The Evolution of CloudML and its Manifestations

Alexander Bergmayr[1], Alessandro Rossini[2], Nicolas Ferry[2], Geir Horn[3],
Leire Orue-Echevarria[4], Arnor Solberg[2], and Manuel Wimmer[1]
[1]TU Wien, Austria
{lastname}@big.tuwien.ac.at
[2]SINTEF, Norway
{firstname.lastname}@sintef.no
[3]University of Oslo, Norway
Geir.Horn@mn.uio.no
[4]TECNALIA, ICT - European Software Institute Division, Spain
Leire.Orue-Echevarria@tecnalia.com

***Abstract*—Several modelling approaches including CloudML emerged to specify the deployment of cloud-based applications and automate the provisioning of computational resources. While CloudML was introduced in the REMICS project, its development continued by ongoing projects, *i.e.*, ARTIST, MODAClouds, and PaaSage. As the evolution of CloudML in the three projects aims for a different goal, a divergence between the current project-specific manifestations of CloudML can be identified. Moreover, as the projects consider different application scenarios, CloudML has been adapted to their needs. In this paper, we distil these needs and investigate how CloudML is currently manifested in the model-based ecosystems employed by the projects. We discuss the main challenges that need to be addressed to achieve a convergence of the current CloudML manifestations.**

Keywords: Cloud Computing, Model-Driven Engineering, UML, Domain-Specific Language, Cloud Modelling Language, CloudML

I. Introduction

To address the complexity of specifying and executing cloud-based applications, several modelling approaches including CloudML[1] (Cloud Modelling Language) [1] have recently been proposed [2], [3]. The REMICS project[2] [4] introduced CloudML [5] in 2012, with the main aim to automate the provisioning and deployment of cloud-based applications. Several other projects, notably ARTIST[3] [6], MODAClouds[4] [7], and PaaSage[5] [8] are currently continuing to develop cloud modelling approaches inspired by CloudML. Because the evolution of CloudML in these projects aim for a different goal and the application scenarios considered by the projects differ, distinct cloud modelling concepts have been introduced in the project-specific manifestations of CloudML.

While the MODAClouds and PaaSage project joined forces for developing an *external* domain-specific language (DSL) [9] realised as a metamodel [1], [10], the ARTIST project developed an *internal* DSL [9] realised as a UML library along with UML profiles [11]. The main rationale behind the latter stems from the goal of the ARTIST project to support migration of existing applications to the cloud, whereby UML models are reverse-engineered and tailored to a selected cloud environment. In contrast, the goal of the MODAClouds and PaaSage projects is to support self-adaptive multi-cloud applications, whereby the models are not necessarily reverse-engineered, but rather include dynamic variability to deal with multiple cloud environments and especially run-time changes in these environments. As a result of the independent evolution of CloudML in different model-based ecosystems, syntactic as well as semantic divergences can be identified. At the same time, it seems nevertheless natural that the developed DSLs share common modelling concepts as a result of their origin even though all three projects, *i.e.*, ARTIST, MODAClouds, and PaaSage, provide model-based support for specific purposes and different application scenarios.

In this paper, we give a brief overview of CloudML by discussing its core modelling concepts and the primary needs of the three projects to adapt it to their model-based ecosystems. To demonstrate these model-based ecosystems, we introduce a representative application called SensApp [12] to which we refer throughout the remaining paper (*cf.* Section II). The project-specific ecosystems are investigated from the perspective of how CloudML has been manifested in them (*cf.* Section III). The insights gained by this investigation enables us to discuss how the DSLs complement each other and even more important how their convergence may be achieved in the long term (*cf.* Section IV). We conclude with an outlook on future work (*cf.* Section V).

II. Modelling in cloud computing

First, we briefly present the main concepts of CloudML and discuss the primary needs of the three projects to adapt it to their model-based ecosystems. Then, we introduce a representative application in the context of cloud computing.

A. CloudML *in a nutshell*

The provisioning and deployment of cloud-based applications is typically carried out based on a deployment topology

[1]CloudML: http://cloudml.org
[2]REMICS EU project: http://remics.eu
[3]ARTIST EU project: http://www.artist-project.eu
[4]MODAClouds EU project: http://www.modaclouds.eu
[5]PaaSage EU project: http://www.paasage.eu

capturing the deployable software artefacts, the middleware required to execute them, and the virtual machines providing the computational resources of a cloud environment. To address these requirements, CloudML is inspired by component-based approaches to facilitate separation of concerns. A CloudML model assembles components exposing ports (or interfaces), and bindings between these ports. CloudML supports application deployments to be specified in terms of *cloud provider-independent models* (CPIM), where the refinement into *cloud provider-specific models* (CPSM) is foreseen in a separate step. The main concepts of CloudML can be summarized as follows (we refer the reader to [1] for details):

- *Internal component*: Represents a reusable type of application component to be deployed onto an external component.
- *External component*: Represents a reusable type of a virtual machine or platform service.
- *Port*: Represents a required or provided port to a feature of a component.
- *Communication*: Represents a communication binding between ports of two components, which implies a dependency between the components.
- *Hosting*: Represents a binding between a component deployed onto another one.
- *Cloud*: Represents a collection of virtual machines offered by a particular cloud provider.

Moreover, CloudML exploits the type-instance pattern [13] to foster reuse of defined types, *e.g.*, a virtual machine type with specific characteristics. Its abstract syntax is realized in terms of a metamodel[6] based on Ecore.

In the MODAClouds and PaaSage project, dedicated tool support has been developed to enact the provisioning and deployment of multi-cloud applications and facilitate dynamic adaptations of provisioned resources at run-time, by leveraging upon the models@run-time approach [14]. For that reason, CloudML has been adapted to the needs of reasoning, simulating, and validating adaptation actions before they are carried out against components deployed onto a cloud environment.

As cloud environments are inherently elastic in the sense that provisioned resources can be scaled on-demand, these demands need to be specified for the artefacts of the deployment topology usually in terms of scalability rules. Probably the simplest form of such a rule is to let the cloud environment decide on provisioning new resources or releasing them. To specify more sophisticated custom rules for a target cloud environment requires dedicated language support as proposed by the PaaSage project.

Finally, in the ARTIST project, the selection of the target cloud environment in the context of a migration scenario has been addressed by introducing concepts that enable not only technical-related information to be captured (*e.g.*, cloud services and performance characteristics), but also business-related ones (*e.g.*, the costs of such services [15]). Particularly, the technical-related information is exploited in the refinement of deployment models towards the selected cloud environment.

[6]CloudML metamodel: https://github.com/SINTEF-9012/cloudml/tree/master/codecs/kmf/metamodel

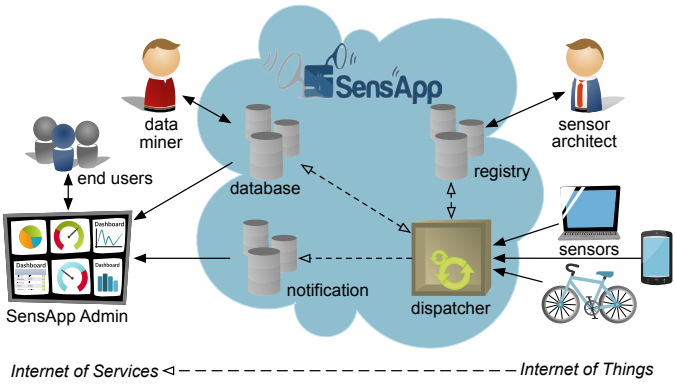

Figure 1. The SensApp architecture [12]

B. SensApp *example*

SensApp[7] is an open-source, service-oriented application for storing and exploiting large data sets collected from sensors and devices [12]. Using SensApp it is possible to register sensors, store their data, and notify clients when new data are pushed.

SensApp consists of four main components (see Figure 1). The *Registry* component stores metadata about the sensors. The *Database* component stores raw data from the sensors in a document-oriented datastore. The *Notifier* component sends notifications when relevant data are pushed. Finally, the *Dispatcher* component receives data from the sensors, stores these data in the Database according to the metadata from the Registry, and then triggers the notification mechanisms for the new data. SensApp Admin (see Figure 1) exploits the public REST API of SensApp to provide support for sensors management and data visualization using a graphical user interface. In order to be deployed, SensApp requires an application container and a database, whilst SensApp Admin requires an application container.

III. Manifestations of CloudML

To give more insights into the evolution of CloudML in the ARTIST, MODAClouds, and PaaSage projects, we briefly summarize its manifestation in the model-based ecosystems employed by these projects.

A. *ARTIST ecosystem*

A major goal of the ARTIST project is to support the migration of existing applications towards a cloud-based environment with the goal to modernise them. This typically implies adaptations to the application components. ARTIST advocates modelling techniques to provide appropriate views that are aimed at increasing the engineer's current understanding of the application components. Clearly, the selected modelling language to represent such views in terms of models plays

[7]SensApp: http://sensapp.org

an important role because the purpose of these views is to support engineers in analysing the application from which the models were produced. In the ARTIST project, UML is favoured to reverse engineer models from software artefacts not only because it is a standardised modelling language but also it provides multiple modelling viewpoints to spot potential cloud-oriented opportunities for modernizing software, platform, and infrastructure artefacts. Moreover, UML profiles facilitate models that capture environment-specific information, which appears beneficial from a reverse engineering perspective and a forward-engineering perspective [16]. Considering the latter perspective, specifying cloud-oriented deployment models directly in UML requires extensions to it, as UML's standard deployment language does not provide modelling concepts specific to cloud environments.

To address this need, in the ARTIST project, CloudML has been realised as a UML internal DSL based on lightweight extensions to the deployment viewpoint in terms of a library and profiles. They are especially beneficial for migration scenarios where reverse-engineered UML models are tailored towards the environment of the selected cloud provider in a forward engineering step. To capture environment-specific information, UML profiles dedicated to well-known cloud environments, e.g., Amazon AWS[8], Google Cloud Platform[9], and Microsoft Azure[10], have been introduced. With the notion of so called meta-profiles, cloud environment profiles can be refined with typically cross-cutting technical-related details, such as the performance of a virtual machine, and business-related information [15], like the upfront and hourly costs of a virtual machine. Exploiting UML profiles to externalize domain knowledge of cloud computing enables a clear separation between a CPIM and a CPSM, where UML profiles are applied to a CPIM for the purpose of refining it towards a CPSM as part of a forward engineering step.

An excerpt of the CPIM of the SensApp example is depicted in Figure 2. It shows SensApp's main components, their manifestation by a deployable artefact, and the deployment of the artifact in a cloud environment. Because CloudML applies the type-instance pattern, it is important to note that the deployment models depicted in Figure 2 and Figure 3 are specified at the instance level. The respective types are defined by a model library realizing CloudML. Assigning types to modelled instance specifications is natively supported by UML via the *classifier* relationship. Also, UML supports component-based modelling by default, which implies that the component concept as defined by CloudML does not have to be recreated in UML.

Considering the deployment viewpoint, Figure 3 shows the result of the refinement from the CPIM towards a CPSM, where the target 'platform' is assumed to be Amazon AWS. For that reason, the UML profile dedicated to Amazon AWS has been applied to the deployment model. While the modelled cloud node refers to the *AWSM3Medium* virtual machine type, the *DynamoDB* service is employed for the required cloud storage capabilities. Considering the defined stereotype corresponding to the *AWSM3Medium* virtual machine type, it is annotated with stereotypes provided by meta-profiles devoted to pricing and performance information of cloud services. For instance, this information can be exploited in the process of selecting the target cloud environment.

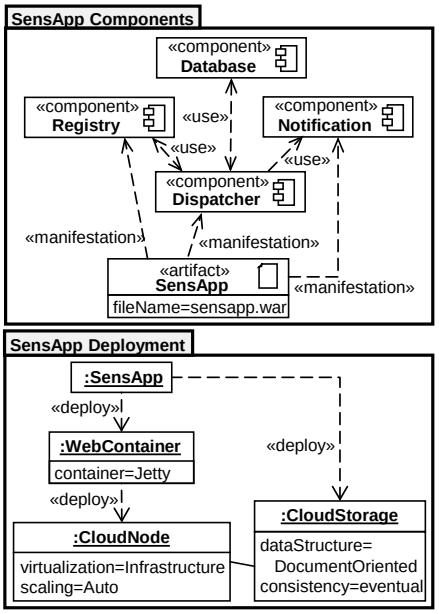

Figure 2. Excerpt of the CPIM of the SensApp example.

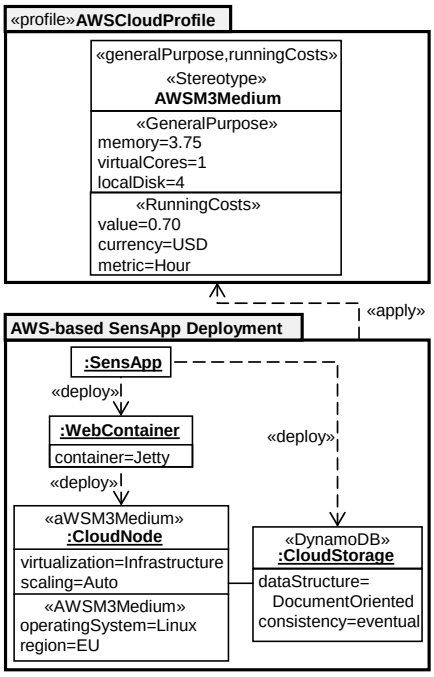

Figure 3. Excerpt of the CPSM of the SensApp example.

[8]Amazon AWS: http://aws.amazon.com
[9]Google Cloud Platform: http://cloud.google.com
[10]Microsoft Azure: http://azure.microsoft.com

B. MODAClouds ecosystem

One of the objectives of the MODAClouds project is to deliver an advanced model-driven approach along with an integrated software development environment to support engineers in building and deploying multi-cloud applications (*i.e.*, applications that run across several clouds and exploit services at the infrastructure-, platform-, and software layer), together with related data [7]. This includes defining quality of service constraints, monitoring mechanisms, prediction models, and adaptive policies to provide quality assurance. To achieve this objective, a large set of tool-supported domain-specific languages collectively called MODACloudML has been developed. MODACloudML relies on the following three layers of abstraction: *(i)* the *cloud-enabled computation independent model* (CCIM) to describe an application and its data, *(ii)* the CPIM to describe cloud concerns related to the application in a cloud-agnostic way, and *(iii)* the CPSM to describe the cloud concerns needed to deploy and provision the application on a specific cloud. Within MODACloudML, CloudML is exploited both at design-time to describe the deployment of application components on cloud resources as well as the provisioning of these resources at the CPIM and CPSM levels, and at run-time to manage the deployed applications. As a result, CloudML model encompasses runtime information such as IP addresses, cloud resources ids and statuses. As a part of MODACloudML, CloudML interacts with CCIM models describing the application to be deployed as well as models exploited for data migration and QoS optimisation and performance analysis:

- *Data Model*: describes the main data structures associated with the application to be. It can be expressed in terms of typical ER diagrams and enriched by a metamodel that specifies functional and non-functional data properties. At the CPIM level, this model refines the CCIM data model to describe it in terms of logical models. At the CPSM level, it describes the data model based on the specific data structures implemented by the cloud providers.
- *QoS Model*: includes QoS properties (*e.g.*, response time) at the application level as well as QoS properties of cloud resources in both a provider-independent (CPIM level) and a provider-specific (CPSM level) way. It includes cost information, thus offering the possibility to estimate an upper-bound for application costs.
- *Monitoring rules*: control the execution of specific software artefacts, including components and data assigned to specific resources. They are used to indicate to the run-time platform the components to be monitored.

Figure 4 depicts a CloudML CPIM model for the deployment and provisioning of the SensApp application together with associated QoS constraints specified with the MODAClouds IDE (aka. Creator 4Clouds). In this deployment model, SensApp and the MongoDB are deployed on a single virtual machine and the SensApp Admin on another one. The QoS constraints are associated to the virtual machine called `CloudNodeInstance` and specify the minimum amount of

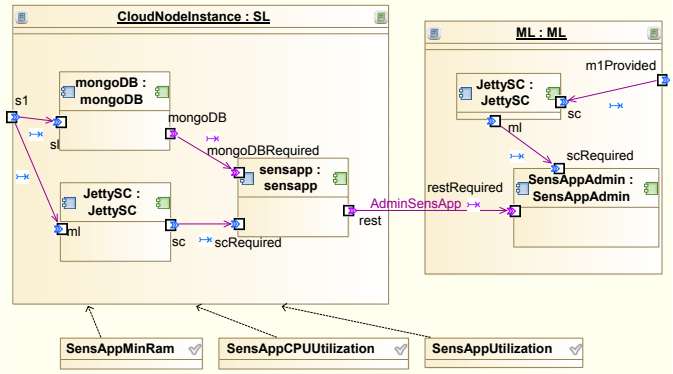

Figure 4. SensApp deployment model with QoS constraints.

RAM (e.g., the virtual machine has to offer a minimum of 2GB of RAM), the maximum CPU utilization (e.g., CPU utilization average should not exceed 90%), and the maximum average response time (e.g., it should not exceed 5 seconds) required to properly execute SensApp.

To facilitate integration and interaction with the other MODAClouds design- and run-time components, deployment models can also be transformed or specified using a JSON-based textual syntax. Listing 1 presents the specification of the SL virtual machine type that is used to host SensApp. In particular, the properties `minCores`, `minRam`, and `minStorage` represent the lower bounds of virtual compute cores, RAM, and storage, respectively, of the required virtual machine (*e.g.*, `minCores=1`, `minRam=2000`). In this case, in order to validate the QoS constraints, the `minRam` property is set to 2GB. The same set of properties can be specified graphically using the MODAClouds IDE.

Listing 1. Virtual machine type in CloudML (JSON syntax).

```
"vms": [{
  "eClass": "net.cloudml.core:VM",
  "name": "SL",
  "minRam": "2000",
  "minCores": "1",
  "minStorage": "50",
  "os": "ubuntu",
  "is64os": true,
  "securityGroup": "sensapp",
  "sshKey": "cloudml",
  "groupName": "sensapp",
  "privateKey": "cloudml.pem",
  "providedExecutionPlatforms": [{
    "eClass": "net.cloudml.core:ProvidedExecutionPlatform",
    "name": "m1Provided",
    "owner": "vms[SL]",
  }]
}]
```

C. PaaSage ecosystem

In order to cover the necessary aspects of the modelling and execution of multi-cloud applications, PaaSage adopts the Cloud Application Modelling and Execution Language (CAMEL) [17]. CAMEL integrates and extends existing DSLs, namely CloudML, Saloon [18], and the Organisation part of CERIF [19]. In addition, CAMEL integrates new DSLs developed within the project, such as the Scalability

Rule Language (SRL) [20]. CAMEL enables engineers to specify multiple aspects of multi-cloud applications, such as provisioning and deployment topology, provisioning and deployment requirements, service-level objectives, metrics, scalability rules, providers, organisations, users, roles, security controls, execution contexts, execution histories, etc. To facilitate the integration across the components managing the life cycle of multi-cloud applications, PaaSage leverages upon CAMEL models that are progressively refined throughout the *modelling*, *deployment*, and *execution* workflow phases:

- *Modelling phase*: Similar to MODAClouds, the envisioned PaaSage engineer designs a CPIM, which specifies the deployment artifacts of a multi-cloud application along with its requirements and objectives in a cloud provider-independent way.
- *Deployment phase*: The Profiler component consumes the CPIM, matches this model with the profile of cloud providers, and produces a *constraint problem*. The Reasoner component solves the constraint problem (if possible) and produces a CPSM, which specifies the deployment of a multi-cloud application along with its requirements and objectives in a cloud provider-specific way. The Adapter component consumes the CPSM and produces *deployment plans*, which specify platform-specific details of the deployment.
- *Execution phase*: The Executionware [25] consumes the deployment plans and enacts the deployment of the application components on suitable cloud infrastructures. Finally, the Executionware records historical data about the application execution, which allows the Reasoner to look at the performance of previous CPSMs when producing a new one.

At the beginning of the PaaSage project, CAMEL consisted of a family of loosely coupled DSLs, where the first member of the family was CloudML. During the course of the project, this family of loosely coupled DSLs was transformed into a single DSL. To integrate concepts from multiple DSLs into CAMEL, we adopted the Eclipse Modelling Framework (EMF). The current version of CAMEL is realized in terms of an Ecore-based metamodel organised into packages that include cross references as a result of the integration.

Listing 2 shows a CAMEL CPIM whereby the deployment model can be regarded as a variant of a CloudML model while the scalability and metric models are SRL models. The scalability model refers to the virtual machine LL (line 24) in the deployment model as well as the composite metric context `AvgExecutionTimeGT5000` (line 28) in the metric model. This scalability model specifies that the virtual machine should scale out every time the average execution time in the last five minutes is greater than 5000 milliseconds.

Listing 2. SensApp deployment, scalability, and metric models in CAMEL.

```
1  deployment model SensAppDeployment {
2    vm LL {
3      requirement set LLReqSet
4      provided host LL
5    }
6    internal component SensApp {
7      provided communication RESTProv {port: 8080}
8      required communication MongoDBReq {mandatory}
9      required host LL
10     configuration SensAppConfiguration {
11       download: "wget -P ..."
12       install: "sudo bash ..."
13     }
14   }
15 }
16 scalability model SensAppScalability {
17   scalability rule AvgExecutionTimeRule {
18     event: SensAppScalability.AvgExecutionTimeViolation
19     actions [ SensAppScalability.HorizontalScaleSensApp ]
20   }
21   horizontal scaling action HorizontalScaleSensApp {
22     type: SCALE OUT
23     vm: SensApp.LL
24     internal component: SensApp.SensApp
25     count: 1
26   }
27   non-functional event AvgExecutionTimeViolation {
28     metric condition: SensAppMetric.AvgExecutionTimeGT1000
29     violation
30   }
31 }
32 metric model SensAppMetric {
33   metric condition AvgExecutionTimeGT5000 {
34     context: SensAppMetric.AvgExecutionTimeGT5000
35     threshold: 5000.0
36     comparison operator: >
37   }
38   composite metric context AvgExecutionTimeGT5000 {
39     metric: SensAppMetric.AvgExecutionTime
40     application: SensApp
41     window: SensAppMetric.WindowFiveMin
42     schedule: SensAppMetric.ScheduleOneMin
43     composing metric contexts [ SensAppMetric.
       RawExecutionTimeContext ]
44     quantifier: ALL
45   }
46   composite metric AvgExecutionTime {
47     property: SensAppMetric.ExecutionTime
48     unit: SensAppUnit.milliseconds
49     value type: SensAppType.AvgExecutionTimeRange
50     metric formula AvgExecutionTimeFormula {
51       function arity: UNARY
52       MEAN ( SensAppMetric.RawExecutionTime )
53     }
54   }
55 }
```

IV. Convergence of CloudML manifestations

We summarise main challenges that need to be addressed for a convergence of the existing CloudML manifestations.

A. External DSL vs. UML internal DSL

As UML supports multiple viewpoints, CloudML and UML overlaps in several respects, particularly when the component viewpoint and deployment viewpoint is considered. To achieve a convergence between the CloudML manifestations requires consideration of UML concepts in addition to the concepts introduced by CloudML itself and the project-specific manifestations of it. For instance, components and ports are defined by both CloudML and UML. At the same time, the differentiation between internal and external components is specific to CloudML. This differentiation, *i.e.*, the specification of cloud services required for an application deployment, is supported via UML profiles by ARTIST's model-based ecosystem.

B. Profiles for cloud environment-specific information

The specification of cloud-specific deployment models may differ between the ecosystems of the projects. This is because

UML profiles introduce an additional typing dimension. These types need to be either introduced in terms of a properly structured type hierarchy in CloudML or translated into corresponding EMF profiles [21]. Because EMF profiles are generally applicable to Ecore-based metamodels and so to CloudML, generating EMF profiles from the UML profiles developed in the ARTIST project appears beneficial. The typing dimension supported by profiles can directly be exploited for CloudML models developed in the MODAClouds and PaaSage project to explicitly represent a CPSM produced from a CPIM. Deployment models for multi-cloud applications can be represented by applying multiple profiles that correspond to the selected cloud environments. Dynamically switching between deployment targets is enabled by (un-/re-)applying respective cloud environment profiles, which can be considered as an alternative approach to switch between ontological types that capture the respective environment-specific information. The latter approach is currently supported by the MODAClouds and PaaSage projects.

C. Possible semantic heterogeneities

The operational semantics may differ between the ecosystems of the projects: the first maps to TOSCA [22] in order to exploit OpenTOSCA [23] for enacting the provisioning and deployment of cloud-based applications, the second is based on the Cloud Modelling Framework (CloudMF) [1] and its models@run-time engine, and the third is based on the model-based Upperware [24] and Executionware [25]. As a result, a comparison of deployment models together with the provisioned cloud resources needs to be carried out before a convergence of the different CloudML manifestations can properly be achieved. The comparison can be achieved using a by-example approach on the basis of SensApp. Based on the created deployment models, respective deployment plans need to be generated. Executing these deployment plans by the employed provisioning engines enables to compare the cloud resources provisioned on a common target cloud environment.

V. Conclusion

As the evolution of CloudML resulted in divergent language manifestations, their convergence appears desirable for the purpose of resolving existing heterogeneities and enabling models to be exchanged between the model-based ecosystems employed by the projects. We aim to define correspondences between the various language concepts. This enables heterogeneities to be resolved in a non-intrusive manner. Moreover, we aim for model transformations that encode the correspondences, thereby enabling models to be exchanged at least partially. As a result, migration techniques of the ARTIST project may be applied by the MODAClouds and PaaSage model-based ecosystems to enable multi-cloud deployment models to be specified based on reverse-engineered models of existing applications. At the same time, the ARTIST model-based ecosystem may benefit from the support of deploying applications to multiple cloud environments as provided by the MODAClouds and PaaSage projects.

References

[1] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, "CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications," in *UCC*, 2014, pp. 269–277.

[2] A. Bergmayr, M. Wimmer, G. Kappel, and M. Grossniklaus, "Cloud Modeling Languages by Example," in *SOCA*, 2014, pp. 137–146.

[3] R. F. Paige, J. Cabot, M. Brambilla, L. M. Rose, and J. H. Hill, Eds., *Workshop Proceedings of CloudMDE*, vol. 1242, 2014.

[4] P. Mohagheghi, A. Berre, A. Henry, F. Barbier, and A. Sadovykh, "REMICS- REuse and Migration of Legacy Applications to Interoperable Cloud Services," in *Towards a Service-Based Internet*, 2010, pp. 195–196.

[5] E. Brandzæg, S. Mosser, and P. Mohagheghi, "Towards CloudML, a Model-Based Approach to Provision Resources in the Clouds," in *CloudMDE*, 2012, pp. 18–27.

[6] A. Bergmayr, H. Bruneliere, J. L. Cánovas Izquierdo, J. Gorroñogoitia, G. Kousiouris, D. Kyriazis, P. Langer, A. Menychtas, L. Orue-Echevarria Arrieta, C. Pezuela, and M. Wimmer, "Migrating Legacy Software to the Cloud with ARTIST," in *CSMR*, 2013, pp. 465–468.

[7] D. Ardagna, E. D. Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan, "MODAClouds: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds," in *MISE*, 2012, pp. 50–56.

[8] K. Jeffery, G. Horn, and L. Schubert, "A Vision for Better Cloud Applications," in *MultiCloud*, 2013, pp. 7–12.

[9] M. Fowler, *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[10] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *CLOUD*, 2013, pp. 887–894.

[11] A. Bergmayr, J. Troya, P. Neubauer, M. Wimmer, and G. Kappel, "UML-based Cloud Application Modeling with Libraries, Profiles and Templates," in *CloudMDE*, 2014, pp. 56–65.

[12] S. Mosser, F. Fleurey, B. Morin, F. Chauvel, A. Solberg, and I. Goutier, "SENSAPP as a Reference Platform to Support Cloud Experiments: From the Internet of Things to the Internet of Services," in *SYNASC*, 2012, pp. 400–406.

[13] C. Atkinson and T. Kühne, "Rearchitecting the UML infrastructure," *TOMACS*, vol. 12, no. 4, pp. 290–321, 2002.

[14] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg, "Models@Run.time to Support Dynamic Adaptation," *IEEE Computer*, vol. 42, no. 10, pp. 44–51, 2009.

[15] J. Cardoso, A. Barros, N. May, and U. Kylau, "Towards a Unified Service Description Language for the Internet of Services: Requirements and First Developments," in *SCC*, 2010, pp. 602–609.

[16] A. Bergmayr, M. Grossniklaus, M. Wimmer, and G. Kappel, "JUMP - From Java Annotations to UML Profiles," in *MODELS*, 2014, pp. 552–568.

[17] A. Rossini and the PaaSage consortium, "D2.1.3 – CAMEL Documentation (Final version)," PaaSage project deliverable, October 2015.

[18] C. Quinton, D. Romero, and L. Duchien, "Cardinality-based feature models with constraints: a pragmatic approach," in *SPLC*, 2013, pp. 162–166.

[19] K. Jeffery, N. Houssos, B. Jörg, and A. Asserson, "Research Information Management: The CERIF Approach," *IJMSO*, vol. 9, no. 1, pp. 5–14, 2014.

[20] K. Kritikos, J. Domaschka, and A. Rossini, "SRL: A Scalability Rule Language for Multi-Cloud Environments," in *CloudCom*, 2014, pp. 1–9.

[21] P. Langer, K. Wieland, M. Wimmer, and J. Cabot, "EMF profiles: A lightweight extension approach for EMF models," *JOT*, vol. 11, no. 1, pp. 1–29, 2012.

[22] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "TOSCA: Portable Automated Deployment and Management of Cloud Applications," in *Advanced Web Services*, 2014, pp. 527–549.

[23] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner, "OpenTOSCA - A Runtime for TOSCA-Based Cloud Applications," in *ICSOC*, 2013, pp. 692–695.

[24] C. Perez and the PaaSage consortium, "D3.1.2 – Model Based Cloud Platform Upperware," PaaSage project deliverable, October 2015.

[25] D. Baur, S. Wesner, and J. Domaschka, "Towards a Model-Based Execution-Ware for Deploying Multi-cloud Applications," in *Advances in Service-Oriented and Cloud Computing*, 2015, pp. 124–138.