Unified Representation and Reuse of Federated Cloud Resources Configuration Knowledge

Denis Weerasiri[1] Boualem Benatallah[1] Jian Yang[2]

[1] University of New South Wales, Australia
{denisw,boualem}@cse.unsw.edu.au
[2] Macquarie University, Australia
jian.yang@mq.edu.au

Technical Report
UNSW-CSE-TR-201411
March 2014

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Current cloud resource delivery models enforce cloud resource consumers to bear the burden of leveraging existing cloud resources configuration management knowledge to satisfy consumers' federated cloud application and resource requirements. Because the support offered by current cloud resources configuration management techniques is mostly limited to segregated cloud infrastructures or platform functionalities, which prevent any coordinated combination of on-premise and off-premise applications, and resources. In this paper, we propose an embryonic data model for unified cloud resources configuration knowledge representations. Also we propose a rule based recommender system, which allows consumers to declaratively specify requirements and get recommendations of configuration management knowledge that satisfies the given requirements. We implemented a proof-of-concept prototype to test our approach.

1 Introduction

The Cloud computing paradigm transforms applications, hardware, and software infrastructures into virtualized and dynamically scalable resources. These resources are available to users in the form of cloud services(Infrastructure as a Service, Platform as a Service, and Software as a Service) which are delivered on demand over a network. These services provide service interface layers that shift the focus from underlying infrastructure and operations such that cloud resource consumers are responsible to deploy and manage a configuration of cloud resources based on consumers' application and resource requirements[1, 2, 21].

Cloud computing is evolving in the form of both public (deployed by IT organizations) and private clouds (usually deployed behind a company firewall). A third option, a hybrid or federated cloud, is now emerging, where computing resources drawn from a subset of private and public clouds combined at the behest of its users. It is imperative that the federation of clouds leads to a unified model which represents a single cloud of multiple cloud platforms that can be used as needed. Thus the cloud federation requires the creation of an agile cloud computing environment, in which cloud capabilities can be procured, configured, deployed, and managed on demand by consumers, regardless of whether cloud capabilities are private or public.

To describe a cloud resources configuration, consumers firstly interpret their resource requirements (e.g., I need a project management service for 50 users from May to July) and application requirements (e.g., I want my logistics application to aggregate together order management, inventory and payroll services from diverse providers that support OAuth 2.0 protocol) in the form of a Cloud Resources Configuration Description(CRCD). CRCDs specify expected resources, service levels, geo-location etc., of cloud resources configurations such that providers can interpret CRCDs and deploy applications. To implement a CRCD, consumers should understand the configuration description language of a cloud resource provider. This language describes the available resources and mechanisms for selecting particular configurations of those resources. Furthermore to deploy and manage the described configuration, consumers need to understand configuration management interfaces and processes of cloud resource providers. When one provider cannot satisfy consumers' requirements, consumers must understand configuration description languages, configuration management interfaces, and processes of multiple providers to manage a federated cloud resources configuration. For an example, two CRCDs and two deployment processes are required to deploy a federated Virtual Machine(VM) over two providers (Amazon-EC2 and Rackspace). Furthermore, to change a configuration (e.g., increasing memory from 1GB to 4GB), consumers should map the change into relevant modifications in both CRCDs and trigger relevant configuration management processes. In a summary, consumers need to incoherently manage component cloud resources to manage a federated cloud resources configuration.

Based on above observations, we concluded that existing configuration management techniques (1) are rarely transparent and adaptive to federated clouds; (2) enforce consumers to gain expertise in multiple configuration management knowledge domains; and (3) lead to increased management costs and potential vendor lock-in as resources management of a new provider requires a different expertise.

Hence an important research question is how to leverage existing cloud resources configuration knowledge to support individualized application requirements on a federated cloud in a systematic manner. The challenges involved in solving this research problem are (1) unified modeling of heterogeneous cloud resources configuration management knowledge; (2) satisfying individualized application and resource requirements with existing configuration knowledge; and (3) capturing, customizing, and reusing existing configuration knowledge.

Solving this research problem is beneficial to any consumer. Because regardless of how different cloud resources (IaaS, PaaS and SaaS) are or how such resources are deployed (public and private), almost every cloud resources configuration expects to satisfy individualized application requirements. Furthermore consolidating and sharing cloud resources configuration management knowledge are advantageous for consumers who do not have much skills in CRCD development and management. Because consumers can leverage existing configuration management knowledge in a unified manner. Also consumers are shielded from provider specific, low level, complex, and heterogeneous configuration management interfaces, and technologies. Based on above analyses, we propose the following contributions.

1. A unified cloud resources configuration knowledge representation model[1]

2. A rule based recommender system. This system leverages the cloud resources configuration knowledge representation model to support individualized application requirements. In essence the recommender system uses a consumer specified application requirement context (e.g., intended task, deployment scenario) to query a "cloud resources configuration knowledge base(KB)", which returns configuration knowledge that satisfies the given context

3. An incremental knowledge acquisition technique, starting with an empty KB which is gradually built up

We also discuss a proof-of-concept prototype implementation and verify our proposed approach based on 3 use cases, which involve 8 different cloud resources.

The paper is structured as follows. Section 2 explains the cloud resources configuration knowledge representation model. Section 3 elaborates the rule based recommender system and incremental knowledge acquisition technique. Section 4 explains the implementation of our solution, followed by related work (Section 5) and the conclusion including future work (Section 6).

2 Unified Knowledge Representation for Cloud Resources Configurations

One of the main concepts of our research work is the unified knowledge representation model for cloud resources configurations. Rather than treating cloud resources as isolated entities, we describe a unified and hierarchical representation model for the logical organization of cloud resources configurations. This model is based on the entity-relationship(ER) modeling. Each entity models

[1]In this paper "cloud resources configuration knowledge representation model" and "configuration knowledge representation model" are used interchangeably.

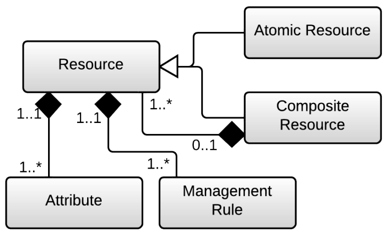

Figure 2.1: UML class diagram of the Configuration Knowledge Representation Model

reusable and customizable configuration knowledge of a desired cloud resource (e.g., Virtual Machine(VM), key-value database, software development platform) without referring to any resource provider. The configuration knowledge captured is modeled as a set of *Attribute*s and *Management Rule*s. A configuration of cloud resources can be composed of one or more component cloud resources (e.g., a web application server, composed of an application runtime and persistent storage). These composites are modeled using relationships among the component resource entities. We decided to follow the ER modeling as it explicitly supports to specify relationships among component resources and configuration descriptions of each component resource. The rest of the section explains how cloud resources and configurations of cloud resources are modeled.

Resource

The *Resource* entity is the base entity, which can be specialized into either *Atomic Resource* or *Composite Resource*. A set of *Attribute*s and *Management Rule*s are associated with each *Resource* to describe capabilities of a cloud resource. *Attribute*s model capabilities in forms of key-value pairs. *Management Rule*s will be explained next.

Management Rules

A cloud resources configuration is open for dynamic state changes during its lifetime. Consumers can invoke configuration management processes (e.g., deploy a MySQL database and increase its capacity up to 10GB) which change the state. Alternatively other events within the external environment can trigger state changes of a cloud resources configuration (e.g., temporal events, application workload metric change). Therefore a configuration knowledge representation model should represent how it modifies its state when such events occur. To capture and react to events, each *Resource* includes a set of *Management Rule*s. *Management Rule*s follow active rule model[4]. Each rule consists of an event name and optional handler. The event name specifies the signal that triggers the invocation of the rule. The handler represents how the *Resource* should behave when the certain event occur. For an example, the handler, which associates with "DEPLOY" event, is triggered when consumers request to deploy a particular configuration representation model. In the current implementation we only support "DEPLOY" event.

Atomic Resource

An *Atomic Resource* represents configuration knowledge of a cloud resource that does not rely on any other *Resource*. In other words, an *Atomic Resource* is an indivisible resource into component resources from the perspective of resource curators (those that primarily add/maintain configuration knowledge representation models in the KB). For an example, a VM with 4GB RAM and 4GHz processing power, can be modeled as an *Atomic Resource* with two *Attributes* for the memory and processing power. *Atomic Resources* act as primary building blocks of *Composite Resources*.

Composite Resource

A *Composite Resource* is an umbrella structure that brings together other *Atomic* and *Composite Resources* to model a configuration, which is composed with multiple cloud resources. An example of *Composite Resource* would be an E-Learning platform that consists of an artifact management service and student identity management service to support 100 students. The cloud resources brought together by a *Composite Resource* are referred to as its *component resources*. A *Composite Resource* can be constructed by associating with other available *Resources*. To represent these associations in our initial data model, a *Composite Resource* consists of a list of references to other*Atomic* and *Composite Resources*, additionally to what is available in a *Resource*. But *Composite Resources* are not allowed to have cyclic references. This list specifies all the components *Resources*, required to construct this particular composite cloud resource. Unlike in *Atomic Resources*, handlers of a *Composite Resource* might trigger other handlers of *component resources* as well.

3 KB based Reuse of Configuration Knowledge

The other concept (apart from the unified knowledge representation model for cloud resources configuration) of our research work is the reuse of existing configuration knowledge for individualized application and resource requirements. We propose a rule based recommender system to implement this concept. Our rule based recommender system intends to suggest configuration knowledge, required for consumers during cloud resources configuration management processes (e.g., deployment and configuration parameter modification). These suggestions are generated based on a consumer specified context (e.g., intended task and deployment scenario). This context represents an individualized application or resource requirement. The suggested configuration knowledge includes all necessary information and instructions, required to deploy a cloud resources configuration, which satisfies the context description. Our system derives suggestions from configuration knowledge artifacts (e.g., executable deployment scripts, packaged virtual appliances) that were created for similar contexts in the past. Consumers can accept or modify recommended configuration knowledge artifacts according to consumers' requirements. Alternatively consumers can reject the recommendation, and create a new configuration knowledge artifact from scratch. Once such modifications are completed, the recommender system translates those modifications into *Recommendation Rules* with help of

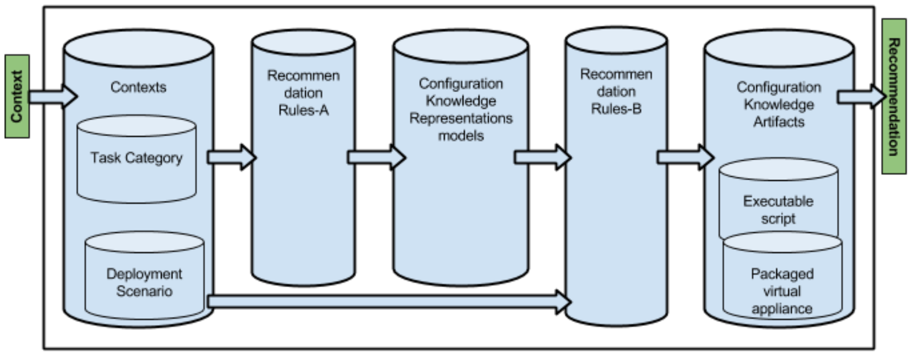

Figure 3.1: Rule based Recommender System Architecture

consumers and make available new recommendations for future consumer requests.

In this section we first introduce *Recommendation Rule*s of the recommender system, followed by the construction, origin and evolution of *Recommendation Rule*s.

3.1 Recommendation Rules

Our recommender system maintains a cloud resources configuration knowledge base(KB) which stores contexts, configuration knowledge representation models, and configuration knowledge artifacts. *Recommendation Rule*s maintain associations between those items in the KB as shown in Figure 3.2 and 3.3. *Recommendation Rule*s consist of contexts (when does the rule apply) and conclusions (what should be recommended when the rule is activated).

Once a consumer specifies a context description, our recommender system generates a recommendation using a two-step process. First, the consumer queries the "Task Category" database (See Figure 3.1) and specifies consumer's intended task (e.g., web application development, storage management, identity management, customer relationship management, etc.,). As the result, the recommender system suggests an instance of an available configuration knowledge representation model that satisfies the intended task. Second, the consumer queries the "Deployment Scenario" database (See Figure 3.1) and specify the preferred deployment model (e.g., local server, private cloud, or Amazon-EC2). As the result, the recommender system suggests a configuration knowledge artifact that satisfies the specified deployment scenario of the previously recommended configuration knowledge representation model in the 1st step. This configuration knowledge artifact is input to a specific provider's deployment service to provision a cloud resources configuration, which satisfies the consumer's requirements.

In a summary, the recommendation in the 1st step is input as a context parameter to the 2nd step. Both steps generate recommendations using two different *Recommendation Rule* types as follows.

1. Recommendation Rules-A - Rules that map a task category to a cloud resources configuration knowledge representation model (Figure 3.2). Cloud

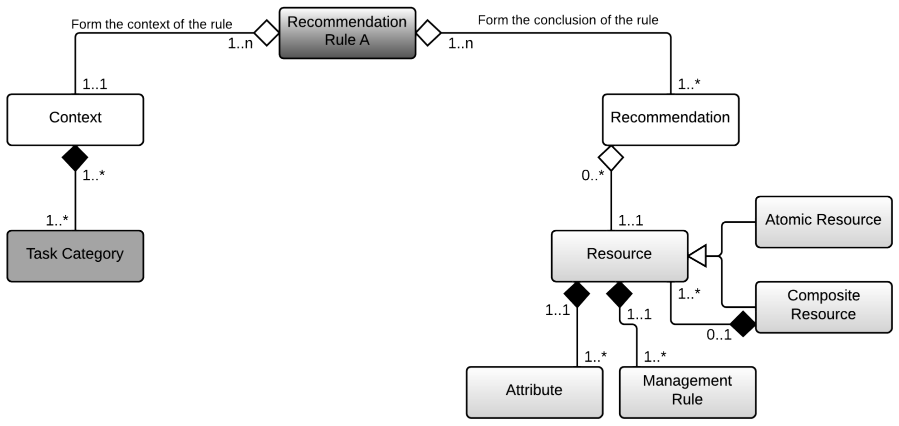

Figure 3.2: UML class diagram of Recommendation Rules-A component

resource consumers or curators of configuration knowledge representation models primarily add/maintain these rules.

2. Recommendation Rules-B - Rules that map a configuration knowledge representation model and deployment scenario to a configuration knowledge artifact (Figure 3.3). Configuration knowledge artifact developers or cloud resource providers primarily add/maintain these rules.

Contexts of Rules

The recommender system maintains "Contexts" data (See Figure 3.1) for task categories and deployment scenarios. The "Task Category" and "Deployment Scenario" databases intend to capture meta-data and common information about classes of similar resource and application requirements. These databases allow cloud resource consumers to reuse and customize shared context knowledge. On the other hand associating configuration knowledge representation models with relevant task categories can effectively segment those models based on potentially satisfying tasks.

The "Contexts" database is implemented as a hierarchical structure of context entities. Therefore task categories can be associated from more generic (e.g., "Application Development", "Storage management") to more specialized entities (e.g., "Java based Web Application Development", "Relational Database services"). Similarly deployment scenarios can be associated from more generic (e.g., "public cloud deployment", "private cloud deployment") to more specialized entities (e.g., "VMWare vSphere Hypervisor 5.5", "Amazon-EC2", "Windows Azure"). This hierarchical structure helps consumers to query "Contexts" data and find either equal or approximately equal (but more generalized) contexts that satisfy consumers' requirements.

Left hand side of *Recommendation Rule*s in the UML class diagrams depicts the context of rules. These two rule types consist of different types of contexts. Rules-A consists of a context that describes the intended task of the consumer (e.g., RuleID 1 in Table 3.1).

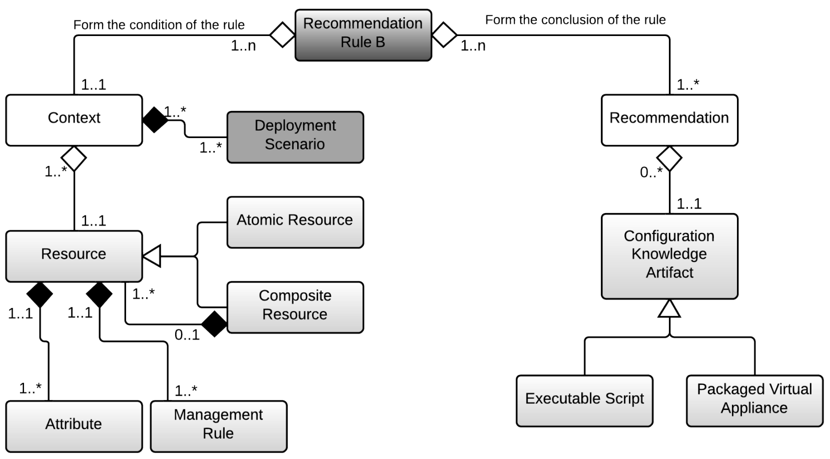

Figure 3.3: UML class diagram of Recommendation Rules-B component

Table 3.1: Examples of Reccomendation Rule Types

RuleID	**Context**	**Recommendation**
1	Task_Category="Tomcat based web application"	KRM_ID*= "15"
2	KRM_ID*= "15" **AND** Deployment_Scenario= "EC2"	KA_ID**= "637"

*KRM_ID stands for Knowledge Representation Model ID

**KA_ID stands for Knowledge Artifact ID

Rules-B consists of a context that describes the intended target deployment of a selected configuration knowledge representation model (e.g., RuleID 2 in Table 3.1).

More design level details on how these rules are originated, processed and evolved are explained later in Section 3.

Conclusions of Rules

The right hand side of *Recommendation Rule*s in the UML class diagrams depicts components, which construct the conclusion of a rule (See Figure 3.2 and 3.3). Both rule types have different conclusions based on the rule's context. Rules-A recommends a configuration knowledge representation model which is described in detail in Section 2.

Rules-B suggests a configuration knowledge artifact, which can be classified as follows.

1. An executable script : describes a deployment process in a cloud resources configuration description language (e.g., Puppet[1], dotCloud[2], or Ansibleworks[3]); or a script implemented in a low level language (e.g., python, shell scripts, or even WS-BPEL). Hence cloud resource consumers can execute those scripts to deploy cloud resources configurations.

[1]`http://docs.puppetlabs.com/puppet/3/reference/lang_summary.html`
[2]`https://www.dotcloud.com/`
[3]`http://www.ansibleworks.com/`

2. A packaged virtual appliance : a virtual appliance packaged in a standard format (e.g., OVF (Open Virtualization Format)). Consumers can download the suggested package and import it into a supported cloud resource provider (e.g., VMWare Workstation, Amazon-EC2).

Once the recommender system suggests a configuration knowledge artifact, consumers can deploy the configuration knowledge artifact via a specific provider's configuration management service. Consumers can manage the deployed cloud resources configuration then. We consider the management aspect of federated cloud resources configurations as a future work.

In the next sections we explain how *Recommendation Rule*s are originated, processed and evolved.

3.2 Reuse of Configuration Knowledge

In order to increment and reuse existing knowledge in KB, we use a knowledge acquisition and management technique called Ripple Down Rules (RDR)[6]. RDR technique has been successfully implemented in many domains (e.g., natural language processing, clinical pathology reports, call centers, database cleansing, UI artifact reuse and soccer simulations). But to the best of our knowledge, there has been no attempt to adapt RDR to the domain of cloud resources management knowledge reuse.

There are different variations of RDR such as Single-Conclusion RDR (SCRDR), Multiple-Conclusion RDR (MCRDR), and Collaboration RDR. In our recommender system we implement SCRDR technique, which allows only conclusion for a given context. SCRDR empower the reusability of existing configuration knowledge representation models and configuration knowledge artifacts. Also SCRDR increment knowledge by integrating new rules to the existing KB. As a future work we will replace our SCRDR implementation with MCRDR which allow multiple conclusions and rule modifications by adding exception rules. MCRDR will enhance the productivity of our recommender system.

Figure 3.4 shows examples of two types of *Recommendation Rule* trees in the KB. Rule A0 and B0 contain the default conclusion("unknown"). The recommender system suggests the default conclusion, when the input context is not specified. Thus in Rules-A tree, the inference engine triggers the default conclusion, when the task category is not defined in input contexts. In Rules-B tree, the engine triggers the default conclusion, when the deployment scenario or configuration knowledge representation model are not specified. The KB depicts "except" (true) branches and "if not" (false) branches. When consumers input a context to the recommender system, the inference engine starts querying the relevant *Recommendation Rule* tree. Starting from the root node, the engine checks whether the next rule node is true or false by comparing the context of each rule node with the consumer specified context. This task is carried out repeatedly until the inference engine cannot proceed to find any more true nodes. The conclusion of the last true node is returned back to the consumer.

To give an example, a curator of our KB may want to model a cloud resources configuration for a "Java based Web application development". But assume our KB does not contain this task category at this moment. That means "Rule A2" does not exist in the Rule-A tree (see Figure 3.4). Hence the curator queries the KB and find a cloud resources configuration knowledge

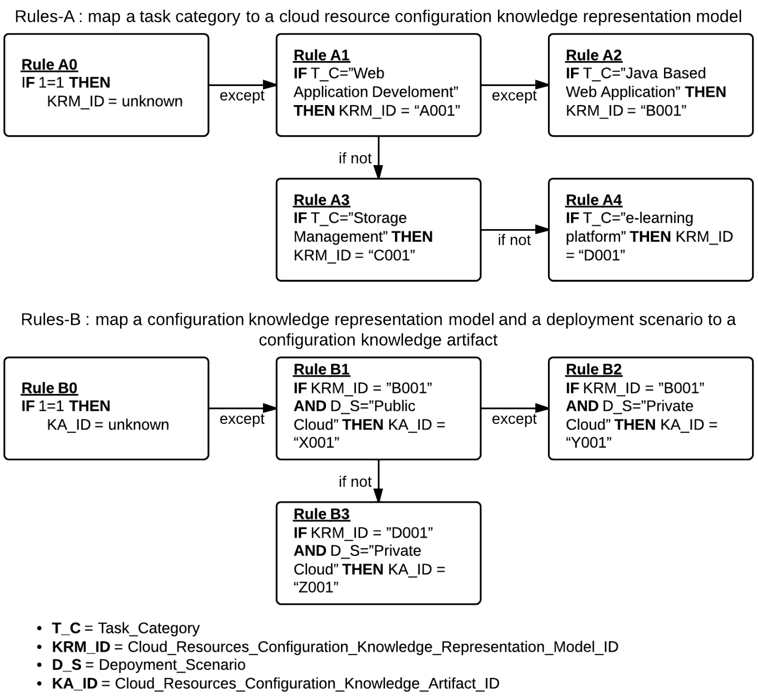
Rules-A : map a task category to a cloud resource configuration knowledge representation model
Rule A0
IF 1=1 THEN KRM_ID = unknown
except
Rule A1
IF T_C="Web Application Develoment" THEN KRM_ID = "A001"
except
Rule A2
IF T_C="Java Based Web Application" THEN KRM_ID = "B001"
if not
Rule A3
IF T_C="Storage Management" THEN KRM_ID = "C001"
if not
Rule A4
IF T_C="e-learning platform" THEN KRM_ID = "D001"
Rules-B : map a configuration knowledge representation model and a deployment scenario to a configuration knowledge artifact
Rule B0
IF 1=1 THEN KA_ID = unknown
except
Rule B1
IF KRM_ID = "B001" AND D_S="Public Cloud" THEN KA_ID = "X001"
except
Rule B2
IF KRM_ID = "B001" AND D_S="Private Cloud" THEN KA_ID = "Y001"
if not
Rule B3
IF KRM_ID = "D001" AND D_S="Private Cloud" THEN KA_ID = "Z001"
T_C = Task_Category
KRM_ID = Cloud_Resources_Configuration_Knowledge_Representation_Model_ID
D_S = Depoyment_Scenario
KA_ID = Cloud_Resources_Configuration_Knowledge_Artifact_ID

Figure 3.4: Example of Recommendation Rule Trees

representation model that is associated with "Task_Category"="Web Application Development" ("Rule A1"). But the curator cannot find an "except" rule that originated from "Rule A1". Therefore the curator firstly composes and registers a configuration knowledge representation model in the KB. This model describes the configuration with required component resources for the Java based Web application development environment. The curator then registers an "except" rule ("Rule A2") under "Rule A1", and refer the composed configuration knowledge representation model as the recommendation.

In another scenario, a cloud resource consumer may need to deploy an E-Learning platform on a "VMWare vSphere Hypervisor"(Private Cloud). Therefore the curator specifies the context as "Task_Category" = "E-Learning Platform". The inference engine checks in Rules-A tree for a rule whose "Task_Category" is equal to "E-Learning Platform". Once the engine finds the particular rule ("Rule A4"), the conclution of that rule is returned back to the consumer. The consumer then specifies a "Deployment_Scenario" for the recommended configuration knowledge representation model. The inference engine checks along the "if not" path way in Rules-B tree, and realize the last rule node that is set to true is "Rule B3". Hence the conclusion (e.g., a download link to a pre built VMWare supported virtual appliance that includes Moodle; an E-Learning platform) of "Rule B3" is recommended back to the consumer.

3.3 Knowledge Acquisition Process

The KB, empowered by SCRDR, incrementally acquires configuration knowledge in forms of rules. Any change in contexts, configuration knowledge representation models, or configuration knowledge artifacts activates an update to the KB. Following cases create new *Recommendation Rule*s in our KB.

1. A new configuration knowledge representation model or a configuration knowledge artifact is registered on an existing context in the KB

2. A configuration knowledge representation model or a configuration knowledge artifact is registered or modified on a non-existing context in the KB

Case 1 - Users (e.g., curators of the KB), who expect to register a configuration knowledge representation model, can register a rule under Rules-A tree by specifying an existing "Task_Category" as the context and referring the new configuration knowledge representation model as the conclusion. Users (e.g., cloud resource providers or configuration knowledge artifact developers), who expect to register a configuration knowledge artifact, can register a new rule under Rules-B tree by specifying an existing configuration knowledge representation model and "Deployment_Scenario" as the context and referring the new configuration knowledge artifact as the conclusion.

Case 2 - Case 2 is triggered in two scenarios: (1) curators need to register, or modify a configuration knowledge representation model; or (2) users (e.g., cloud resource providers, or configuration knowledge artifact developers) expect to register, or modify configuration knowledge artifacts. In both scenarios, the expected "Task_Category" or "Deployment_Scenario" does not exist in "Contexts" database (see Figure 3.1). Therefore users firstly register relevant entries in "Contexts" database. When a new context is a specialization to an existing context, the new context is positioned accordingly in the hierarchical structure. Next, an "except" rule is registered in Rules-A or Rules-B tree accordingly. Alternatively a new rule is registered in a "if not" branch in Rules-A or Rules-B tree.

These processes allow the evolution of our SCRDR based KB over time. Our approach makes more productive suggestions when there are enough rules. So our approach needs little consumer effort to specify contexts of rules when rules are being created.

In a summary, our rule based recommender system lets consumers to focus on their application and resource requirements, while the system shields consumers from technical complexity of federated cloud service solutions. We argue that a framework, which decouples a cloud resource requirement specification from underlying resource and service configuration needs, caters for a flexible characterization and planning of resource needs over time.

4 Implementation

To evaluate our approach, we implemented a proof-of-concept prototype. To test the prototype, we also developed test cases based on 3 practical user scenarios. These scenarios were based on a SENG1031[1] lecturer's application requirements

[1]http://www.cse.unsw.edu.au/~se1031

(e.g., a pre-configured development environment packaged as a virtual appliance for students, multi tenant software project management environment for all student groups) which involved a composition of 8 different cloud services (Heroku, Google-Drive, GitHub, LucidChart, PivotalTracker, Facebook developer API, Twitter developer API, and Amazon-S3).

We implemented our recommender system as a proof-of-concept prototype which includes two main components: an in-memory "Contexts" database and SCRDR system using Java. These components collectively provide three features: recommending configuration knowledge representation models for a given task category; recommending configuration knowledge artifacts for a given deployment scenario and configuration knowledge representation model; and acquiring, and evolving configuration management knowledge.

To generate recommendations, there should be some rules in the recommender system. Thus we constructed a context hierarchy and added rules following SCRDR techniques as prerequisites. We added a set of task categories, deployment scenarios and configuration knowledge representation models in our "Contexts" database. We modeled each *Atomic Resource* and modeled *Composite Resource*s by reusing previously modeled *Atomic Resource*s. *Recommendation Rule*s were registered in the KB which allowed lecturers to query and get recommendations of cloud resources configurations for given task categories. Lecturers selected an available task category (e.g., SENG1031-Software Engineering Workshop: server side appliances, Software development and management appliances for COMP9323). Selected context information was sent to the inference engine. The engine matched the specified context with contexts of available rules and returned a configuration knowledge representation model as the recommendation. Lecturers then specified the deployment scenario from the "Contexts" database. The chosen deployment scenario was sent back to the recommender system and the system returned a configuration knowledge artifact as the recommendation. In our initial prototype recommendations are URLs. Hence consumers can access and download the configuration knowledge artifact and deploy it. Alternatively consumers can modify that artifact and add a new rule that maps the modified artifact to a context in "Contexts" database.

5 Related Work and Discussion

In this section, we review and differentiate different approaches in research related to the cloud resources configuration management.

Some Software Configuration Management(SCM) solutions (e.g., Puppet, Opscode Chef, and SmartFrog) and some other research initiatives automate behaviors and capabilities of virtual machines(VM) in an cloud infrastructure[5, 11, 16]. Ubuntu Juju and Amazon OpsWorks automate behaviors and capabilities of cloud services, which may consist of several component cloud resources together. Hence those SCM solutions are capable of encapsulating a complete service by defining how each component resource should be constructed and managed. All these solutions use script-based and procedural programming approaches for configuration management and automation. Those approaches are feasible for technical people (e.g., system administrators and devOps) rather non-IT people who want to deploy and manage their own private or public cloud[19].

AutoNetKit provides a template-based configuration description language for large and complex network emulations[17]. AutoNetKit generates vendor specific device configuration knowledge (in forms of scripts) using the template based configuration description. Our approach leverages existing configuration knowledge using a unified configuration knowledge representation model.

A configuration knowledge reuse mechanism is analyzed for individualized use of a software system called MythTV here[15]. In difference to our approach, this research does not follow a unified data model to represent configuration knowledge. A unified meta-model is proposed to capture meta-data (e.g., security, pricing, legal etc.,) of cloud services in[14]. This approach focuses on service discovery aspect, but we focus on configuration management aspect.

There are three main areas of related work: CRCD languages, configuration management and orchestration tools built on top of those languages, and approaches to federated cloud service management.

Cloud Resources Configuration Description(CRCD) Language Classification

CRCD languages can be template-based and model-driven[18].

Template-based approaches aggregate resources from a lower level of the cloud stack and expose the package, along with some configuration options, to a higher layer. The OVF manages resources of various types by describing how a cloud offering is presented and consumed. The service offering is abstracted from the specific type of cloud resource offered. Consumers use service templates to describe what a cloud service can offer. Galán et al. extend OVF standard through configuration parameters for components included in Virtual Machines (VM) as capabilities that should be exposed by an IaaS provider[9].

Model-driven approaches define various models of the software at different levels of the cloud stack; and aim to automate the deployment of abstract, predefined, and composite solutions on cloud infrastructures[5, 10]. Konstantinou at al. proposes a virtual appliance model that treats virtual images as building blocks for IaaS composite solutions[16]. Developers determine deployment-time requirements in a cloud-independent manner using a parameterized deployment plan. At the SaaS level, a model-driven approach can allow users to design applications independent of any platform[13]. This research proposes an application meta-model that enables the modeling of components of an application, component dependencies at both SaaS and IaaS levels[8]. Other approaches describe how to create SaaS application definitions with variability points that can be customized to particular requirements[12].

Configuration Management and Orchestration Tools Classification

Cloud resource providers and 3rd party organizations create tools for CRCD modeling, composition, deployment and management on top of provider specific CRCD languages. These tools automate configuration management and orchestration operations while give operational control to consumers.

1. Some tools are based on unstructured CRCDs
    (a) e.g., To deploy a LucidChart academic runtime instance, consumers need to send an email to the LucidChart support team mentioning

resource requirements

2. Some tools are based on structured CRCDs and provide a web based control interface. For an example, the deployment of web application development runtimes in Nitrous.IO requires users to complete an on-line form.

3. Some tools are based on structured CRCDs and provide a control API. These tools use a script based and procedural CRCD language for configuration management and orchestration (e.g., Puppet, Chef, Ubuntu Juju, Amazon OpsWorks, and RightScale)

Federated Cloud Service Management

The topic of the federation of cloud services has recently gained traction in the scientific community. This research proposes a utility-oriented federation across different cloud providers that support dynamic expansion or contraction of capabilities (storage and computing resources)[3]. This research tackles the problems of cloud intermediation and proposes a cloud broker topology where an intermediate layer federates separate "back end" clouds[7]. Villegas et al. propose a layered architecture mediated by a cloud broker at each cloud stack tier to cater for specific concerns of the parties at that tier[20]. We specifically focus on the configuration management aspect of federated cloud resources.

6 Conclusion and Future Works

In this paper, we have presented a framework to build unified, customizable and reusable configuration knowledge representations for federated cloud resources. The framework consists of (1) a unified configuration knowledge representation model for federated cloud resources; (2) a declarative and context-aware language to specify consumers' application and resource requirements in terms of task categories and deployment scenarios; (3) automatic recommendation of configuration knowledge artifacts for a given context; and (4) an incremental configuration knowledge acquisition mechanism based on a Ripple Down Rules base. To evaluate the feasibility and efficiency of the proposed framework, we implemented our system as a proof of concept prototype. As future work, we plan to (1) extend our declarative requirement specification language with other dimensions (e.g., the expected uptime period of a cloud resources configuration) to bring time-aware configuration management and orchestration capabilities, and (2) extend the unified configuration knowledge representation model to support model-driven configuration management and orchestration techniques.

Bibliography

[1] M. Armbrust and et al. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[2] T. J. Bittman. The road map from virtualization to cloud computing. `https://www.gartner.com/doc/1572031`, March 2011. Accessed: 24/11/2013.

[3] R. Buyya and et al. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[4] S. Ceri, R. Cochrane, and J. Widom. Practical applications of triggers and constraints: Successes and lingering issues. In *Proc. 26th VLDB*, pages 254–262, 2000.

[5] T. C. Chieu and at al. Solution-based deployment of complex application services on a cloud. In *SOLI, 2010 IEEE International Conference on*, pages 282–287. IEEE, 2010.

[6] P. Compton and et al. Ripple down rules: Turning knowledge acquisition into knowledge maintenance. *Artificial Intelligence in Medicine*, 4(6):463–475, 1992.

[7] E. Elmroth and L. Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *2009. GCC'09. Eighth International Conference on*, pages 253–260. IEEE, 2009.

[8] C. Fehling and R. Mietzner. Composite as a service: Cloud application structures, provisioning, and management. *it - Information Technology*, 53(4):188–194, 2011.

[9] F. Galán and et al. Service specification in cloud environments based on extensions to open standards. In *Proceedings of the fourth international ICST conference on communication system software and middleware*, pages 16–25. ACM, 2009.

[10] P. Goldsack and at al. The smartfrog configuration management framework. *ACM SIGOPS Operating Systems Review*, 43(1):16–25, 2009.

[11] P. Goldsack, J. Guijarro, S. Loughran, A. N. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft. The smartfrog configuration management framework. *Operating Systems Review*, 43(1):16–25, 2009.

[12] S. Hagen and A. Kemper. Model-based planning for state-related changes to infrastructure and software as a service instances in large data centers. In *Cloud Computing, 2010 IEEE 3rd International Conference on*, pages 11–18. IEEE, 2010.

[13] M. Hamdaqa, T. Livogiannis, and L. Tahvildari. A reference model for developing cloud applications. In *CLOSER*, pages 98–103, 2011.

[14] P. Hoberg, J. Wollersheim, and H. Krcmar. Service descriptions for cloud services-the customer's perspective. In *Proceedings of ConLife Academic Conference*, 2012.

[15] J. Huh, M. W. Newman, and M. S. Ackerman. Supporting collaborative help for individualized use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3141–3150. ACM, 2011.

[16] A. V. Konstantinou and et al. An architecture for virtual solution composition and deployment in infrastructure clouds. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '09, pages 9–18, New York, NY, USA, 2009. ACM.

[17] H. Nguyen and et al. How to build complex, large-scale emulated networks. *Testbeds and Research Infrastructures. Development of Networks and Communities*, 46:3, 2011.

[18] M. P. Papazoglou. Making business processes compliant to standards and regulations. In *EDOC, 2011 15th IEEE International*, pages 3–13. IEEE, 2011.

[19] B. Satzger and et al. Winds of change: From vendor lock-in to the meta cloud. *Internet Computing, IEEE*, 17(1):69–73, 2013.

[20] D. Villegas and et al. Cloud federation in a layered service model. *J. Comput. Syst. Sci.*, 78(5):1330–1344, Sept. 2012.

[21] L. Wang, R. Ranjan, J. Chen, and B. Benatallah. *Cloud computing: methodology, systems, and applications.* CRC Press, 2012.