CloudMPL: A Domain Specific Language for Describing Management Policies for an Autonomic Cloud Infrastructure

Marwah M. Alansari[1], Andre Almeida[2], Nelly Bencomo[3] and Behzad Bordbar[1]

[1]*School of Computer Science, University of Birmingham, Birmingham, U.K.*

[2]*Federal Institute of Science of Education, Science and Technology, Parnamirim, Brazil*

[3]*School of Computer Science, Aston University, Birmingham, U.K.*

Keywords: Management Policies, Rule Language, Domain Specific Language, Autonomic Architecture, Cloud Infrastucture.

Abstract: To benefit from the advantages that Cloud Computing brings to the IT industry, management policies must be implemented as a part of the operation of the Cloud. Among others, for example, the specification of policies can be used for the management of energy to reduce the cost of running the IT system or also for security policies while handling privacy issues of users. As cloud platforms are large, manual enforcement of policies is not scalable. Hence, autonomic approaches for management policies have recently received a considerable attention. These approaches allow specification of *rules* that are executed via rule-engines. The process of rules creation starts by the interpretation of the policies drafted by high-rank managers. Then, technical IT staff translate such policies to operational activities to implement them. Such process can start from a textual declarative description and after numerous steps terminates in a set of rules to be executed on a rule engine. To simplify the steps and to bridge the considerable gap between the declarative policies and executable rules, we propose a domain-specific language called CloudMPL. We also design a method of automated transformation of the rules captured in CloudMPL to the popular rule-engine Drools. As the policies are changed over time, code generation will reduce the time required for the implementation of the policies. In addition, using a declarative language for writing the specifications is expected to make the authoring of rules easier. We demonstrate the use of the CloudMPL language into a running example extracted from a management energy consumption case study.

1 INTRODUCTION

Management of Cloud infrastructure supported by autonomic techniques has recently received a considerable attention (Beloglazov and Buyya, 2010; Mi et al., 2010; Maurer et al., 2013; Borgetto et al., 2012). Several existing autonomic techniques make a use of rule-based systems (Mi et al., 2010; Maurer et al., 2013; Borgetto et al., 2012; Alansari and Bordbar, 2013). The rule-based systems operate by using a set of statements in a form of "if $< condition >$ then $< action >$ " which are known as rules executed by a rule-engine. Rule-based frameworks have been used to automatically trigger the live-migration of virtual machine by using different types of constraints rules. The constraints rules are formulated as management policies (Borgetto et al., 2012)(Alansari and Bordbar, 2013).

In (Alansari and Bordbar, 2013), the authors propose an architectural framework for automatically executing management policies on Cloud infrastructure. The rule engine is integrated to work with Cloud management system to control the migration of running virtual machines among hosting nodes. Policy Rule Engine and Cloud Manager are communicated through sensors and actuators. The sensor is directly interlinked with Cloud APIs for management of virtual machines that are responsible for requesting monitoring parameters such as Estimated Energy Consumption and Current Resource Usage. Whilst the actuator uses action APIs that directly launch the management actions, such as virtual machine migration action (Alansari and Bordbar, 2013).

There are three steps defined for designing suitable policies which can be executed into the autonomic framework proposed in (Alansari and Bordbar, 2013) or similar frameworks as in (Maurer et al., 2013). These steps are Policy Authoring, Policy Implementation and Policy Deployment and Execution

M. Alansari M., Almeida A., Bencomo N. and Bordbar B..
CloudMPL: A Domain Specific Language for Describing Management Policies for an Autonomic Cloud Infrastructure.
DOI: 10.5220/0005446004510462
In *Proceedings of the 5th International Conference on Cloud Computing and Services Science* (CLOSER-2015), pages 451-462
ISBN: 978-989-758-104-5

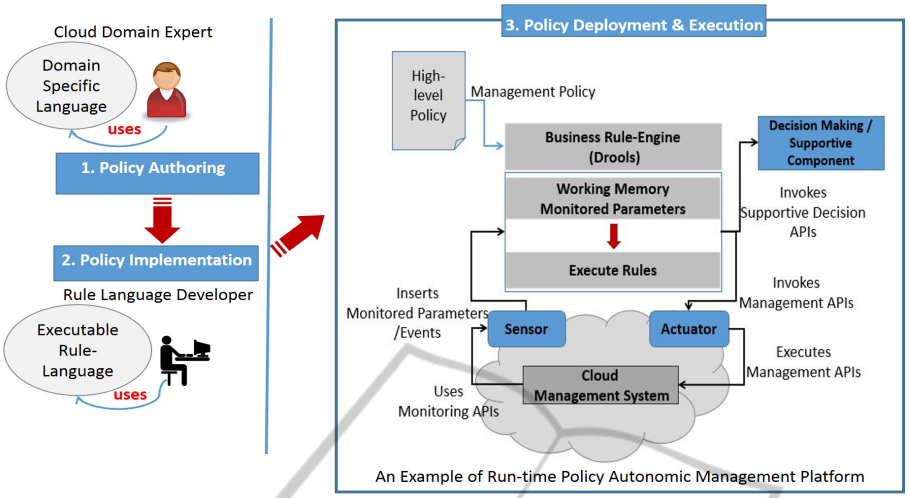

Figure 1: The steps for designing and publishing management policies.

(see Figure 1). Cloud-domain experts and rule developers are the roles which are involved during the design process of the management policies. Cloud-domain experts write the policies in simple plain English sentences "if/then ". Rule developers encode the described policies using a rule language and generating executable management policies as well as Cloud-infrastructure domain model (Alansari and Bordbar, 2014).

Management policies are updated regularly because of alteration in the technical environment, changes in the regulations, modifications of SLAs and changes in business requirements. While going thorough this process, there is a semantic gap between Policy Authoring and Implementation levels. This gap can be bridged by using a combination of *Domain Specific Language (DSL)* for authoring of the rules and *automated code generation* for producing rules which are executed in the rule-engine. The *Domain Specific Language (DSL)* supplies expressive representations, textual notions and high-level specification to describe policies written using plain English. Such a language helps the domain experts, who may have little knowledge about formulating rules, to avoid the complexity of writing management policies. On the other hand, *Code generation* is used to convert the policies captured in the DSL to be an input for executable Rule Languages such as Drools.

This paper makes two contributions. Firstly, we propose Cloud Management Policy Language (CloudMPL), a domain specific language to support Cloud-domain experts to specify policies. CloudMPL is a textual language with a specification partially inspired by the RELAX Language(Whittle et al., 2010). Secondly, we design a set of mapping rules used for automatic transformation from CloudMPL to rule languages based on Object Patterns such as Drools (JBossCommunity, 2011) and JRules (IBM-ILOG, 2007). The objective is to built a foundation for the automatic generation of a management policy from Policy Authoring to Implementation levels and hence bridge the gap described above.

The paper is organised as follows. Section 2 has an overview about rule-based systems and domain specific languages used in Cloud environment. CloudMPL language and its specification are discussed in Section 3. The specification of management policies in an executable rule language is explained in Section 4. Section 5 includes the designed mapping rules from CloudMPL to Drools. Section 6 contains a demonstration of CloudMPL in XText for authoring migration policies in energy management case study. Finally, the conclusion is presented in Section 7.

2 BACKGROUND AND RELATED WORK

This section introduces concepts about rule engines and domain-specific languages.

2.1 Rule Engine and Rules

Policies can be executed via a rule-based engine such as Drools(JBossCommunity, 2011) and ILOG JRules (IBM-ILOG, 2007). Both Drools and JRules engines are based on the enhanced version of the RETE-Algorithm for supporting Object Oriented Pattern. RETE-Algorithm was introduced by Charles Forgy; it is one of the efficient implementation rules inference engines. The algorithm represents rules as an acyclic

graph to form a RETE-network and provides a pattern matching process (Forgy, 1982).

The architecture of the rule engine is shown in Figure 2. Both Drools and JRules rule engines include Rule-Base and Working Memory. Rule Base is a long-term memory in which rules are stored. Working Memory is a type of short-term memory, which contains Facts that need to be evaluated by the inference engine. Facts are object models that contain attributes that illustrate domain data for an application. Agenda is the place where a rule that has become active is stored for later in order to fire satisfied rules. The agenda uses resolving conflicting methodology for ordering the execution of active rules (Forgy, 1982). In order to execute policies in a rule-based engine, policies have to be written in a form of rule-sets. A rule-set consists of a number of conditions followed with a set of sequential actions in which the rule-set is expressed in the following format:

when (condition statements) **then** (action statements)

Figure 3 illustrates a sample of a constraint-based policy encoded in Drools Language. The function of this rule-set is to allow migration from Host1 to either Host2 or Host3. The action will be invoked which requests to migrate a virtual machine from Host1 when the conditions are satisfied.

2.2 Domain Specific Languages DSLs

Domain-specific languages (DSLs) are languages which are designed to be used in a specific application domain. DSLs languages provide a special feature in terms of the expressiveness and simplicity compared with general-purpose programming languages (Mernik et al., 2005). Using DSLs have several advantages. They can speed up the development time since the language is designed to be used for specific environment. In addition, the language can assist to reduce the number of domain and programming expertise which are required (Mernik et al., 2005). Furthermore, the domain-language is an extendible and

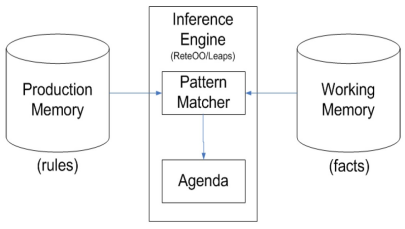

Figure 2: The architecture of OO-RETE engine.

```
rule "POLICY1"
dialect "mvel"
salience 106
no-loop
when
      Host(name=="host1" && Violation_Percenateg <20 && Energy_Consumption >2000)
then
      CloudManager.MigrateHost3OrHost2()
```

Figure 3: A sample of a constraint management policy expressed in Drools Language.

a machine readable which allows to build auto-code generation tools in order to reduce the development time (Mernik et al., 2005). To accomplish these features provided by DSLs, designing such languages requires the experience in both domain-knowledge and language development. In our research, we focus on domain-specific languages designed for Cloud-platform.

2.2.1 DSLs used in Cloud Computing

There are extensive research for proposing many DSLs for automating the deployment of applications into Cloud-environment. One of these languages is *Crawl* which is apart of *Cloud Crawler* environment proposed for automating the execution of applications performance test in IaaS used by Cloud application developers (Cunha et al., 2013). *Crawl* is a declarative and an extensible domain-specific language (DSL) to provide a high-level specification that captures all the technical important information for executing application performance tests(Cunha et al., 2013). Instances of these information are the configuration parameters and the quantity of the resources allocated to application components(Cunha et al., 2013). The language's textual notion is described via YAML. Furthermore, the language allows using XML and JSON to define new specification of test scenarios(Cunha et al., 2013).

Neptune is also another domain specific language (DSL) designed to automate the configuration and deployment HPC applications executed in Cloud(Bunch et al., 2011). The objective of *Nuptune* is to provide portability and flexibility to the developers of HPC(Bunch et al., 2011). *Neptune* is a meta-programming extension of the Ruby programming language with a flexibility to run large number of ruby's libraries which are designed to communicate with Cloud- infrastructure(Bunch et al., 2011). *Neptune* programmes allow users to write Ruby scripting code and also Nuptune programs can also be used in Ruby programs using *neptune keyword*. The programmes of *Nuptune* is composed of one or more invocations for jobs to be processed in Cloud ser-

vices(Bunch et al., 2011). The language is integrated to run into AppScale which is an open-source Cloud environment that uses Google App Engine APIs(Bunch et al., 2011).

Pim4Cloud DSL is a Platform Independent Model for Cloud-based application which is designed using a component-based approach(Brandtzæg et al., 2012). A Cloud application- designer models the application by using *Pim4Cloud DSL*. Meanwhile, at the other side the available resources for the modelled application are specified by Cloud provider (Brandtzæg et al., 2012). The *Pim4Cloud* has an interpreter which is used to match the assigned resources to application requirement. *Pim4Cloud DSL* is implemented in to Scala which includes different set of codes for modelling different topology for Cloud applications (Brandtzæg et al., 2012). The syntax of the *Pim4Cloud DSL* starts by defining the application as an abstract class which can factorise the shared entities. Each application topology can extend the abstract class. *Pim4Cloud DSL* platform supports a static analysis for the modelled application and also allow the deployment of Cloud-component to be reused (Brandtzæg et al., 2012).

3 CloudMPL LANGUAGE

CloudMPL can be applied instead of using Plain English sentences for writing the management policy at early stage. CloudMPL is a textual language that specifically is designed to be used by Cloud domain-experts to describe management policies before interpreting them to an executable rule language. CloudMPL is enriched with domain vocabularies and expressive operators for expressing conditions and actions parts which are partially inspired by the RELAX Language(Whittle et al., 2010). CloudMPL is targeting to author management policies which will be executed in Infrastucture-as-a-Service(IaaS) Cloud model.

When Cloud-domain experts specify policies, basically they are concerned about if some specific conditions are met, based on resources that compose a Cloud infrastructure, and actions that might be taken upon these conditions. Therefore, CloudMPL is designed to focus on the specification of Cloud-related resources and policies. The specification of CloudMPL includes a domain-vocabulary, a meta-model, a set of operators, which are extracted from RELAX language to describe various types of conditions might be found in a management policy, and a grammar for using the language.

3.1 CloudMPL Meta-model

Figure 4 presents the meta-model for CloudMPL language. In the meta-model described in Figure 4, a *Resource* is an element that can be managed and/or monitored which could be a host, a virtual machine, a cluster or any kind of resource that can described in term of *Properties*. Those *Properties* should represent any kind of information that can be monitored or used to describe a *Resource*. Every property has a *Type* that can be either a built-in type, such as *Number* or *String*, or could be a *Resource* created by the user. Each *Policy* is composed of several *conditions*, detailed in terms of constraints. *Constraints* are three different types which are:

1. *Time*, when an expert is interested in time related events.
2. *Location*, which is an optional element, is to check if a given resource is in a specific location.
3. *Ordinal* where raw values or a status of a given property related to a resource are checked.

In this level of abstraction, an *Action* invocation is an equivalent to a method call of any high-level programming language. The actions definition is similar to a method signature description, embedded into a *ActionManager*, which will be equivalent to an interface of a high-level programming language. To fully implement the important actions, it would require for the experts a deep knowledge on programming skills as well as specific-language information.

Table 1 gives the set of CloudMPL elements, organised into *Statement*, *Time*, *Location* and *Constraint*. The statements define the blocks of the language in terms of the main elements for expressing the policies. The *Time* and *Location* operators define conditions for time and location. The *Constraint* conditions can be applied to ordinals or status information for basic comparison.

3.2 CloudMPL Syntax

The syntax of CloudMPL expressions are defined by the grammar in Figure 5. Manageable resources are described by the *Create Resource* statement, defined by an ID and their respective properties. The *ActionManager* requires an ID and at least one action(method) to be defined. Policies are identified by an ID and composed of CloudMPL operators. As showed in Table 1 CloudMPL has *Location*, *Time* and *Constraint* operators, that can be combined using logical operator such as *AND,OR,NOT* and parenthesis. If the specified conditions are satisfied an action that belong to an *ActionManager* should be invoked.

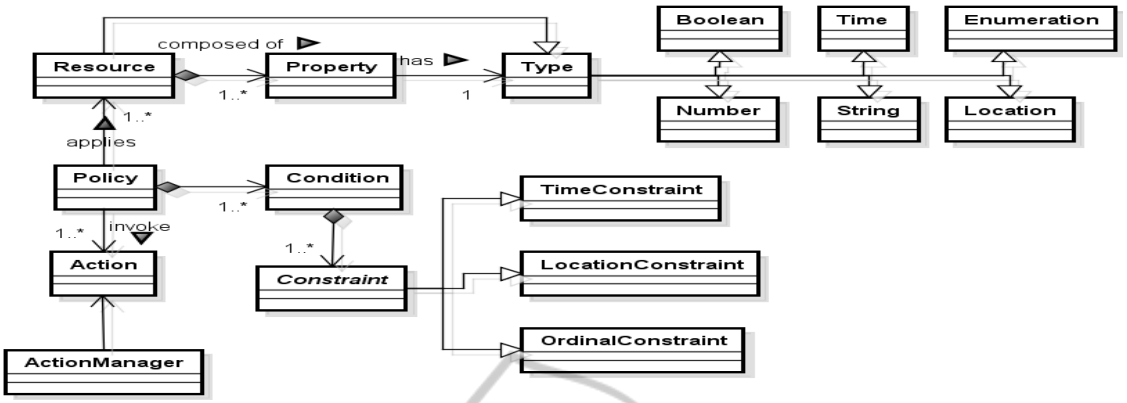

Figure 4: CloudMPL meta-model.

<**CLOUDMPL**>	::=	<RESOURCE> {<RESOURCE>} <ACTIONMANAGER> {<ACTIONMANAGER>} <CREATE_RESOURCE> {<CREATE_RESOURCE>} <POLICY> {<POLICY>}
<RESOURCE>	::=	**DEFINE RESOURCE** <ID> *AS* <PROPERTY> {<PROPERTY>}
<PROPERTY>	::=	<ID> *AS* <TYPE>
<ACTIONMANAGER>	::=	**DEFINE ACTIONMANAGER** <ID> *AS* <ACTION> {<ACTION>}
<ACTION>	::=	**ACTION** <ID><PARAMETERLIST>
<POLICY>	::=	**POLICY** <ID> **IF** <CONDITIONS> **THEN** <ACTION_INVOCATION> {<ACTION_INVOCATION>)}

Figure 5: A simplified EBNF grammar for CloudMPL.

The following steps are required for applying CloudMPL:

- **Step 01:** Modelling the resources that are going to be managed and used to describe the policies.
- **Step 02:** Definition of the possible actions used in case a policy condition is true. The Actions are created by defining an *ActionManager* element.
- **Step 03:** Writing the policies using the operators and the syntax for the CloudMPL language. Each policy should be identified by a valid ID.

In order to illustrate the usage of CloudMPL, consider the following policy:

- **Policy1:** if the violation percentage rate is less than 20% in Host1 and the energy consumption is above than 2000 kwph in Host1 then it is necessary to migrate the contents of Host1 to Host3 or Host2

```
1  DEFINE RESOURCE Host AS
2  BEGIN
3  Violation_Percentage AS Number
4  Energy_Consumption AS Number
5  END
6  DEFINE ACTIONMANAGER Manager AS
7  BEGIN
8  ACTION Migrate_Alternative_Host Host origin,
9  Host alternativeA , Host alternativeB
10 END
11 CREATE host1,host2,host3 AS Host
12 POLICY policy1
13 IF ((host1.Violation_Percentage FEW_AS 20)
       AND (host1.Energy_Consumption MANY_AS
       2000))  THEN
14 Manager.Migrate_Alternative_Host host1 ,
       host2, host3
```

Figure 6: CloudMPL representation for the given policy.

Using CloudMPL grammar defined in Figure 5, Figure 6 presents *Policy* written in CloudMPL.

First it is necessary to create a resource called *Host* (lines 1 to 5) which describes a computer host in a private Cloud infrastructure. Considering the given policy the manager is interested in monitoring the energy consumption and the rate of violation(error rate). As stated in the policy definition the action that needs to be done if the conditions are met is to migrate contents of Host1 to Host2 or Host3. This action in CloudMPL is embedded in an *ActionManager*, called *Manager*, which has a single method *Migrate_Alternative_Host* (lines 6 to 10). Considering that the policy requires the usage of three host, it is necessary to create them (line 11). Finally the policy itself (lines 12 to 14) is defined in terms of the resources and properties, using *Constraint* operators. The CloudMPL language was implemented using XText(Xtext, 2014) and the current implementation is available for download at http://www.dimap.ufrn.br/splmonitoring/adaptmcloud/index.php

Table 1: CloudMPL Operators and Elements for Describing Management Policies.

CloudMPL Operators & Statements	Description
Statements:	
DEFINE RESOURCE *id* AS BEGIN *property* AS *type* END	Declares a Resource, by specifying its ID and properties
DEFINE ACTIONMANAGER *id* AS BEGIN ACTION *id*,*parameters* END	Declares an *ActionManager* if a set of actions. Each action is declared with a ID and a set of parameters
CREATE *id* as *resource*	Creates a *Resource* with the given ID
POLICY *id* IF *conditions* THEN *actions*	Declares a policy with the given ID and starts the policy conditions definition, followed by a action(s) call(s).
Time Operators:	
resource.time AFTER *threshold*	Checks if the property of *Time* type is after the given threshold
resource.time BEFORE *threshold*	Checks if the property of *Time* type is before the given threshold
resource.time BETWEEN $threshold_a$ TO $threshold_b$	Checks if the property of *Time* type is in the specified interval.
Location Operator:	
resource.location IN *location*	Checks if a given *Resource* is in the specified *location*.
Constraint Operators:	
resource.property FEW_AS *value*	Checks if the ordinal representation for a property is less than a given value
resource.property MANY_AS *value*	Checks if the ordinal representation for a property is greater than a given value
resource.property IS *status*	Checks if the a given property is within one of the following status: HIGH, NORMAL or LOW

4 THE SPECIFICATION OF MANAGEMENT POLICIES IN RULE LANGUAGE

Management policies expressed in CloudMPL can be directly mapped to executable management policies by designing mapping rules between two languages. To produce such policies, any CloudMPL policy is encoded as a special form of production rules (more information about the productions rules can be found in (REWERSE, 2012)). These production rules follow a defined specification for formulating both condition and action parts in Rule Language which can be applied to either Drools(JBossCommunity, 2011) or JRules (IBM-ILOG, 2007). In this work, we briefly discuss the specification for formulating conditions and action parts for a policy in an executable rule language. The specification includes the meta-model for conditions, the classification for operators used in conditions and the types of actions used by a management policy. Both CloudMPL and rule language specification will be used for designing a translator from CloudMPL to Drools or JRules in future.

Table 2: The syntax for conditional expressions for a management policy.

Expression	<property:>	<operator:>	<value_expr:>
Constraints-1:	monitorable_prams	Comparison	DataTerm
Constraints-2:	monitorable_pram_status	Level Specification	DataTerm
SelectTargetHost:	id or name	TargetHost Selection	DataTerm
TargetHost Location:	location	Location	DataTerm
TargetHostTime:	current_time_status	Time	ObjectTerm

4.1 Conditions Meta-model for a Policy

Conditions for a management policy can be expressed through the meta-model presented in Figure 7 which is inspired from URML meta-model (REWERSE, 2012). This meta-model is resulted from our classification for general rules used for management purposes in Cloud.

In Figure 7, a single condition in a management policy is a boolean expression which can be composed with other conditions by using *composition operators*. From URML rule meta-model (REWERSE, 2012), we extracted some elements for modelling various conditions. These elements are *Term*, *DataTerm*, *ObjectTerm*, *uml_property*, and *Object Variable* (REWERSE, 2012).

In Figure 7, the conditional expression are classified into five types. Each expression in the condition meta-model uses a property. The property are extracted from Target Host that is running in a Cloud-platform. In addition, each expression also has a *value_expr* which might be of the following types: Data Term, Object Term, or uml property. Furthermore, suitable operators are grouped to match each conditional expression type. The operators are presented in Figure 8. Thus, by using both Figure 7 and Figure 8, each conditional expression and its syntax are explained as follows:

1. **ConstraintsExpr:** a comparative condition used to compare monitorable parameters against a

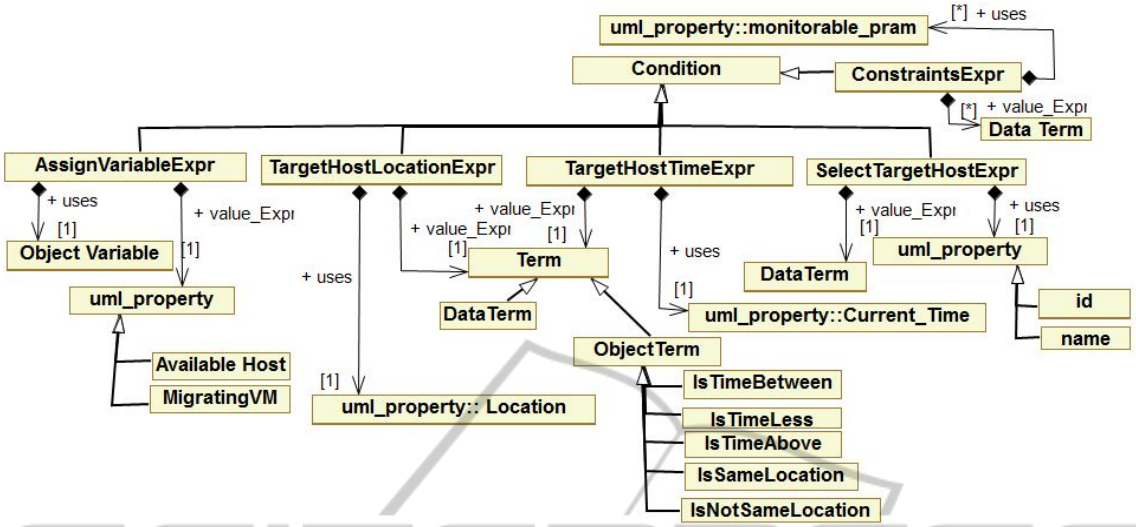

Figure 7: The meta-model for conditions of a management policy in Rule Language (Drools and JRules).

specified threshold value or to specify status of monitorable parameters. Examples for monitorable parameters are CPU_Usage, Violation_Percentage and Energy_Consumption. ConstraintsExpr has two different syntaxes, which are presented in Table 2.

2. **SelectTargetHostExpr:** an identification expression, which is used to select a targeted host. This expression is necessary to be included in a management policy. The expression syntax is shown in Table 2.
3. **AssignVariableExpr:** a selection expression, which is to get values from some properties and to assign them to an object variable. This expression can be used in a management policy for extracting variables which are required by management APIs. The expression is optional to be included in the policy. The syntax for the expression is different from the syntax for other expressions which is as follows:
   $< operator : Assignment\$ >< property : ObjectVariable >< operator : Assignment :>< property : ObjectTerm >$
4. **TargetHostLocationExpr:** a location_based expression, which is used to specify a geographical location of a target host. Since a physical Cloud host can be located at any location around the world, the policy meta-model should allow an option for such a restriction. This expression is an optional in the policy based on the requirement. The syntax for the expression is shown in Table 2 where its Data Term can be either of String type or GeoLocation which is Enumeration type. An example for TargetHostLocation is $location == GeoLocation.Asia$.
5. **TargetHostTimeExpr:** a time_based expression that specifies the time status at a target host. Any target host in Cloud-platform has some operations to deal with time expression.
   These operations are $IsTimeBetween(< Time\_Begin >,< Time\_End >)$, $IsTimeLess(< Time\_Literal >)$, and $IsTimeAbove(< Time\_Literal >)$. The syntax for the time_based expression is presented in Table 2. The following expression is a simple expression for checking time status:
   current_time_status== IsTimeBetween(16.00,23.00)

4.2 Policy Action Description

The action of a management policy in a rule language is expressed as action expressions. These action expressions can be either expressions for assigning values or expression for invocation actions. To simplify the transformation process for future, we only use invocation actions expression from the rule language which is denoted as *InvocationActionExpr*in (REWERSE, 2012). In any management

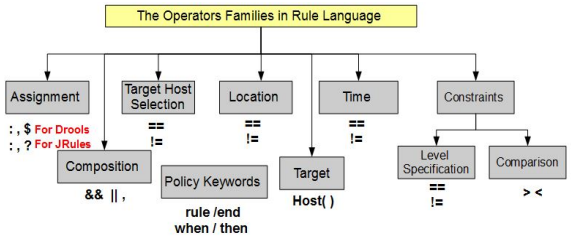

Figure 8: The possible common rule language operators families for expressing a management policy.

Table 3: Operations used by Cloud Manager instance in a management policy described in Drools.

< OperationName >	*< Parameters : Type >*
Migrate	Original: TargetHost
MigrateAlternativeHosts	Original: TargetHost , Destination1:TargetHost, Destination2: TargetHost
MigrateToLocation	Original: TargetHost , LocationName: String
ReportingNoMigration	Original: TargetHost
Calculate	Original:TargetHost

policy, the invocation action expressions include calls for management APIs/Operations specified in Cloud manager. The syntax used for expressing *InvocationActionExpr* is:
CloudManager.Operation_Name(parameters);

In this syntax, *CloudManager* represents the instance of a Cloud manager which has a number of management operations. Each defined operations for *CloudManager* instance has a number of parameters which are necessary for migration of virtual machines, reporting information and calculating service at the target host side in Cloud-platform. Table3 shows each defined operation and its related parameters.

5 THE TRANSFORMATION FROM CloudMPL TO DROOLS

After presenting CloudMPL language for expressing management policies at the description level and the specification for formulating such policies at the implementation level, the transformation between CloudMPL and Drools is proposed. The objective is to build the foundation for automatically generating an executable management policy from the description level.

The transformation process requires to have a conceptual mapping from CloudMPL to Drools Language. Therefore, designing a set of mapping rules from CloudMPL to Drools for both conditions and actions parts is necessary. To design these mapping rules, we used CloudMPL meta-model as well as its syntax and Drools specification mentioned in Section 4. Firstly, the mapping step starts by presenting the

Table 4: Mapping generic syntax and keywords for a policy in CloudMPL to Drools.

CloudMPL Generic Syntax		Drools Generic Syntax
POLICY *< ID >*	→	**rule** *< ID >*
IF	→	**when**
< CONDITIONS >		*< Host(Conditions) >*
THEN	→	**then** *< Actions >*
< ACTION_INVOCATION >		
{*< ACTION_INVOCATION >*}		end

Table 5: Mapping CloudMPL conditions to Drools conditions using Table 1 in Section 3 and Table 2 in Section 4.

CloudMPL Expression	**Drools Expression**
<attribute:TIME> *AFTER* <value:threshold>	TargetHostTimeExpr
<attribute:TIME> *BEFORE* <value:threshold>	TargetHostTimeExpr
<attribute:TIME> *BETWEEN* < value: threshold_a > *TO* <value:threshold_b>	TargetHostTimeExpr
IN <value:ID> <value:location>	SelectTargetHostExpr + TargetHostLocationExpr
<attribute:monitorable> *FEW_AS* | *MANY_AS* <value:threshold>	ConstraintsExpr-1
<attribute:monitorable>*IS* <value:status>	ConstraintsExpr-2

mapping for the general syntax for a management policy and keywords in both CloudMPL and Drools.

Table4 shows the mapping of generic syntax and special keywords from CloudMPL to Drools. It is noticeable from the generic syntax in Table4 that any statement between *IF* and *THEN* is mapped as Conditions in Drools which it should be enclosed with Target operator mentioned in Figure 8. Furthermore, any statement after *THEN* is mapped as Actions in Drools. The mapping of both conditions and actions requires more explanation which will be appeared in the following subsections.

5.1 Mapping Conditions

In CloudMPL, a condition block consists of one or more condition. Therefore, any condition in CloudMPL can be structured as an attribute, an operator and a value. The attribute in CloudMPL is usually written before CloudMPL operator. Usually, the value is after CloudMPL operator. Thus, the mapping for conditions is shown in Table 5 which applies the following mapping rules:

1. The dot operator < . > in CloudMPL is mapped as '==' operator and <value:ID or Name> is mapped as <value_expr: DataTerm>.
2. *After* operator is mapped as '==' combined with <ObjectTerm:IsTimeAbove> in Drools.
3. *Before* is mapped as '==' combined with <ObjectTerm:IsTimeLess> in Drools.
4. *BETWEEN*,*TO* operator is mapped as '==' combined with <ObjectTerm: *IsTimeBetween*>.
5. < value: threshold > is mapped as <Time_Literal> parameter for both IsTimeAbove and IsTimeLess in Drools.
6. < value: threshold_a > and <value:threshold_b> are mapped as <Time_ Literal_ Begin> and

<Time_Literal_End> parameters for IsTimeBetween in Drools.

7. <attribute:Time> is mapped as <property: current_time_status>.
8. *IN* is mapped as '==' operator and <value:location> is mapped as <value_expr: DataTerm> which can be either String or Enumeration.
9. *FEW_AS* or *MANY_AS* are mapped as Comparison operators.
10. <attribute:monitorable> is mapped as <property: monitoriable_parms> and <value: threshold> is mapped as DataTerm.
11. *IS* operator is mapped as '==' operator and <value:status> is mapped as <DataTerm: Status>.
12. <attribute:monitorable> in *IS* expression is mapped as <property: monitoriable_ Parms_ status>.
13. *AND* and *OR* is mapped as && and || operators in Drools receptively.

To elaborate the mapping of policy condition, we provide a sample of conditions which are written in both CloudMPL and Drools in Figure 9. These conditions are for expressing a management policy extracted from Energy Management Running Example presented in Section 6. In Figure 9, the first statement is CloudMPL expression for three conditions, which are Violation_Percentage *FEW_AS* 20, Energy_Consumption *MORE_AS* 2000, *host*1. These conditions are composed in CloudMPL by *AND* operator. The same figure also includes conditions expressed in Drools which map CloudMPL conditions. In Figure 9, the arrows represent the types of the mapping rules that can be applied.

5.2 Mapping Actions

In CloudMPL, any action statement is mapped as a call method for management operations in a policy expressed in Drools. The mapping rules for actions are:

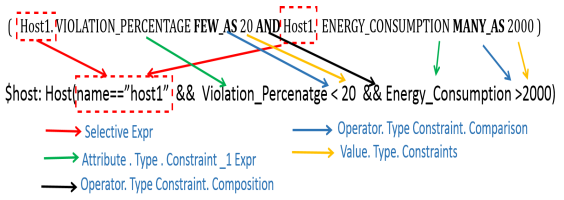

Figure 9: Using mapping rules for mapping CloudMPL condition block to Drools condition part for a policy.

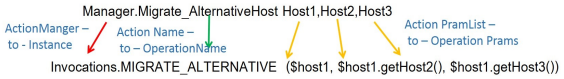

Figure 10: Using mapping rules for mapping CloudMPL action block to Drools action part for a policy.

1. Action $< ID >$ in CloudMPL is mapped as the Name of the operation in CloudManager (See Figure 10).
2. The parameters List, which includes a set of Parameter Expression, is mapped as the operation parameters in CloudManager.
3. In CloudMPL, Parameter Expression consists of $< TypeID >$. Type is mapped as either $< ObjectTerm >$ or $< getObjectRef >$ in Operation. Whilst ID is the mapped as the name of the parameter.
4. In CloudMPL , if Type is Host and it is the first parameter in the statement, then it is mapped as the Original and its type is TargetHost in Drools.
5. In CloudMPL, if Type is Host and is not the first parameter, then it is mapped as either to Destination1 or Destination2 based on ordering parameters in Drools.

Figure 10 illustrates applying the mapping rules for action from CloudMPL to Drools. The figure includes an operation defined in Table3. The operation has three parameters which are Host1, Host2, Host3. In Drools mapping, Action ID is mapped as Migrate_Alternative_Host. Whilst the first parameter is mapped as $\$host1$. Both Host2 and Host3 are mapped as $\$host1.getHost2()$ and $\$host1.getHost3()$, respectively.

The previously mentioned mapping rules for both conditions and actions parts will be used to design an interpreter to automatically or semi-automatically generate Drools codes for policies that would be executed into the architecture shown in Figure 1. The Drools code generation will be a future step.

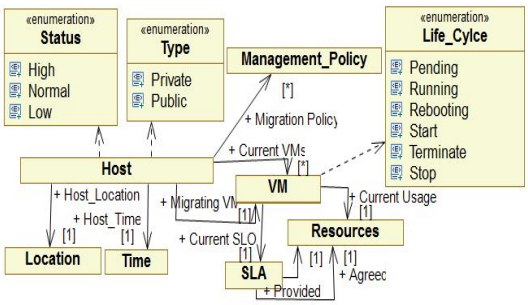

Figure 11: An UML class diagram for Cloud infrastructure used in the case study.

```
Define Resource Host as
begin
Violation_Percentage as Number
Violation_Status as Status
CPU_Usage as Status
Energy_Consumption as Number
Current_Time as Time
End
DEFINE ACTIONMANAGER Manager AS
BEGIN
ACTION Migrate_Alternative_Host Host origin,
Host alternativeA , Host alternativeB
ACTION NoMigrate
ACTION MigrateLocation Host origin, Location location
END
CREATE host1,host2,host3 AS Host
```

(a) CloudMPL Declarations

```
POLICY policy1
IF ((host1.Violation_Percentage FEW_AS 20) AND (host1.Energy_Consumption MANY_AS 2000))
 THEN
Manager.Migrate_Alternative_Host host1 , host2, host3

POLICY policy2
IF ((host1.Current_Time AFTER 23:00) AND ((host1.Violation_Status IS HIGH) OR
(host1.CPU_Usage is HIGH)))  THEN  Manager.NoMigrate

POLICY policy3
IF ((host1.Current_Time BETWEEN 01:00 TO 07:00) AND ((host1.Violation_Status 20 IS HIGH) OR
(host1.CPU_Usage is HIGH)))
 THEN Manager.Migrate_Alternative_Host host1 , host2, host3
```

(b) CloudMPL Expressions_1

```
POLICY policy4
IF ((host1.Current_Time AFTER 23:00) AND ((host1.Violation_Status IS HIGH)
OR (host1.CPU_Usage is HIGH)))  THEN
Manager.NoMigrate

POLICY policy5
IF ((host1.Current_Time BEFORE 09:00))  THEN
Manager.NoMigrate

POLICY policy6
IF ((host1.Location in 'ASIA') and (host1.Current_Time Between 16:00 to
23:00) and ((host1.Violation_Status IS HIGH) OR (host1.CPU_Usage is HIGH)))
Service.MigrateLocation  host1, 'ASIA'
```

(c) CloudMPL Expressions_2

Figure 12: A sample of CloudMPL (XText) for management policies used in the case study.

6 CloudMPL IN PRACTICE

CloudMPL is used as a language for expressing a number of management policies extracted from Management Energy Consumption Case Study presented in (Alansari and Bordbar, 2013). The Management Energy Consumption Case Study implemented in OpenNebula(Toraldo, 2012) which is a Cloud-platform management system. In the case study, there are a set of management policies which are enforced in Drools rule-engine which periodically responses to changes in three monitored parameters. The parameters are average CPU_usage for running VMs, Violation Rate and Average Energy Consumption for private hosts per hour. The engine automatically controls the migration action for running virtual machines deployed on three private physical hosts. The hosts runs Ubuntu OS and KVM as virtualization environment. The policies should trigger a migration action for running virtual machines and may switch off idle hosts. The policies which are provided in the case study are written in Drools Rule Language (Alansari and Bordbar, 2013)). Figure 11 presents the UML model for the Cloud-infrastructure for the used example.

In Figure 11, the Cloud infrastructure consists of three main components, which are Host, VM images and SLA classes. The Host class runs a number of virtual machines and has a number of monitorable attributes such as CPU_Usage, Energy_Consumption,etc.). Furthermore, Host class has a time and also a location attribute, which represents the geographical location for the host. Each running virtual machine in the model belongs to only one Cloud consumer. Thus, each running VM instance is associated with only one SLA and also has a life cycle (For more details about Figure 11 see (Alansari and Bordbar, 2014)).

We took some management policies encoded into Drools and we used CloudMPL to write them. Both CloudMPL Declarations and CloudMPL Policies for the case study, which are implemented using XText(Xtext, 2014), are shown in Figure 12.

Looking at Figure 12, there are six policies expressed in CloudMPL which demonstrate the usage of all operators suggested by the language. Policy 1 is constraints and it requires to use monitorable parameters and uses the CloudMPL operators *FEW_AS* and *MANY_AS*. Whilst policy 2, policy 3 and policy 4 are composed of both time and constraints expressions. Both policy 2 and policy 4 use the operator *After* whereas policy 3 includes the ClodMPL time operator *Between / To*. The constraints operator used in these policies is *IS*. Policy 5 has only a single time expression which uses the CloudMPL time operator *Before*. The final policy, which is policy 6, contains a location expression besides the time and constraints conditions. The location condition uses the CloudMPL operator *IN*.

6.1 The Interpretation of CloudMPL Policies to Drools

We applied the mapping process introduced in Section 5 to the policies of the case study. Using the designed mapping rules explained in Section 5 and the meta-model presented in Figure 11, Drools codes are generated manually. The purpose is to test the mapping rules. A sample of Drools code for policy 1, policy 3, policy 5 and policy 6 are shown in Figure 13 which are depicted into two groups. The important Drools operators used in the conditions are highlighted in Blue.

Taking policy 3 as an example, this policy is mapped as a combination of Time and Constraints_2 Expressions which are shown in management policies conditions meta-model mentioned in Section 4. The mapping for the condition part applies the rules number 1, 4, 6, 7,11, 12 and 13 explained in Mapping Conditions at Section 5. On the other hand, all the rules proposed for mapping the action part expressed in Section 5 are applied. As a result, policy 3 will have a rule code similar to what is illustrated in Figure 13. This method is applied to all remaining CloudMPL policies captured in Figure 12.

```
rule "POLICY1"
when
 $host: Host(name=="host1" && Violation_Percenateg <20 && Energy_Consumption >2000)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
rule "POLICY2"
when
$host1: Host (name=="host1" &&  current_time_status==IsTimeBetween(01.00, 07.00) &&
        Violation_Satus ==Status. High || CPU_Usage.Status==Status. High)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
```

(a) Drools Rules "Group_1"

```
rule "POLICY1"
when
 $host: Host(name=="host1" && Violation_Percenateg <20 && Energy_Consumption >2000)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
rule "POLICY2"
when
$host1: Host (name=="host1" &&  current_time_status==IsTimeBetween(01.00, 07.00) &&
        Violation_Satus ==Status. High || CPU_Usage.Status==Status. High)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
```

(b) Drools Rules "Group_2"

Figure 13: The generated Drools rules from CloudMPL policies.

7 CONCLUSION

This paper aims at reducing the existing gap between the specification of management policies for Cloud and the implementation of policies via rule-engines. We presented CloudMPL that is a domain-specific language for specifying management policies for Cloud-environments. Furthermore, we described a method of automating the creation of various CloudMPL expressions to an executable Rule Language such as Drools. CloudMPL establishes a set of operators that deal with several kinds of constraints, from ordinal, ranging trough time and locations constraints, that can be applied to a specific or a set of Cloud resources. In addition, CloudMPL supports user-defined actions to deal with the consequences of conditions specified by the managers of Cloud computing infrastructure. The usage of both CloudMPL and the automated approach, which is based on designing mapping rules between CloudMPL and Drools, is illustrated with the help of an example related to management energy consumption by migrating virtual machines.

It is also important to highlight that, even without the transformation process fully implemented, the suggesting method for automating policy creation assists to cope with frequent changes in policies specified at the description level. Using both CloudMPL and the automated process for creation management policies can decrease the amount of time that requires to spend in implementing such policies. Due to the closeness of CloudMPL to natural languages and its declarative nature, it is easier for non-technical people to make a use of the language for specifying policies. As a future step, it is imperative to fully implement the transformation process and also to build an integrated environment that supporting the specification, translation and deployment of the policies.

REFERENCES

Alansari, M. and Bordbar, B. (2014). Modelling and analysis of migration policies for autonomic management of energy consumption in cloud via petri-nets. In *Proceedings of the The International Conference on Cloud and Autonomic Computing*. IEEE.

Alansari, M. M. and Bordbar, B. (2013). An architectural framework for enforcing energy management policies in cloud. *2013 IEEE Sixth International Conference on Cloud Computing*, 0:717–724.

Beloglazov, A. and Buyya, R. (2010). Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM.

Borgetto, D., Maurer, M., Da-Costa, G., Pierson, J.-M., and Brandic, I. (2012). Energy-efficient and sla-aware management of iaas clouds. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 25:1–25:10, New York, NY, USA. ACM.

Brandtzæg, E., Mohagheghi, P., and Mosser, S. (2012). Towards a domain-specific language to deploy applications in the clouds. In *Cloud Computing 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 213–218.

Bunch, C., Chohan, N., Krintz, C., and Shams, K. (2011). Neptune: A domain specific language for deploying hpc software on cloud platforms. In *Proceedings of the 2Nd International Workshop on Scientific Cloud Computing*, ScienceCloud '11, pages 59–68, New York, NY, USA. ACM.

Cunha, M., Mendonca, N., and Sampaio, A. (2013). A declarative environment for automatic performance evaluation in iaas clouds. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 285–292.

Forgy, C. L. (1982). Rete : A fast algorithm for the many patternimany object pattern match problem. *Artificial Intelligence*, 19:17–37.

IBM-ILOG (2007). Ilog jrules techincal. http://logic.stanford.edu/poem/externalpapers/iRules/WP-JRules50Strengths.pdf.

JBossCommunity (2011). Drools tools reference guide.

Maurer, M., Brandic, I., and Sakellariou, R. (2013). Adaptive resource configuration for cloud infrastructure management. *Future Generation Computer Systems*, 29(2):472 – 487.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344.

Mi, H., Wang, H., Yin, G., Zhou, Y., Shi, D., and Yuan, L. (2010). Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In *IEEE International Conference on Services Computing*, pages 514–521. Ieee.

REWERSE (2012). Uml-based rule modeling language. http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=URML.

Toraldo, G. (2012). *OpenNebula 3 Cloud Computing*. PACKT Publishing, Birmingham B3.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H., and Bruel, J.-M. (2010). Relax: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196.

Xtext (2014). Xtext textual domain-specific language (dsl). http://www.eclipse.org/Xtext/.