. Using this approach, one can not select among several clouds and only configure IaaS VMs. Cavalcante *et al.* propose an adaptation of the SPL-based development process to deploy their Health Watcher system [6]. Regarding the application to be deployed, they include in its FM "cloud features", *e.g.*, `Amazon S3` for the storage feature, that have been collected by studying applications already deployed in the cloud. Contrarily to our approach, they modify the original application FM and, in a way, they influence the cloud provider final choice. Moreover, they do not provide any means of selecting among several clouds. In [23], the authors propose an approach based on SPL engineering to configure multi-tenant cloud applications. They rely on an ex-

[2]`http://www.cloudbees.com`
[3]`http://www.cloudfoundry.com`
[4]`https://www.dotcloud.com`
[5]`http://www.heroku.com`

tended FM to describe functionalities and quality of services for the application to be deployed and plan to use in their future work an adaptive staged configuration process for reconfiguration of FM variants. A stakeholder for each cloud level (IaaS, PaaS, SaaS) selects features for its level. Cloud provider choice is thus limited to the IaaS/PaaS selected by the corresponding stakeholder.

6. CONCLUSION

Selecting a cloud provider to host its application leads to complex choices to deal with a wide range of resources at different levels of functionality among available cloud solutions, in particular when migrating services from one cloud to another and managing distributed applications spanning multiple clouds. In this paper, we proposed a model-driven approach implemented in the SALOON framework that uses feature models to represent clouds variability, as well as ontologies to describe the heterogeneous aspect of the cloud ecosystem. A semantic mapping is defined between ontologies concepts and FMs features, that bridges the gap between application requirements and cloud providers configurations. This mapping is automated and avoids the SALOON user to select features for each cloud provider FM manually. By combining several valid cloud configurations, our approach supports multi-cloud configuration and thus faces the challenges identified in SEC. 2.2. As a preliminary validation, we used our framework on four cloud providers and showed that SALOON can be used to *(i)* check wether a cloud configuration is valid or not and *(ii)* if invalid, wether another cloud could be used in a multi-cloud configuration or not.

For future work, we plan to extend our approach to help the user in its configuration process by using previous configuration decisions as a feedback for quite equivalent application's requirements. We currently work on applying our approach as the entry point of a whole SPL, adding assets composition and product generations steps to SALOON.

Acknowledgments

This work is supported by Ministry of Higher Education and Research, Nord–Pas de Calais Regional Council, the FEDER through the *Contrat de Plan Etat Region Campus Intelligence Ambiante* (CPER CIA) 2007-2013 and the EU FP7 PaaSage project.

7. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, University of California, Berkeley, 2009.

[2] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated Reasoning on Feature Models. In *Proceedings of the 17th international conference on Advanced Information Systems Engineering*, CAiSE'05, pages 491–503, 2005.

[3] D. L. Berre and A. Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

[4] E. Brandtzæg, M. Parastoo, and S. Mosser. Towards a Domain-Specific Language to Deploy Applications in the Clouds. In *3rd International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 213–218, 2012.

[5] 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 Gener. Comput. Syst.*, 25:599–616, 2009.

[6] E. Cavalcante, A. Almeida, T. Batista, N. Cacho, F. Lopes, F. C. Delicato, T. Sena, and P. F. Pires. Exploiting Software Product Lines to Develop Cloud Computing Applications. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 179–187, 2012.

[7] CloudTimes. Cloud Computing Ecosystem. `http://cloudtimes.org/wp-content/uploads/2011/11/Clouds.cloudtimes.png`, 2012. Accessed 31.10.12.

[8] O. Corcho, M. Fernández-López, and A. Gómez-Pérez. Ontological Engineering: Principles, Methods, Tools and Languages. In *Ontologies for Software Engineering and Software Technology*, pages 1–48. 2006.

[9] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Formalizing Cardinality-based Feature Models and their Specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[10] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya. An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 104–112, 2010.

[11] P. de Leusse and K. Zielinski. Towards governance of rule and policy driven components in distributed systems. In *ServiceWave*, volume 6994 of *Lecture Notes in Computer Science*, pages 317–318, 2011.

[12] B. Dougherty, J. White, and D. C. Schmidt. Model-driven Auto-scaling of Green Cloud Computing Infrastructure. *Future Gener. Comput. Syst.*, 28(2):371–378, Feb. 2012.

[13] D. Gasevic, D. Djuric, and V. Devedzic. *Model Driven Engineering and Ontology Development (2. ed.)*. 2009.

[14] N. Grozev and R. Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, pages n/a–n/a, 2012.

[15] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[16] N. Haderer, R. Rouvoy, and L. Seinturier. A Preliminary Investigation of User Incentives to Leverage Crowdsensing Activities. In *2nd International IEEE PerCom Workshop on Hot Topics in Pervasive Computing (PerHot)*, San Diego, États-Unis, Mar. 2013. IEEE Computer Society.

[17] J. Kang and K. M. Sim. Cloudle: An Ontology Enhanced Cloud Service Search Engine. In *Proceedings of the 2010 international conference on Web information systems engineering*, WISS'10, pages 416–427, 2011.

[18] M. Mendonca, A. Wąsowski, and K. Czarnecki. SAT-based Analysis of Feature Models is Easy. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 231–240, 2009.

[19] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier. A Federated Multi-cloud PaaS Infrastructure. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 392 –399, june 2012.

[20] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. 2005.

[21] C. Quinton, R. Rouvoy, and L. Duchien. Leveraging Feature Models to Configure Virtual Appliances. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, CloudCP '12, pages 2:1–2:6, 2012.

[22] A. Sampaio and N. Mendonça. Uni4Cloud: An Approach based on Open Standards for Deployment and Management of Multi-cloud Applications. In *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*, SECLOUD '11, pages 15–21, 2011.

[23] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau. Dynamic Configuration Management of Cloud-based Applications. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 171–178, 2012.