Elastically Ruling the Cloud: Specifying Application’s Behavior in Federated Clouds

Daniel Morán*, Luis M. Vaquero†, and Fermín Galán*
*Telefonica Investigacion y Desarrollo
†Hewlett-Packard Labs
Email: dmj@tid.es,luis.vaquero@hp.com,fermin@tid.es

Abstract—Most Infrastructure as a Service clouds present limited capabilities to control how a service behaves at runtime, once it has been deployed, beyond basic low-level scalability rules for VMs. Higher-level approaches fail to provide mechanisms for a fine-grained level of control of the service at runtime, being only focused on scaling. These scalability rules are based on an ad hoc “grammar” that is not expressive enough to reflect other desired control mechanisms at runtime (e.g. reconfigurations, changes in the rules or in the components of the application, re-tiering, etc.). Here, we present an analysis on different alternatives for supporting such features. The Rule Interchange Format (RIF) emerges as a likely candidate to support the required flexibility and so it is proved in a typical use case. Also, a preliminary implementation of a mapping mechanisms is offered to parse RIF rules to widespread rule engines such as Drools and Jess.

I. INTRODUCTION

Infrastructure as a Service (IaaS) clouds have become a reality with regard to the deployment of virtual machines (VMs) in third-party managed infrastructure resources. These VMs no longer need to be managed or controlled by the service provider (SP), but it is to the IaaS provider to offer these services [1][2]. IaaS providers typically expose their services through WSDL or REST APIs: Amazon API [3] or VMware vCloud [4] are some examples. However, the trend is to provide higher level services making IaaS and Platform as a Service (PaaS) clouds converge, such as Amazon’s Cloud Watch and AutoScale. Also, RightScale manages the creation and removal of VMs according to queues or user-defined hardware and process load metrics [5].

All the systems above present limited capabilities to control how a service behaves at runtime. Recent approaches extended the available capabilities by offering a holistic mechanism for controlling how a given service (rather than individual VMs) should scale [6]. These authors also fail to provide mechanisms for a fine-grained level of control of the service at runtime, being only focused on scaling. Their system, Claudia, receives a file with a complete description of the service, based on an extended version of the Open Virtualization Format (OVF) [7], an standard from the Distributed Management Task Force (DMTF)†. Although this set of extensions allows to define scalability rules at runtime, they are specified in an ad hoc manner. In addition, these scalability rules are later mapped to JBoss Drools², but other cloud providers may use another rule engine (RE), which reduces OVF re-usability across IaaS/PaaS clouds. Also, the above ad hoc “grammar” (i.e. a grammar for a particular purpose, not designed with generality in mind) is not expressive enough to reflect other desired control mechanisms at runtime. This expressiveness has to be balanced by the fact that rules should be abstract enough to reflect users’ mindset, leading to ease of use of the system. In addition, the rules may need to be changed at different stages in which the service architecture (tiers, components, etc.) may be changed on-the-fly.

Thus, any system aiming at offering service behavior control features over a cloud environment will have to face a series of expressiveness- and interoperability-related problems. There is a wealth of rule languages and products that can potentially be used for the task of service runtime behavior control across federated cloud environments [8], [9], [10], [11], [12], [13], [14], [15], [16].

The rest of this paper is structured as follows: first, Section II highlights the special needs that the Cloud imposes to enforce holistic service governance. Section III presents the most relevant work on languages that can be used for specifying the governance directions based on the previous requirements. Based on the pre-selection of available languages, we will analyze them against some restrictions imposed by the rules typically used in the Cloud (Section IV). The result will be the selection of the most appropriate language to be included in the standard service definition language that describes the service as a whole. Next, a sample use case of the inclusion and usage of the selected rule language in a cloud environment (see Section V) is described. Finally, Section VI discusses the obtained results and wraps up the major lessons learned from this experience.

II. CLOUD-DERIVED REQUIREMENTS

When choosing among the different rule languages available, the desired logic expression capabilities should be taken into consideration. A priori a non-modal, first order logic (FOL) language with constraint support should suffice. However, let us analyze important requirements for rule languages arisen in the context of cloud computing.

†http://www.dmtf.org

²http://www.jboss.org/drools
Previous works concerned two type of knowledge manipulation:

- Numerical knowledge manipulation concerning performance metrics both over physical components and business-wide ones, possibly involving constraint checking over these numerical factors.
- Making decisions using forward chaining inference over the actual fact base, firing appropriate actions where necessary.

However, the approach presented by these authors [6], [7] does not allow for querying the actual fact base (backwards chaining), which may be useful for inferring consequences of a fact addition. Although a set of rules may be syntactically valid, it does not mean the rules governing the cloud service are “what the service provider is expecting”. Consequently, semantic validation against a set of well-defined ontologies of the used cloud data model is also a very desired feature. This will allow users to make ontological queries on the cloud data model that could be used to improve the decision-making process.

In addition to expressing scalability, means are needed that allow service providers to deal with complex rules governing other aspects of the service such as: retiering application components; complex scalability scenarios (e.g. some scalable services, like MySQL Cluster are composed of different kinds of nodes that scale based on different conditions and require different procedures to do it), automated backup procedures (including extracting a machine from the cluster or shutting it down to make the backup).

Different cloud vendors may rely on a wealth of Rule Engines (RE) to help enforcing the rules governing the service during its whole life-cycle. This reduces the portability of any given set of rules and calls for the usage of a standard procedure for specifying the governance rules.

III. RELATED WORK

This section present the most relevant work on rule languages specification that could be used to meet the requirements above.

First, we present some notations that could be used to specify the rules in an abstract/graphical way, which reduces their portability and machine readability. PRR (Production Rule Representation) is an Object Management Group (OMG) effort aimed to offer an abstract production rule representation format. This language is focused on a standard metamodel definition, rather than on a rule interchange format per se. Similarly, the Business Process Modeling Notation (BPMN) is an OMG standard for business process modeling\(^3\). It provides a graphical notation for specifying business processes in a Business Process Diagram (BPD).

A second group of languages that could be employed for specifying behavior rules are those mimicking natural language. Among them, the Natural Rule Language (NRL)\(^4\)

\(^3\)http://nrl.sourceforge.net/spec-action/
\(^4\)http://www.bpmn.org/

action (originally a validation language for XML) provides common programming language constructs that could be used to specify rules. In the same way, SBVR (Semantics of Business Vocabulary and Business Rules) is an OMG effort meant to representing knowledge for documenting the semantics of business vocabularies, business facts, and business rules. Both these languages provide a human-friendly, non-technical way of specifying behavior, but lack of the machine-friendly features provided by XML and are hardly interoperable.

A third group of rule languages could include those dealing with a specific domain of application. For instance, Semantic Web Rule Language (SWRL) is a rule language aimed to specify behavior on the Semantic Web; it’s based on RuleML (see below) and Ontology Web Language (OWL) and it consists of two notations: abstract syntax and XML Concrete Syntax (mixture of RuleML XML and OWL XML). Similar purpose-specific languages would include conventional workflow languages, such as Business Process Execution Language (BPEL); these also offer capabilities for expressing rules. However, they are not designed for this problem and using workflow engines to evaluate simple production rules may prove highly inefficient. Also in this general group we could gather the Positional-Slotted Language (POSL) is a human friendly Prolog-based notation for slotted, positional logic to represent rules and facts. All these specific languages are hardly generalizable to other application domains (or they become too verbose and prone to failure).

Current scenario implies a variety of high level languages that could be employed to specify application behavior in cloud environments (top boxes in Figure 1). As the Cloud continues its evolution towards higher abstraction and full automation, many cloud providers have appeared that can enforce user-defined rule by using a variety of REs. That variety of available REs lets us make a fourth rule language group (bottom boxes in Figure 1). These are not portable at all and include some RE-specific abstractions that make portability harder. For instance, IBM’s IRL (ILOG Rule Language), which defines the rule syntax, and BAL (Business Action Language), which defines ILOG action and expression syntax represent low level Production Rule languages.

In order to reduce the aforementioned portability problem, several rule interchange formats are handy that need to be evaluated against other requirements above (see Section II) to assess their appropriateness for cloud environments (mid box in Figure 1). RuleML is itself an organization devoted to the creation of standards for rule interchange, cooperating with W3C in the standardization of some common languages. RuleML is also a modular family of Rule Languages with Datalog at its core, being extensible by means of XMLSchema documents. Unfortunately, it does not allow for backward chaining or semantic validation. Other rule interchange language, ISO Common Logic, specifies a standard format for knowledge interchange based on FOL, which defines an abstract model, later used in notation sub-languages (dialects) such as the Structured English (controlled natural language), CLIF (Lisp like), XCL (XML compliant). However, the spec-
Another approach to interchange languages is the Rule Interchange Format (RIF)\(^5\), which offers an abstract and XML-based syntax with a core specification and dialects: 1) Production Rule Dialect (PRD) production rules: specify condition and action syntax. The catalog of actions is limited to asserting and retracting facts and executing atoms. 2) Basic Logic Dialect (BLD) adds common logic features not available on the core. A priori RIF offers the required portability and allows for inferences and semantic capabilities since the rules interchanged using the RIF may depend on or be used in combination with Resource Description Framework (RDF) data and RDF Schema or OWL ontologies.

IV. RIF MAPPING

Despite meeting important requirements, RIF deserves further evaluation in order to make sure rules for controlling cloud applications behavior can effectively be mapped to some other underlying RE-specific languages. These include language-specific constructs that need to be represented in RIF for the sake or portability. Also, cloud rules are typical Event-Condition-Action (ECA) or inference rules so that only a subset of RIF-PRD is needed. This subset needs to include mechanisms for expressing RE-specific constructs.

In this case, we will provide a subset of RIF-PRD considering the two most widespread RE-specific rule languages: Drools and Jess, thus demonstrating its interoperability (this does not mean, however, that a complete formal semantic validation of this equivalence is done for the whole set of languages). This subset will be defined by two requisites: being object-orientation-friendly; and having the minimum size that will cover cloud services' needs.

To achieve object orientation on the rules, frame logic will be used, mapping classes to frames and attributes to slots. This approach has some difficulties as addressed in\(^6\), which proposes an OO extension to RIF-Core. As this extension has been recently proposed and it’s not yet part of the official RIF-PRD specification, it has not been adopted, preferring a subset of frame logic that may meet the same needs with standard constructs. The difficulties highlighted in this paper involve the semantics of the assert and modify actions and the cardinality of the slot values: the assertion of a new value on a frame object’s slot add a new value to it while it will modify its previous value on a program object. To address this difficulty, the semantics of Drools and Jess insert are used for the assert statement, creating a new fact for each assert, so it isn’t necessary to deal with value lists.

Another problem of object orientation is the type of variables and frame’s slots. While RIF-PRD makes use of the types defined in XSD\(^7\), sometimes, as it will be seen in the example ontology, a slot value may not have a basic type as defined on the XML Schema namespace, but a user-defined one, for instance, another object (frame). Slot values with class type may be constrained by bounding a variable to that class in other parts of the rules and constraining the value of the slot with this variable (class names will be referred to with its IRI). While this behavior can be accomplished in Drools, Jess facts need a workaround: a variable bound to a Jess shadow fact\(^8\) is bound to the Fact object facade, instead of being bound to the object. The object reference can be retrieved from a special slot called OBJECT in the same fact.

The selected subset is finally defined by the following constraints:

- Only slotted facts\(^9\) will be used, so every fact can be directly mapped to an object and every slot to an attribute.
- For the sake of simplicity, only non-inline constraints will be used. This restriction doesn’t change the expressive power of the language, any inline constraint can be expressed as a non-inline one using slot binding (although the code might lose efficiency due to some optimizations that rely on inline expressions).
- Universal quantifiers will also not be used in the translation, due to a difference between the meaning of operator forall as used in RIF-PRD and both Drools and Jess: in the former, the forall sentence is used to qualify a variable over a whole rule, while the latter use them to qualify conditional expressions inside a pattern. This semantic difference may be solved with some logical transformations over the expressions to translate, but the details are out of the scope of this paper for sake of briefness\(^10\).
- RIF Lists of terms will not be used, as the Object Orientation paradigm suggests the treatment of slots as single value attributes (see discussion on slot types above).
- Full OO imposes another constraint on the selected subset: whenever a Frame is used in RIF with a variable

---

\(^{5}\)http://www.w3.org/standards/techs/rif#w3c_all

\(^{6}\)http://www.w3.org/2005/rules/wiki/RIF-OO

\(^{7}\)http://www.w3.org/2005/rules/wiki/DTB

\(^{8}\)Name of the Jess facts who are reflecting an object in the data model

\(^{9}\)As opposed to ordered facts. Slotted facts are unordered facts where matching is done against a slot name, instead of a position in the fact declaration

\(^{10}\)Jess documentation proposes the following translation for the universal quantifier (actually, the internal implementation of that operator): (not(and(a ?x) (not (b ?x))))
as its object, that variable has to be constrained to a class membership with a Member predicate, so that the class of each variable can be inferred from previous constraints. Having an ontology handy eases direct fact assertion as objects of the cloud data model.

- Another constraint will be held on the available data sets: all the facts used in the rules must be defined in one of the provided ontologies. Thus, a second user-designed ontology with transitory objects was fed to accommodate temporal facts (i.e. the Full(type) fact representing incoming values for user-specified measurements); this ontology was translated into internal object declarations.

In both cases, ontologies were codified in OWL-XML. This constrained subset of RIF-PRD was chosen as one of the possible subsets covering the requirements; other translations may be defined.

Figure 2 shows the ontology used for our specific example in a rough cloud data model; all the ontology entities shown here are linked to the underlying OO data model, so an update on the facts also updates the model. The model shows a typical IaaS architecture with the following characteristics:

- Customers pay the IaaS provider for hosting their services.
- Services consist of a set of VMTypes; each VMType represents an ISO image and a set of hardware requirements (modeled by HWComponent entity) that will be used to generate VMs.
- Each VM contains a set of hardware elements, whose state is represented with HwState entity.
- VMs are deployed in DataCenters (the place where the physical host machines are placed). Each datacenter has a limited amount of each kind of resource: CPU, disk, memory, etc. (represented by the Resource entity);
- The overall performance of the service is measured with user-defined Key Performance Indicators (KPI).

A rule is composed by a formula expressing the condition being observed and an action block expressing the actions to be taken when the condition is met in order to enforce the rule. Formulas, in turn, can be expressed as atomic members, existential quantifiers, logical “ands”, “ors”, or negations. Table I shows the correspondence between language constructs used during translation. As can be observed, these basic building blocks find a way to be expressed in RIF and, later, mapped to a RE-specific construct such Drools or Jess rules.

A particular problem was found with the construct no-loop of Drools and Jess, as there is no way of representing it on RIF. The decision was take for adding this constraint to every rule, based on the way the engine will be used on the cloud: it will be continuously called, so the assertion of new facts may lead to recursion in subsequent execution stages, but not in the same one it was asserted on.

Having an ECA/inference rules portable language and appropriate mapping for common RE-linked languages assesses the feasibility of this approach for the Cloud and that RE-induced constraints are also being met by our selected RIF subset. These will be further demonstrated by means of a Use Case in the following section.

A. RIF-expressed Rules in OVF

As mentioned in the introduction, Claudia uses OVF as service definition language. Therefore, a mechanism to include RIF-based rules to describe the behavior of the service along with its other properties in OVF (architecture, hardware requirements, customization, etc.) needs to be provided.

OVF is XML-based and the main extension mechanism is based on the definition of new OVF sections. These can be used in service definitions along with the ones already defined in the standard at [17]. An OVF section is basically an XML element with a well-defined syntax (specified in XSD).

In order to include RIF rules, we have defined the GovernanceRuleSection, which precise specification in XSD is shown in Listing 1. The section is basically a container for RIF elements defined in the RIF namespace, in particular the ones with types rif:IRIMETA, rif:directive and rif:payload. Note that rif:Document element cannot be directly included, given this element is not associated to any named type in the RIF XSD. However, it can be observed in the RIF XSD that the rif:Document element is defined as a sequence of rif:IRIMETA, rif:directive and rif:payload exactly in the same way that in Listing 1, so at the end, there is a bijective mapping between GovernanceRuleSections and rif:Documents. Finally, note that the ruleId attribute can be used to specify rule precedence order among several GovernanceRuleSections within the same OVF descriptor. An example of GovernanceRuleSection is shown in Listing 2.

V. USE CASE

For this use case, we propose a hypothetical situation where two IaaS providers are using two different production REs: JBoss Drools (Java based) and Jess (Lisp variation) to provide an execution environment for the same service of a service provider. In both cases, the same service descriptor file will
<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>RIF</th>
<th>Drools</th>
<th>Jess</th>
</tr>
</thead>
<tbody>
<tr>
<td>RULE</td>
<td>Conditional part of the rule, should accept logical operators grouping atoms, frames and restrictions.</td>
<td>&lt;Implies&gt;&lt;if&gt; FORMULA &lt;/if&gt;&lt;then&gt; ACTION_BLOCK &lt;/then&gt;&lt;/Implies&gt;</td>
<td>(defrule FORMULA -&gt; ACTION_BLOCK)</td>
<td>And and or same as for Drools; negation in polish notation.</td>
</tr>
<tr>
<td>FORMULA</td>
<td>Conditional part of the rule, should accept logical operators grouping atoms, frames and restrictions.</td>
<td>(defrule FORMULA -&gt; ACTION_BLOCK)</td>
<td>(defrule FORMULA -&gt; ACTION_BLOCK)</td>
<td>And and or same as for Drools; negation in polish notation.</td>
</tr>
<tr>
<td>ATOMIC</td>
<td>An atomic member of a formula.</td>
<td>(exists FORMULA)</td>
<td>(exists FORMULA)</td>
<td></td>
</tr>
<tr>
<td>EXISTS</td>
<td>Existential quantifier.</td>
<td>(exists FORMULA)</td>
<td>(exists FORMULA)</td>
<td></td>
</tr>
<tr>
<td>MEMBER</td>
<td>Predicate that checks whether a variable is a member of a class or not.</td>
<td>(name(attr==value, *)</td>
<td>(name(slot value, *))</td>
<td></td>
</tr>
<tr>
<td>FRAME</td>
<td>Slotted fact, that will correspond to objects in the final data model, representing its attributes as slots.</td>
<td>(name(attr==value, *)</td>
<td>(name(slot value, *))</td>
<td></td>
</tr>
<tr>
<td>EXTERNAL/ NON INLINE RESTRICTION</td>
<td>Comparison restrictions on constants and variables bound by the previous matches. Non-inline are the ones who doesn’t appear inside other elements.</td>
<td>(test (expression))</td>
<td>(test (expression))</td>
<td></td>
</tr>
<tr>
<td>ASSERT</td>
<td>Asserts a fact to the working memory.</td>
<td>insert(new object())</td>
<td>(assert (fact))</td>
<td></td>
</tr>
<tr>
<td>MODIFY</td>
<td>Modifies a fact from the working memory.</td>
<td>update(object)</td>
<td>update(object)</td>
<td></td>
</tr>
<tr>
<td>RETRACT</td>
<td>Remove the fact from the working memory.</td>
<td>retract(object)</td>
<td>retract(object)</td>
<td></td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Executes an externally defined function or method.</td>
<td>Being OO, actions should be executed by invoking methods on any object imported into the rule package (with the typical Java syntax).</td>
<td>Object instances can be created using operator new, and its methods called using the function call on the bound variable.</td>
<td></td>
</tr>
</tbody>
</table>
The first rule checks if any measurement has gone over its permitted limit, raising a new Full(type) fact, indicating that there is a lack of resources.

Two other rules determine when a new virtual machine (VM) of a certain type should be created due to a lack of a specific resource. Two kinds of VMs are supposed to exist in this example: BigDiskVM and BigBandwidthVM. A new VM will be created if the VM limit has not been reached for this type and there is a lack of resources that can be solved with a new machine of this type. In this case, the Full(type) should also be retracted and the VM count modified.

- Once a VM has been created, a rule exists to determine for each DataCenter, if it is suitable to deploy the VM. This rule has to have priority over the next.
- The last rule chooses one of the suitable datacenters for deploying the VM on it.

The code snippet in Listing 3 shows the rules used for the experiments (expressed in informal programming logic notation, for the sake of brevity):

**B. Experimental Support**

The experiment was designed with the RIF-PRD and OWL-XML files as an input in order to complete two goals:

- first, to translate the RIF-coded rules to the specific rule languages accepted by each RE, thus proving interoperability;
- and second, to execute both generated rule sets under the same set of facts, to show how they produce the same results (i.e. they are semantically equivalent).

Having an ontology handy eases direct fact assertion as objects of the cloud data model (both REs provide mechanisms to inject objects in their fact base). Thus, ontology-aware RIF to Drools and RIF to Jess translators have been implemented following the mapping rules and constraints presented above.
The asserted objects were translated into the appropriate object import statements by each translator.

Since an important subset of the Cloud required rules are ECA, the consequent part of the rules is supposed to need the interaction with the real system, a last piece of data was provided to the REs: a set of “action classes” to be instantiated at runtime, which methods could be used to fire actions on the system, out of the RE. These actions were codified in RIF as external function calls, and then used as method calls on a globally defined instance in Drools and Jess. For the experiments, a single Actions class was created with one method, createVM(type), only to show the link with the RE actually works, but any number of them could be defined.

The experiment followed this steps:

- First, the translator component is fed with the RIF document and the correspondent ontologies.
- Then, the translator translates the documents into both languages, Drools and Jess.
- Once translated, the fact base is externally fed with KPI facts so to make the scalability rules fire.
- The two rule engines are activated to execute the translated rules. As a result of this execution, both decide to create a BigDiskVM on the Spanish site.

The results of the inference (as can be shown in Figure 4) were the following executions:

1) Assert the fact: Full(type -¿ QueueLength).
2) Call the external method createVM on class actions; retract the Full(type) fact; and raise the VM Count.
3) Mark the unsuitable datacenters.
4) Deploy the generated VM in the selected datacenter.

This proof of concept was put into practice (along with the OVF RIF integration) in a full working project with the participation of the Claudia system as part of the EU FP7 project Reservoir.

VI. DISCUSSION AND CONCLUSIONS

The current wealth of efforts on rule notations can be grouped in the several classes. In this work we have offered a categorization of these and presented a model enabling their interoperability. We have detected that there is a mapping problem from more abstract/general/graphic languages to the concrete RE-used languages. Some rule interchange formats are handy to help mitigate this problem, but they often prove too extensive for the simple type of rules required for controlling cloud applications. We have analyzed these requirements arisen from the type of rules typically employed for governing the Cloud. Then, we have selected the most appropriate interchange language and build generic RIF to widespread RE languages translators that boost interoperability among clouds and service definition portability. Appropriate mapping from/to

(see Section IV).
RIF to/from RE-specific languages has been established to build ECA/inference rules for cloud applications. Also, semantic search and validation are enabled by the inclusion of a cloud data model ontology. Finally, this work has analyzed the suitability of RIF-expressed rules into to-be standards for cloud service definition such as OVF.

Cloud application management has two key requirements: interoperability between different RE providers (in other words, portability of the application behavioral rules); and the ability to reason and interact with the underlying OO data model. While the second requisite is common among rule language providers, only RuleML and RIF have been designed with the objective of interoperability. Having very similar features, RIF was chosen for its W3C recommendation, that may let it lead the rule language standardization efforts. Also, RIF was superior to RuleML with regards to its ability to support ontologies of the underlying cloud data model.

As it has been shown in the experiments, RIF-PRD offers a slot-oriented, FOL-capable production rule language that meets the minimum requirements to bring the power of programming logic to the service behavior specification on IaaS clouds. In the use case, a subset of RID-PRD capable of meeting the expected requirements was presented, and its interoperability capabilities demonstrated showing a complete match with constructs on two low level rule languages. Then, during the experiments, a translator from this subset to each language was built; an example of RIF codified rules was used then to prove the two rule set files generated could be executed against a common fact base, yielding the same result (so showing its semantic equivalence).

This work has only dealt with the mapping from RIF to other RE-attached languages (lower-level mapping). More work is needed to help bridging the whole gap of rule languages from abstract ones to RIF. Since RIF is aimed at portability an idea possibility is using RIF for high level purposes as well. However, commercial interests, the ease of use of natural language or graphic tools, etc. will make the higher level mapping (from these high-end languages to RIF) needed. In this regard, being RIF and RuleML W3C-derived, both have joined forces and they are working on a common kernel of both languages called Dlex which makes higher-level mapping more straightforward. This is so since some languages above are based on (SWRL) or are easily translatable to (POSL) RuleML. Another challenge the future work will face is the scalability aspect of the proposed solution, when multiple services may be deployed over multiple clouds, potentially interacting between them (where isolation problems may also rise).

As it has been shown, the use of a production rule languages can help to improve the control over a service behavior, in a way that is: 1) expressive enough for FOL while retaining the ability to interact with data models and internal functions; 2) independent of the underlying implementation: using a standard interoperable language, based on XML leads to a service behavior definition that can be shared among different IaaS providers supporting rule languages, with independence of the RE implementations they use in their own codes; 3) reconfigurable at runtime, as the rule set, as well as the actions provided, can be changed on runtime without needing to undeploy or stop the service.

The mapping mechanism herein proposed is limited to two of the most widespread open rule languages, but does not take importance references such as IBM’s ILOG into account. This is so for the sake of repeatability and does not reduce the generality of the proposed mapping mechanism since OO-based ontology-powered ECA/inference rules constructs can also be mapped to ILOG by means of a specific translator.

The Cloud is still in its infancy and more type of rules will be included as the variety of actions triggered when a condition is met is also expanded. The proposed system is general enough but it can still be extended to support new constructs as these are required by the users.

ACKNOWLEDGMENT

The authors would like to thank Telefonica’s team for their huge effort and professionalism beyond their duty. The authors would like to emphasize that this work has been fully developed in Telefonica within the Real Elastic Cloud (REC) initiative.

REFERENCES