2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing

# CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications

Nicolas Ferry, Hui Song, Alessandro Rossini, Franck Chauvel, Arnor Solberg
Department of Networked Systems and Services
SINTEF, Oslo, Norway
{name.surname}@sintef.no

***Abstract*—The market of cloud computing encompasses an ever-growing number of cloud providers offering a multitude of infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) solutions. The heterogeneity of these solutions hinders the proper exploitation of cloud computing since it prevents interoperability and promotes vendor lock-in, which increases the complexity of executing and managing multi-cloud applications (*i.e.*, applications that can be deployed across multiple cloud infrastructures and platforms). Providers of multi-cloud applications seek to exploit the peculiarities of each cloud solution and to combine the delivery models of IaaS and PaaS in order to optimise performance, availability, and cost. In this paper, we show how the Cloud Modelling Framework leverages upon model-driven engineering to tame this complexity by providing: (*i*) a tool-supported domain-specific language for specifying the provisioning and deployment of multi-cloud applications, and (*ii*) a models@run-time environment for enacting the provisioning, deployment, and adaptation of these applications.**

## I. Introduction

Cloud computing is a computing model enabling ubiquitous access to a shared and virtualised pool of computing capabilities (*e.g.*, processing, memory, storage, and network) that can be rapidly provisioned with minimal management effort [1]. The landscape of cloud computing encompasses an ever-growing number of providers offering a multitude of infrastructure-as-a-service (IaaS) and platform-as-a-this service (PaaS) solutions. In this landscape, application providers are facing the emergent need to execute and manage multi-cloud applications [2] (*i.e.*, applications that can be deployed across multiple cloud infrastructures and platforms). In particular, they need to exploit the peculiarities of each cloud solution as well as to combine the delivery models of IaaS and PaaS in order to optimise performance, availability, and cost. However, this is a complex task as stated in the CORDIS reports on cloud computing [3], [4], *"whilst a distributed data environment (IaaS) cannot be easily moved to any platform provider (PaaS) [...], it is also almost impossible to move a service/image/environment between providers on the same level."*

There are several projects that aim to promote interoperability and prevent vendor lock-in, but they are not sufficient to properly tame the complexity of executing and managing multi-cloud applications [5] at both design- and run-time. In particular, existing cloud solutions typically focus on supporting either IaaS or PaaS, but not both. Moreover, they adopt a design-time representation of the applications that is not kept synchronised with the underlying running systems, which hinders their continuous design and dynamic adaptation.

To address these challenges, we have extended our model-based framework called Cloud Modelling Framework (CloudMF) [6], [7]. CloudMF consists of *(i)* the Cloud Modelling Language (CloudML), a tool-supported domain-specific language (DSL) for specifying the provisioning and deployment of multi-cloud applications and *(ii)* a models@run-time environment for enacting the provisioning, deployment and adaptation of these applications.

In this paper, we present the extended version of CloudMF. The extensions consist of: *(i)* support for management of multi-cloud applications that may rely simultaneously on both IaaS and PaaS solutions (previously only IaaS was supported); *(ii)* definition of simple graphical and textual syntaxes for the language; and *(iii)* support for remote access to the models@run-time engine by third parties, including reasoning engines.

The extended version of CloudMF enables managing multi-cloud applications in a cloud provider-independent way while still exploiting the peculiarities of each IaaS and PaaS solution. By supporting both IaaS and PaaS, CloudMF enables several levels of control of multi-cloud applications: *(i)* in case of execution on IaaS or white box PaaS solutions, it offers full control with automatic provisioning and deployment of the entire cloud stack from the infrastructure to the application, or *(ii)* in case of execution on black box PaaS solutions, it offers partial control of the application (note that if parts of the multi-cloud application execute on IaaS or white box PaaS, CloudMF offers full control of those parts). In addition, by providing a model-based representation of the applications that is causally connected to the underlying running systems through the models@run-time environment, CloudMF enables the continuous design and dynamic adaptation of multi-cloud applications, where the adaptation can be automated through the usage of reasoning engines. This way, CloudMF promotes the DevOps method [8], which aims at achieving better delivery life-cycle by integrating development and operation activities. In this respect, the contribution of CloudMF is the model-based provisioning, deployment, and orchestration of multi-cloud applications.

The remainder of the paper is organised as follows. Section II introduces SensApp, a multi-cloud application used as a motivating example throughout the paper. Sections III and IV describe the DSL and the models@run-time environment, respectively. Section V reports the status of the implementation, and how it addresses the requirements identified in the motivating example. Finally, Section VI compares the

978-1-4799-7881-6/14 $31.00 © 2014 IEEE
DOI

proposed approach with the state-of-the-art and Section VII draws some conclusions.

## II. Motivating Example

SensApp[1] is an open-source, service-oriented application for storing and exploiting large data sets collected from sensors and devices [9]. It is designed to seamlessly bridge the gap between the Internet of things (IoT) and the cloud [9].

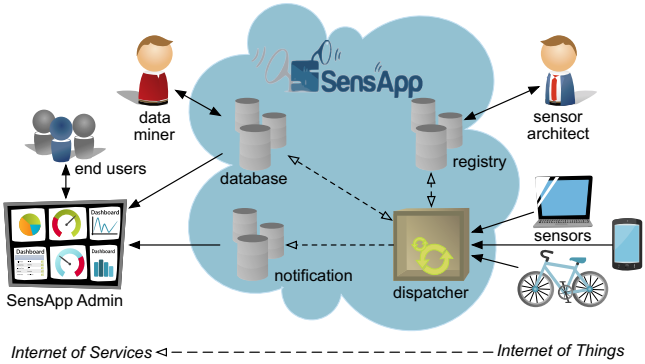


Figure 1. The SensApp architecture [9]

SensApp provides capabilities to register sensors, store their data, and notify clients when new data are pushed. It consists of three main components (see Figure 1). The *Registry* component stores metadata about the sensors (*e.g.*, description and creation date). The *Database* component stores raw data from the sensors in a MongoDB database. The *Notifier* component sends notifications to third-party applications when relevant data are pushed (*e.g.*, when new data collected by air quality sensors become available). Finally, the *Dispatcher* component orchestrates the other components: it 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) uses the public REST API of SensApp and provides capabilities to manage sensors and visualise data using a graphical user interface. In order to be deployed, SensApp requires a Servlet container and a database, whilst SensApp Admin requires a Servlet container only.

In this paper, we adopt a running example illustrating different scenarios of provisioning and deployment of SensApp and SensApp Admin. In order to reduce the management efforts, the SensApp Admin is deployed on a public Amazon Elastic Beanstalk PaaS, since it is a typical Servlet and does not require custom run-time environments. On the other hand, SensApp, which may store private data, has to be deployed in a private cloud. In this experimental scenario, the company exploiting SensApp uses its private OpenStack IaaS. Initially, as a test deployment, both SensApp and the MongoDB are deployed on the same virtual machine (VM). Eventually, as a production deployment, SensApp is migrated to a new VM whilst the database remains on the existing VM. This example motivates for the following requirements that should be addressed by CloudMF:

[1]http://sensapp.org

- **Cloud provider independence ($R_1$):** CloudMF shall support a cloud provider-agnostic specification of the provisioning and deployment. This will simplify the design of multi-cloud applications and prevent vendor lock-in.
- **Separation of concerns ($R_2$):** CloudMF shall support a modular, loosely-coupled specification of the provisioning and deployment so that the modules can be seamlessly substituted. This will facilitate the maintenance as well as the dynamic adaptation of the deployment topology.
- **Reusability ($R_3$):** CloudMF shall support the specification of types or patterns that can be seamlessly reused to design the system. This will ease the evolution as well as the rapid development of different variants of a system in time and in space.
- **Abstraction ($R_4$):** CloudMF shall provide a single domain-specific language and abstraction for specifying deployment on both IaaS and PaaS in a cloud provider-independent or -specific way. In addition, CloudMF shall provide a continuously up-to-date, abstract representation of the running system. This will facilitate the reasoning, simulation, and validation of the adaptation actions before their actual enactments.
- **White- and black-box infrastructure ($R_5$):** CloudMF shall support IaaS and PaaS solutions. This will enable various degrees of control over underlying infrastructures and platforms of multi-cloud applications.

Cloud provider independence ($R_1$) is justified by the need of deploying SensApp and SensApp Admin on multiple clouds as well as by the need of migrating SensApp from one cloud to another. Furthermore, separation of concerns ($R_2$) and reusability ($R_3$) are justified by the need of migrating SensApp *(i)* without modifying the specification of the components that are not concerned with the migration, and *(ii)* without redefining the deployment management of the components involved in the migration. Moreover, abstraction ($R_4$) is justified by the need of monitoring the status of SensApp and SensApp Admin as well as planning their adaptation. Finally, support for white- and black-box infrastructure ($R_5$) is justified by the need to deploy SensApp on a IaaS in order to enable its full control and to deploy SensApp Admin on a PaaS in order to reduce its management efforts.

In the next two sections, we present the two main components of CloudMF, *i.e.*, CloudML and the models@run-time environment, and how they address these requirements.

## III. The CloudML Modelling Language

CloudML relies on model-driven techniques and methods. Model-driven engineering [10] is a branch of software engineering which aims at improving the productivity, quality, and cost-effectiveness of software development by shifting the paradigm from code-centric to model-centric. This approach, which is commonly summarised as "model once, generate anywhere", is particularly relevant to tame the complexity of developing complex systems such as multi-cloud systems [6], [7]. Models and modelling languages as the main artefacts

of the development process enable developers as well as reasoning engines to work at a high level of abstraction by focusing on cloud concerns rather than implementation details. Model transformation serves as the primary technique to generate (parts of) software systems restrains developers from repetitive and error-prone tasks.

CLOUDML is inspired by the OMG's Model-Driven Architecture [11] and allows developers to model the provisioning and deployment of a multi-cloud application at two levels of abstraction: *(i)* the Cloud Provider-Independent Model (CPIM), which specifies the provisioning and deployment of a multi-cloud application in a cloud provider-agnostic way (addressing the requirement $R_1$); *(ii)* the Cloud Provider-Specific Model (CPSM), which refines the CPIM and specifies the provisioning and deployment of a multi-cloud application in a cloud provider-specific way. This two-level approach is agnostic to any development paradigm and technology, meaning that the application developers can design and implement their applications based on their preferred paradigms and technologies.

CLOUDML is also inspired by component-based approaches [12], which facilitate separation of concerns ($R_2$) and reusability ($R_3$). In this respect, deployment models can be regarded as assemblies of components exposing ports (or interfaces), and bindings between these ports.

In addition, CLOUDML implements the *type-instance* pattern [13], which also facilitates reusability ($R_3$) and abstraction ($R_4$). This pattern exploits two flavours of typing, namely *ontological* and *linguistic*, respectively [14]. Figure 2 illustrates these two flavours of typing. `SL` (for Small Linux) represents a reusable type of VM. It is linguistically typed by the class `VM` (for Virtual Machine). `SL1` represents an instance of the VM `SL`. It is ontologically typed by `SL` and linguistically typed by `VMInstance`.

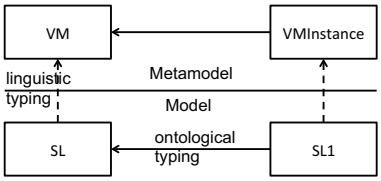


Figure 2. Linguistic and ontological typing

Previously, the CLOUDML editor supported two formats for the textual syntax, namely the JavaScript Object Notation (JSON) and the XML Metadata Interchange (XMI) which resulted from the serialization of deployment models. We have extended CLOUDMF with: *(i)* a web-based editor that supports a new graphical syntax (see Figure 4) and *(ii)* an Eclipse-based editor (see Figure 3) that supports a new textual syntax and offers features such as syntax highlighting, auto-completion, and on-site validation. The models defined using this syntax can be serialized into the JSON and XMI formats. Figure 4 illustrates the deployment model of our example using this graphical syntax.

### A. Cloud Provider-Independent Model

In the following, we provide a description of the most important classes and corresponding properties in the CLOUDML

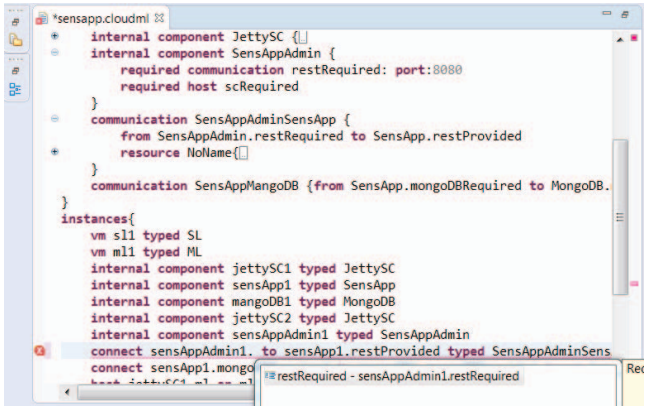


Figure 3. Screenshot of the text-based CloudML Editor

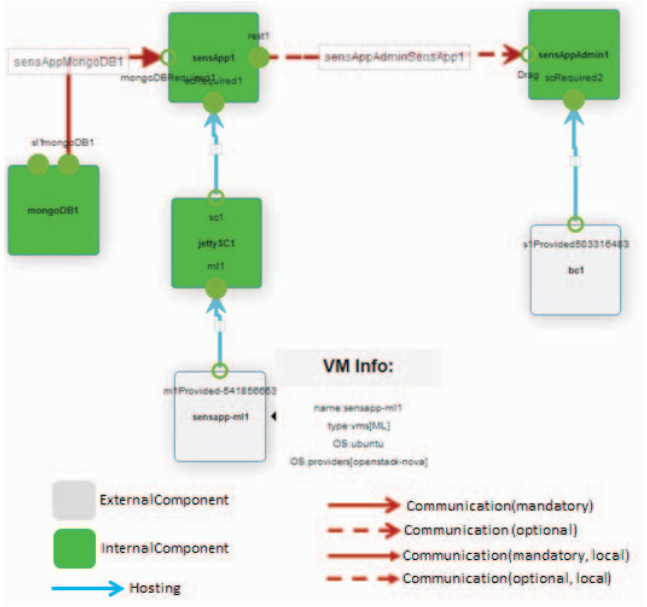


Figure 4. SENSAPP deployment model

metamodel as well as sample models in the associated textual syntax. The textual syntax better illustrates the various concepts and properties that can be involved into a deployment model and that can be hidden in the graphical syntax. Both the textual and the graphical syntax offer an abstraction over multi-cloud and cloud provider-specific concepts as well as over the deployment process.

Figure 5 shows the type portion of the CLOUDML metamodel in Ecore format[2].

A `CloudMLModel` consists of `CloudMLElements`, which can be associated with `Property` and `Resources`. A `Resource` represents an artefact (*e.g.*, scripts, binaries, configuration files, etc.) adopted to manage the deployment lifecycle (*e.g.*, download, configure, install, start, and stop). The three main types of `CloudMLElements` are `Component`, `Communication`, and `Hosting`.

[2]http://www.eclipse.org/modeling/emf/

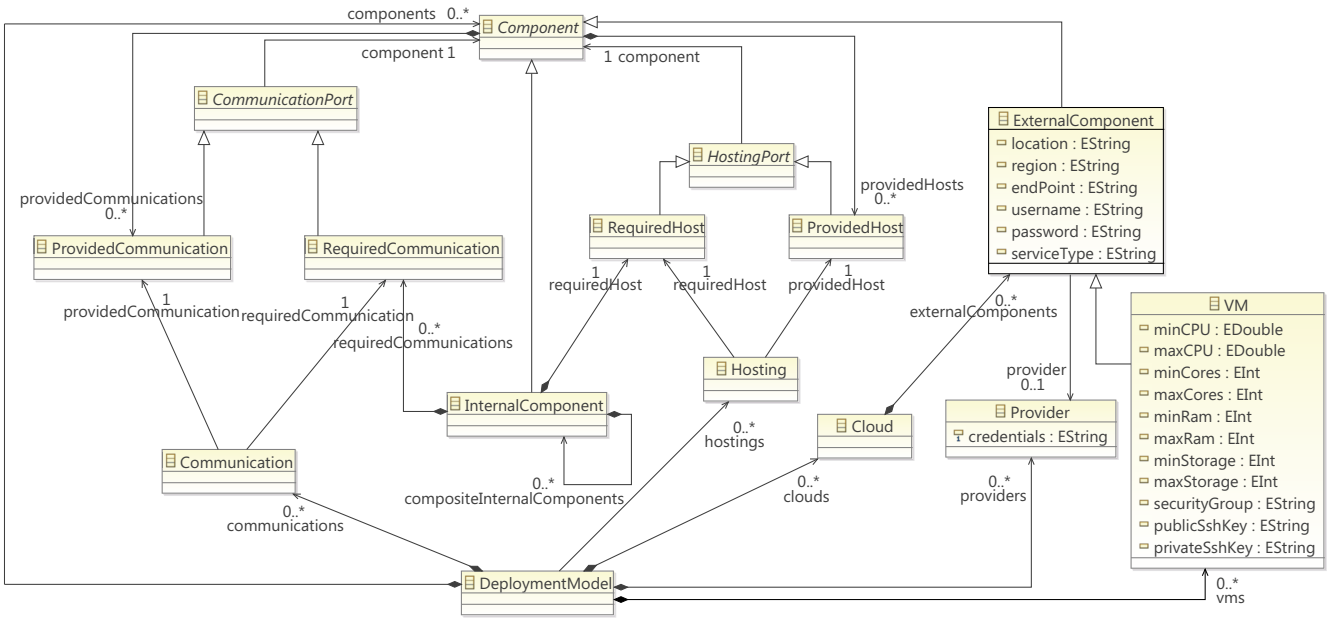


Figure 5. Type part of the CLOUDML metamodel

A `Component` represents a reusable type of component of a cloud-based application. A `Component` can be an `ExternalComponent`, meaning that it is managed by an external `Provider` (*e.g.*, an Amazon Beanstalk container, see Listing 1), or an `InternalComponent`, meaning that it is managed by CLOUDMF (*e.g.*, a Servlet container or SENSAPP). This mechanism enables supporting both IaaS and PaaS solutions through the single abstract concept of component ($R_5$). The property `location` of `ExternalComponent` represents the geographical location of the data centre hosting it (*e.g.*, `location="eu-west-1"`, short for West Europe). The properties `username`, `password`, and `serviceType` represent the authentication information needed to access to this specific service and its type (*e.g.*, database, application container), respectively.

Listing 1. An example of an External Component type from a CPIM

```
external component BeanstalkContainer {
  provider: Beanstalk, location: "eu-west-1"
  environment: "Tomcat", autoscaling: false
  provided host SCProvided {("language": "Java")}
}
```

An `ExternalComponent` can be a `VM` (*e.g.*, a VM running GNU/Linux, see Listing 2). The properties `minCores`, `maxCores`, `minRam`, `maxRam`, `minStorage`, and `maxStorage` represent the lower and upper bounds of virtual compute cores, RAM, and storage, respectively, of the required VM (*e.g.*, `minCores=1`, `minRam=1024`). The property `OS` represents the operating system to be run by the VM (*e.g.*, `OS="ubuntu"`). All these constraints are optional and do not have to be defined in the CPIM.

Listing 2. An example of a VM type from a CPIM

```
vm SL {
  provider: OpenStackNova, os: "ubuntu", os64
  ram: 1024.., cores: 1.., storage: 50..
  securityGroup: "SensApp", sshkey: "cloudml", groupName: "
     sensapp"
  provided host slProvided
}
```

`Components` are connected through two kinds of ports. A `CommunicationPort` represents a communication interface of a component. A `CommunicationPort` can be a `ProvidedCommunication`, meaning that it provides a feature to another component (*e.g.*, SENSAPP provides a REST interface, see Listing 3), or a `RequiredCommunication`, meaning that it consumes a feature from another component (*e.g.*, SENSAPP requires a MongoDB interface, see Listing 3). Only internal components can have a `RequiredCommunication` since they are managed by CLOUDMF. The property `isLocal` shows whether the component providing the feature and the component consuming the feature have to be deployed on the same external component (*e.g.*, in the initial deployment SENSAPP and MongoDB have to be deployed on the same VM, see Listing 3). The property `isMandatory` of `RequiredCommunication` represents that the `InternalComponent` depends on this feature (*e.g.*, SENSAPP depends on MongoDB and hence MongoDB has to be deployed before SENSAPP, see Listing 3).

A `HostingPort` represents a hosting interface of a component. A `HostingPort` can be a `ProvidedHost`, meaning that it provides hosting facilities (*i.e.*, it provides an execution environment) to another component (*e.g.*, a VM running GNU/Linux provides hosting to a Servlet container, see Listing 2), or a `RequiredHost`, meaning that an internal component requires hosting from another component (*e.g.*, SENSAPP requires hosting from a Servlet container, see Listing 3).

Listing 3. An example of an Internal Component type from a CPIM

```
internal component SensApp {
  resource SensAppResource {
    download: "wget -P ~ http://github.com/downloads/SINTEF
     -9012/sensapp/sensapp.war; wget -P ~ http://cloudml.org
     /scripts/linux/ubuntu/sensapp/sensapp.sh",
    install: "sudo bash sensapp.sh"
  }
  provided communication restProvided {local, port: 80}
  required communication mongoDBRequired {local, port: 0,
     mandatory}
  required host SCRequired { ("language" : "Java") }
}
```

A `Communication` represents a reusable type of communication binding between a `Required-` and a `ProvidedCommunication` (*e.g.*, SENSAPP communicates with SENSAPP ADMIN through HTTP on port 80, see Listing 4). A `Communication` can be associated with `Resources` specifying how to configure the components so that they can communicate with each other.

Listing 4. An example of a Communication type from a CPIM

```
communication SensAppAdmin2SensApp {
  from SensAppAdmin.restRequired
  to SensApp.restProvided,
  resource SensappAdmin2SensAppResource: {
    download: "wget -P ~ http://cloudml.org/scripts/linux/
     ubuntu/sensappAdmin/install_sensappadmin.sh",
    configure : "sudo bash install_sensappadmin.sh"
  }
}
```

A `Hosting` represents a reusable type of hosting binding between `Required-` and a `ProvidedHost` (*e.g.*, a Servlet container is contained by a VM running GNU/Linux, see Listing 5). A `Hosting` can be associated with `Resources` specifying how to configure the components so that the contained component can be deployed on the container component.

Listing 5. An example of an Hosting type from a CPIM

```
execution JettySC2SL {
  from JettySC.slRequired to SL.slProvided
}
```

These types can be instantiated in order to form an assembly of components that specifies a deployment model. Each instance is identified by a unique identifier and refers to a type (see Listing 6).The deployment model of our SENSAPP example can then be specified as depicted in Figure 4.

Listing 6. Example of instances from a CPIM

```
instances {
  external component bc1 typed BeanstalkContainer
  internal component sensApp1 typed SensApp
  internal component sensAppAdmin1 typed SensAppAdmin
  internal component jettySC1 typed JettySC
  vm sl1 typed SL
  connect sensAppAdmin1.restRequired to sensApp1.
     restProvided typed SensAppAdmin2SensApp
  host jetty1.slRequired on sl1.slProvided typed JettySC2SL
}
```

### B. From CPIM to CPSM

In the following, we show how a CPIM can be refined into a CPSM. A deployment model at the CPSM level consists of an enrichment of the instances of the corresponding CPIM with cloud provider-specific information. This enrichment mainly affects external components.

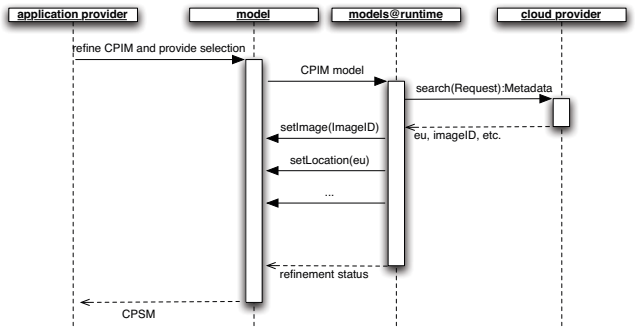


Figure 6. Example of refinement of a CPIM into a CPSM

The transformation from CPIM to CPSM consists in: *(i)* adding the actual data resulting from the resolution of the constraints defined in the external component types (*e.g.*, actual number of cores, RAM size, storage size), and *(ii)* adding data required for the deployment and management of the application that are cloud provider-specific. Thanks to this enrichment, it is possible to retrieve data about the actual resources provisioned including how they can be accessed and how they can be configured. Such data is particularly useful during the process of configuration of the components and their bindings. Rather than relying on models specifying all possible cloud providers and corresponding cloud services supported by CLOUDMF, which should be maintained up-to-date with the current offerings, the refinement of a CPIM into CPSM retrieves cloud provider-specific information directly from the cloud providers. In order to achieve this refinement, the models@run-time environment interacts with the APIs of the cloud providers.

Figure 6 presents an example of a refinement of a CPIM into a CPSM in the context of the motivating example. The application provider specifies the cloud provider on which the application will be provisioned and deployed (*e.g.*, the `sensApp1` Servlet, `jettySC1` Servlet container, and the `sl1` VM running GNU/Linux will be provisioned and deployed in the private OpenStack IaaS). The models@run-time engine requests the cloud providers for a list of available VMs compatible with the constraints defined in the VM type (*e.g.*, the list of VMs with at least 1 core, at least 1024 MiB of RAM, and at least 50 GiB of storage available on the private OpenStack IaaS). The cloud provider responds the models@run-time engine with this list along with metadata associated with each VM (*e.g.*, the `small` VM instance located in the `EU`). Similar metadata can be requested for PaaS (*e.g.*, the public Amazon Elastic Beanstalk PaaS supports Java and autoscaling). Finally, the models@run-time engine uses this metadata to refine the CPIM into a CPSM, and enacts the actual provisioning and deployment of this CPSM.

In the following, we describes how the models@run-time environment exploits CPSMs to provision, deploy, and adapt multi-cloud application.

## IV. THE CLOUDMF MODELS@RUN-TIME ENVIRONMENT

Models@run-time [15], [16] is an architectural pattern for dynamically adaptive systems that leverages models as executable artefacts supporting the execution of the system.

In particular, models@run-time provides an abstract representation of the underlying running system, which facilitates reasoning, simulation, and enactment of adaptation actions. A change in the running system is automatically reflected in the model of the current system. Similarly, a modification to this model is enacted on the running system on demand. This causal connection enables the continuous evolution of the system with no strict boundaries between design-time and run-time activities.

We believe that the models@run-time approach is particularly relevant in the context of multi-cloud applications, since the time overhead introduced by the models@run-time engine is negligible compared to the time needed to enact an adaptation action in the cloud (*e.g.*, provisioning of VMs may take several minutes). This is in contrast with other application areas where the time overhead introduced by models@run-time can be challenging to cope with.

### A. Overview

Within CLOUDMF, the models@run-time environment provides a CPSM causally connected to the running system (addressing $R_4$). On the one hand, any modification to the CPIM will be reflected in the CPSM and, in turn, automatically propagated onto the running system. On the other hand, any change in the running system will be reflected in the CPSM, which, in turn, can be assessed with respect to the CPIM.

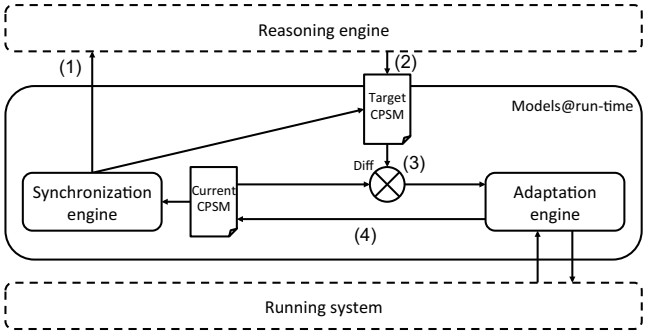


Figure 7. The CLOUDMF models@run-time architecture

Figure 7 depicts the architecture of the CLOUDMF models@run-time engine. The reasoning system reads the current CPSM (step 1), which describes the running system, and produces a target CPSM (step 2). Then, the run-time environment computes the difference between the current CPSM and the target CPSM (step 3). Finally, the adaptation engine enacts the adaptation by modifying only the parts of the system necessary to account for the difference and the target CPSM becomes the current CPSM (step 4).

The current CPSM can be manipulated in both imperative and declarative ways, *i.e.*, it can be modified through a set of instructions, or a new CPSM can be provided to replace the existing one. Both approaches result in a target CPSM that is consumed by the comparison engine.

### B. Comparison engine

The inputs to the comparison engine (also called *diff*) are the current and target CPSMs. The output is a list of actions representing the required changes to transform the current CPSM into the target CPSM. The types of potential actions are listed in Table I and result in: *(i)* modification of the provisioning and deployment topology, *(ii)* modifications of the components' properties, or *(iii)* modifications of their status on the basis of their life-cycle. In particular, the status of an external component can be `running`, `stopped` or in `error`, whilst the status of an internal component can be `uninstalled`, `installed`, `configured`, `running` or in `error`.

The comparison engine processes, in order, external components, internal components, hostings, and communications, on the basis of the logical dependencies between these concepts. In this way, all the components required by another component are deployed first.

For each of these concepts, we compare the two sets of instances from the current and target CPSMs. This comparison is achieved on the matching the properties of both instances and corresponding types. For each unmatched instance from the current CPSM, a `remove` action with the instance as argument is created. Similarly, for each unmatched instance from the target CPSM, an `add` action with the instance as argument is generated.

In the context of the motivating example, the migration of SENSAPP on a new VM consists in providing as input the target CPSM depicted in Figure 8. This will result in the following actions: add new instances of the VM type `SL`, the hosting type `JettySC2SL`, the hosting type `MongoDB2SL` and remove the existing instance of the hosting type `JettySC2SL`. Because of the abstraction offered by CLOUDML, this adaptation is achieved without the need for new types and resources or for redefining the existing ones. In addition, thanks to the comparison process, the MongoDB remains available during the adaptation process.

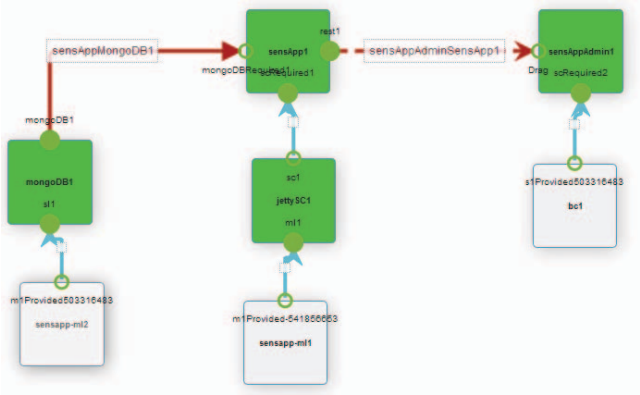


Figure 8. SENSAPP CPSM after adaptation

Please note that, at this stage, the target CPSM has priority over the current CPSM, which for example means that any VM instance in the target CPSM that does exist in the current CPSM will be regarded as one that needs to be created. Conversely, any VM in the current CPSM that does not exist in the target CPSM will be removed. This approach is effective when there is no change in the running system during reasoning. Coping with changes that occur during

Table I. TYPES OF OUTPUT ACTIONS GENERATED BY THE COMPARISON ENGINE

| Action | Parameter | Effect |
|---|---|---|
| addExternalComponent | ExternalComponent | provision a new VM or start a PaaS service |
| removeExternalComponent | ExternalComponent | terminate a VM or stop a PaaS service |
| addInternalComponent | InternalComponent | deploy the internal component on a host |
| removeInternalComponent | InternalComponent | remove the internal component instance from its current host |
| addCommunication | Communication | configure the endpoints of the communication |
| removeCommunication | Communication | unconfigure the endpoints of the communication |
| addHosting | Hosting | configure the endpoints of the hosting |
| removeHosting | Hosting | unconfigure the endpoints of the hosting |
| setStatus | Status | change the status of a component |
| setProperty | Property | change a property of a component |

reasoning could be handled in various ways, for instance as part of a third step of the adaptation process (model checking). Currently, CLOUDMF does not handle changes that occur during reasoning.

While comparing the current and target CPSM, the comparison engine deduces an ordering of deployment actions, which accounts for the dependencies captured by the model (provisioning thus precedes configurations of communications). However, ad hoc synchronisation issues between asynchronous configuration actions (*e.g.*, database migration) must be handled in the configuration management scripts that perform these actions.

### C. Synchronisation between the models@run-time engine and third parties

The models@run-time environment also provides synchronisation mechanisms for remote third parties (*e.g.*, reasoning engines) to adapt the system. The need for mechanisms to facilitate the synchronisation of the models@run-time environment with third parties has emerged from several use cases that apply self-adaptive mechanisms on top of CLOUDMF. Such mechanisms typically involve decision making engines, which reason on their own views of the model of the running system. Because these views have to remain synchronised with the running system, appropriate transformations must be automatically triggered at run-time.

As an example, a use case from the DIVERSIFY EU project[3] consists in building one or several bio-inspired engines to automatically repair cloud-based applications [17]. One of the reasoning engines involved in this use case is a diversity controller which reasons on a population model (*i.e.*, representing species and the number of individuals in each species). In order to ensure that the controller reasons on an up-to-date population model, the latter has to be synchronised with the CPSM.

The model synchronisation is implemented by the propagation of changes in both directions, namely *notification* and *command*. A notification allows the models@run-time engine to propagate its change to third parties, whilst a command enables modifications on the current CPSM. Because two models used by two parties can be isolated from each other and may not be aware of the whole model state, only the sequence of modifications is propagated, without carrying the start state of each change. Therefore, both notification and command are a sequence of modifications.

The communication with third parties is achieved using the WebSocket protocol[4] in order to enable light-weight communications. Events are encoded as plain text, and can be defined using a domain-specific language. This includes the text formatting, the query and criteria to locate the relevant model element, the modification or change on the element, and the combination of other events. We have defined the standard MOF-reflection modifications as the primitive events, and allow developers to define further high-level events as the composition of primitive ones. Using this language, it is also possible to define the changes on an abstract model as the composition of events on a concrete model, and thus implement event-based transformation. After each adaptation, the engine wraps the modification events into one message and sends it to the WebSocket port.

In order to handle concurrency (*i.e.*, adaptation actions coming from several third parties) the models@run-time environment uses a simple transaction-based mechanism. The WebSocket component creates a single transaction which contains all the modifications from a third party, and passes it to a concurrency handler. The handler queues the transactions and executes them one after another without overlapping. Since all the modifications are simply assignment or object instantiation commands on the model in the form of Java objects, the time to finish a transaction of events is significantly shorter than the adaptation process.

## V. SYNTHESIS

In the following, we discuss how our approach addresses the requirements defined in Section II and we report the status of the reference implementation of CLOUDMF.

### A. Requirements

The following list summarises how CLOUDMF fulfils the requirements presented in Section II.

- **Cloud provider-independence ($R_1$):** The layering of the modelling stack into CPIMs and CPSMs ensures that the provisioning and deployment templates are cloud provider-independent.
- **Separation of concerns ($R_2$):** The component-based design of the CLOUDML meta-model ensures that the provisioning and deployment templates and models are modular and loosely-coupled. Furthermore, the CPIM and CPSM abstraction levels effectively ensures

[3] http://www.diversify-project.eu

[4] http://www.websocket.org/

the separation of concerns between cloud provider-specific from the cloud-provider general provisioning and deployment design.

- **Reusability ($R_3$):** The type-instance pattern in the CLOUDML metamodel ensures that types a can be reused across several models.
- **Abstraction ($R_4$):** Thanks to CLOUDML, our framework offers a single domain-specific language and abstraction which enables the management of application on both IaaS and PaaS solutions. Independently of their delivery mode, these solutions can be represented in an homogeneous way as components. In addition, the models@run-time environment provides an abstract and up-to-date representation of the running system which can be dynamically manipulated, and the CPIM provides abstraction over cloud provider-specific concerns.
- **White- and Black-box infrastructure ($R_5$):** CLOUDMF includes concepts and mechanisms that enable the support for both IaaS and PaaS solutions, enabling management of components where CLOUDMF has full control of their underlying infrastucture and platforms (IaaS/white-box) and exploitation of advanced and rigid PaaS's that feature little control from the outside (black-box).

### B. Reference implementation

CLOUDMF has been applied in a set of use cases in various research projects, in particular in the FP7 EU projects REMICS [5], MODAClouds[6], PaaSage[7], and DIVERSIFY[8]. The set of use cases includes the migration of legacy applications to the cloud as well as the migration of applications from one cloud to another. In addition, all these projects apply self-adaptive mechanisms on top of CLOUDMF in order to optimise the execution of multi-cloud applications with maximum quality of service at minimum cost.

CLOUDMF is available as an open-source project[9] implemented in Java using Maven as the build tool. The current codebase consists of around 24 000 lines of code. The CLOUDML models and metamodels are represented as plain Java objects. These models can be serialised in either JSON or XMI. The JSON and XMI codecs are based on Kotlin[10] and the Kevoree Modeling Framework (KMF) [18]. The textual syntax editor[11] is based on the Xtext framework [19]. For the IaaS level management, the provisioning and deployment engine relies on jclouds[12] and the Flexiant Cloud Orchestrator API[13]. For the PaaS management, the engine uses the Cloud4SOA [20] library and the Amazon Elastic Beanstalk[14] and RDS [15] APIs.

[5] http://www.remics.eu
[6] http://www.modaclouds.eu
[7] http://www.paasage.eu
[8] http://www.diversify-project.eu
[9] https://github.com/SINTEF-9012/cloudml
[10] http://kotlin.jetbrains.org/
[11] https://github.com/SINTEF-9012/cloudml-dsl
[12] http://jclouds.incubator.apache.org/
[13] http://docs.flexiant.com
[14] http://aws.amazon.com/fr/elasticbeanstalk/
[15] http://aws.amazon.com/fr/rds/

## VI. RELATED WORK

This research is the continuation of our model-based framework called Cloud Modelling Framework (CLOUDMF) [6], [7]. In this paper, we present the extended version of CLOUDMF which adds *(i)* support for management of multi-cloud applications that may rely simultaneously on both IaaS and PaaS solutions (previously only IaaS was supported); *(ii)* definition of simple graphical and textual syntaxes for the language; and *(iii)* support for remote access to the models@runtime engine by third parties, including reasoning engines.

In the cloud community, libraries such as jclouds[16] or DeltaCloud[17] provide generic APIs abstracting over the heterogeneous APIs of IaaS providers, thus reducing cost and effort of deploying multi-cloud applications. While these libraries effectively foster the deployment of cloud-based applications across multiple cloud infrastructures, they remain code-level solutions, which make design changes difficult and error-prone. More advanced frameworks such as Cloudify[18], Puppet[19], or Chef[20] provide DSLs that facilitate the specification and enactment of provisioning, deployment, monitoring, and adaptation of cloud-based applications, without being language-dependent. As for the research community, the mOSAIC [21] project tackles the vendor lock-in problem by providing an API for provisioning and deployment of multi-cloud applications. This solution is also limited to the code level. The Topology and Orchestration Specification for Cloud Applications (TOSCA) [22] is a specification developed by the OASIS consortium, which provides a language for specifying the components comprising the topology of cloud-based applications along with the processes for their orchestration. By contrast with CLOUDMF, the aforementioned approaches only focus on the management of cloud-based applications deployed on IaaS environments, and are conceived for design-time only.

The literature encompasses several approaches to the management of cloud-based applications deployed on PaaS environments. Sellami *et al.* [23] propose a model-driven approach to PaaS-independent provisioning and management of cloud-based applications. This approach includes a language for modelling provisioning and deployment, as well as a REST API for enacting them. The Cloud4SOA EU project [20] provides a framework for facilitating the matchmaking, management, monitoring, and migration of cloud-based applications on PaaS environments. By contrast with CLOUDMF, these approaches focus on one cloud delivery model only (*i.e.*, either IaaS or PaaS, but not both). In addition, their models are not causally connected to the running system, and may become irrelevant as soon as the running system is changed. The approaches proposed in the CloudScale [24] and Reservoir [25] projects suffer similar limitations.

The work of Shao *et al.* [26] was a first attempt to build a models@run-time platform for the cloud, but remains restricted to monitoring, without providing support for enactment of provisioning and deployment. To the best of our knowledge, CLOUDMF is thus the first attempt to reconcile

[16] http://www.jclouds.org
[17] http://deltacloud.apache.org/
[18] http://www.cloudifysource.org/
[19] https://puppetlabs.com/
[20] http://www.opscode.com/chef/

cloud management solutions with modelling practices through the use of models@run-time.

## VII. Conclusion

In this paper, we presented how the Cloud Modelling Framework (CloudMF) leverages upon model-driven techniques and methods to enable the specifications of the provisioning and deployment of multi-cloud applications at design-time and their enactment at run-time. The Cloud Modelling Language (CloudML) supports the cloud provider-independent specification of multi-clouds application including IaaS and PaaS solutions, which are both modelled in an endogenous way through the concept of component. The associated models@run-time environment provides mechanisms for the dynamic provisioning, deployment, and adaptation of multi-cloud applications to third parties through a well-defined interface.

## Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number: 318484 (MODAClouds), 317715 (PaaSage). In addition, we would like to thank Nikolay Nikolov for his help with and comments on the CloudML metamodel.

## References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Special Publication 800-145, September 2001.

[2] D. Petcu, "Consuming Resources and Services from Multiple Clouds," *Journal of Grid Computing*, pp. 1–25, 2014.

[3] SSAI Expert Group, "The Future of Cloud Computing," Tech. Rep., 2010. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[4] ——, "A Roadmap for Advanced Cloud Technologies under H2020," Tech. Rep., 2012. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-expert-group/roadmap-dec2012-vfinal.pdf

[5] D. Ardagna, E. Di Nitto, G. Casale, D. Petcu, P. Mohagheghi, S. Mosser, P. Matthews, A. Gericke, C. Balligny, F. D'Andria, C.-S. Nechifor, and C. Sheridan, "MODACLOUDS, A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds," in *ICSE MiSE: International Workshop on Modelling in Software Engineering*. IEEE/ACM, 2012, pp. 50–56.

[6] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proceedings of CLOUD 2013: IEEE 6th International Conference on Cloud Computing*, L. O'Conner, Ed. IEEE Computer Society, 2013, pp. 887–894.

[7] N. Ferry, F. Chauvel, A. Rossini, B. Morin, and A. Solberg, "Managing multi-cloud systems with CloudMF," in *Proceedings of NordiCloud 2013: 2nd Nordic Symposium on Cloud Computing and Internet Technologies*, A. Solberg, M. A. Babar, M. Dumas, and C. E. Cuesta, Eds. ACM, 2013, pp. 38–45.

[8] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

[9] 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: 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE Computer Society, 2012, pp. 400–406.

[10] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[11] "OMG Model-Driven Architecture." [Online]. Available: http://www.omg.org/mda/

[12] C. Szyperski, *Component Software: Beyond Object-Oriented Programming (2nd edition)*. Addison-Wesley Professional, 2011.

[13] C. Atkinson and T. Kühne, "Rearchitecting the UML infrastructure," *ACM Transactions on Modeling and Computer Simulation*, vol. 12, no. 4, pp. 290–321, 2002.

[14] T. Kühne, "Matters of (meta-)modeling," *Software and Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.

[15] 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.

[16] G. Blair, N. Bencomo, and R. France, "Models@run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[17] F. Chauvel, N. Ferry, B. Morin, A. Rossini, and A. Solberg, "Models@Runtime to Support the Iterative and Continuous Design of Autonomous Reasoners," in *Proceedings of MRT 2013: 8th International Workshop on Models@run.time at MODELS 2013: ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems*, N. Bencomo, R. France, S. Götz, and B. Rumpe, Eds. CEUR Workshop Proceedings, 2013.

[18] F. Fouquet, G. Nain, B. Morin, E. Daubert, O. Barais, N. Plouzeau, and J.-M. Jézéquel, "An Eclipse Modelling Framework Alternative to Meet the Models@Runtime Requirements," in *MODELS 2012: 15th International Conference on Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, R. B. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds., vol. 7590. Springer, 2012, pp. 87–101.

[19] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in *SPLASH/OOPSLA 2010: Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, W. R. Cook, S. Clarke, and M. C. Rinard, Eds. ACM, 2010, pp. 307–309.

[20] "Cloud4SOA EU project." [Online]. Available: http://www.cloud4soa.eu/

[21] C. Sandru, D. Petcu, and V. I. Munteanu, "Building an Open-Source Platform-as-a-Service with Intelligent Management of Multiple Cloud Resources," in *UCC 2012: IEEE 5th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2012, pp. 333–338.

[22] D. Palma and T. Spatzier, "Topology and Orchestration Specification for Cloud Applications (TOSCA)," Organization for the Advancement of Structured Information Standards (OASIS), Tech. Rep., June 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/cos01/TOSCA-v1.0-cos01.pdf

[23] M. Sellami, S. Yangui, M. Mohamed, and S. Tata, "PaaS-Independent Provisioning and Management of Applications in the Cloud," in *CLOUD 2013: IEEE 6th International Conference on Cloud Computing*, L. O'Conner, Ed. IEEE Computer Society, 2013, pp. 693–700.

[24] G. Brataas, E. Stav, S. Lehrig, S. Becker, G. Kopčak, and D. Huljenic, "CloudScale: scalability management for cloud systems," in *ICPE 2013: 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 335–338.

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

[26] J. Shao, H. Wei, Q. Wang, and H. Mei, "A Runtime Model Based Monitoring Approach for Cloud," in *CLOUD 2010: IEEE 3rd International Conference on Cloud Computing*. IEEE Computer Society, 2010, pp. 313–320.