2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing

An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery

Amir Vahid Dastjerdi[1], Sayed Gholam Hassan Tabatabaei[2], and Rajkumar Buyya[1]
[1]**Clou**d Computing and **D**istributed **S**ystems (CLOUDS) Laboratory,
Department of Computer Science and Software Engineering, The University of Melbourne, VIC 3010, Australia,
[2]Department of Software Engineering, Faculty of Computer Science and Information Systems,
Universiti Teknologi Malaysia (UTM), 81310 Skudai, Johor, Malaysia,
amirv@student.unimelb.edu.au, gtsayed2@siswa.utm.my, raj@csse. unimelb.edu.au

Abstract— **Cloud computing is a computing paradigm which allows access of computing elements and storages on-demand over the Internet. Virtual Appliances, pre-configured, ready-to-run applications are emerging as a breakthrough technology to solve the complexities of service deployment on Cloud infrastructure. However, an automated approach to deploy required appliances on the most suitable Cloud infrastructure is neglected by previous works which is the focus of this work. In this paper, we propose an effective architecture using ontology-based discovery to provide QoS aware deployment of appliances on Cloud service providers. In addition, we test our approach on a case study and the result shows the efficiency and effectiveness of the proposed work.**

Keywords- Cloud Computing; Virtual Appliances; Semantic Web Service; Web Service Modeling Ontology (WSMO); Service-Level Agreements (SLA); Open Virtualization Format (OVF).

I. INTRODUCTION

Cloud Computing is becoming one of the next emerging IT industry technologies. There are already more than twenty definitions for Cloud computing [1]. Among them, Ian Foster and colleague's definition of Cloud highlighted main aspects of Cloud namely as dynamic scalability, deliverable in economy of scales and on demand capability of scaling. According to their definition Cloud is [2]: "A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet."

On the other hand, as mentioned by Ian Foster et al. [2], clusters, supercomputers and partially grid relied on non Service Oriented Architecture (SOA) application, while Cloud focuses on Web 2.0 and SOA technology. Although Clouds adopted some common communication protocols such as HTTP and SOAP, the integration and interoperability of all services and finally service deployment remain biggest challenges. Service deployment, the process of making a service ready for use, often includes deploying multiple, interrelated software components into heterogeneous environments. Different technologies and tools try to satisfy user requirements in terms of software and hardware and to address these complexities by describing the environments, abstracting the dependencies, and automating the process [3] [4]. Among them, virtual appliances have been increasingly adopted by industry.

Virtual appliances, a set of virtual machines including optimized operating systems, pre-built, pre-configured, ready-to-run applications and embedded appliance specific components, are emerging as a breakthrough technology to solve the complexities of service deployment. Virtual appliances are proved to provide a better service deployment mechanism [5]. Therefore, they are going to be adopted as a major Cloud component working in application layer of Cloud [2].

Nevertheless, most of related works focused on satisfying user requirements using SOA architecture and virtualization, neglecting the proper consideration of Cloud computing environment as a service deployment resource provider. In a heterogeneous environment such as Cloud, it is difficult to enforce syntax and semantics of virtual machine description and user requirements. Therefore, applying symmetric attribute–based matching between requirements and request is impossible. In order to tackle those problems, we propose a flexible approach for performing Ontology-based discovery of Cloud virtual units. Followings are main contributions offered by this work:

1) Offering an approach which gives enough flexibility to end users to discover their needed appliance from range of providers and dynamically deploy it on different IaaS providers. 2) Proposing an advertisement approach for IaaS providers based on modeling virtual units into one of the most prominent initiatives in Semantic Web services, i.e., Web Service Modeling Ontology (WSMO) [25]. 3) Using ontology-based discovery for QoS-aware deployment of appliances on IaaS providers. This helps users to deploy their appliances on the most proper IaaS providers based on their QoS preferences when both sides (the providers and users) are not using the same notation to describe their services and requirements.

The remainder of this paper is organized as follows. We first give some background of related ideas in Section II. The whole architecture and its components are explained in Section III following by a case study in Section IV. Section V focuses on implementation. We review some of related works in Section VI and finally conclude the work in Section VII.

978-0-7695-4039-9/10 $26.00 © 2010 IEEE
DOI 10.1109/CCGRID.2010.87

II. PRELIMINARIES

In this section, concepts which are related to our approach, i.e. Web Service Modeling Ontology (WSMO), Ontology-based resource matching, Virtual appliance, and Open Virtualization Format (OVF) are described.

A. Web Service Modeling Ontology (WSMO)

WSMO [25] defines a model to describe Semantic WSs, based on the conceptual design set up in the Web Service Modeling Framework WSMF [26]. WSMO identifies four top-level elements as the main concepts [27]:

- *Ontologies*, provide the (domain specific) terminologies used and is the key element for the success of Semantic Web services. Furthermore, they use formal semantics to connect machine and human terminologies.
- *Web services*, are computational entities that provide some value in a certain domain. The WSMO Web service element is defined as follows:
  - *Capability*: This element describes the functionality offered by a given service.
  - *Interface*: This element describes how the capability of a service can be satisfied. The Web service interface principally describes the behavior of Web Services.
- *Goals,* describe aspects related to user desires with respect to the requested functionality, i.e. they specify the objectives of a client when consulting a WS. Thus they are an individual top-level entity in WSMO.
- *Mediators,* describe elements that handle interoperability problems between different elements, for example two different ontologies or services. Mediators can be used to resolve incompatibilities appearing between different terminologies (data level), to communicate between services (protocol level), and to combine Web services and goals (process level).

Besides these main elements, Non-Functional properties such as cost, deployment time, performance, scalability, and reliability are used in the definition of WSMO elements that can be used by all its modeling elements. Furthermore, there is a formal language to describe ontologies and Semantic Web services called WSML (Web Service Modeling Language) which contain all aspects of Web service descriptions identified by WSMO. In addition, WSMX (Web Service Modeling eXecution environment) is the reference implementation of WSMO, which is an execution environment for business application integration. [28].

B. Ontology-Based Resource Matching

There are works [6, 7, 8, 39] focusing on resource matching issues in the Grid using Semantic Web technologies. They have designed and prototyped an ontology-based resource selector that exploits ontologies, background knowledge, and rules for solving resource matching in the Grid. Resource matching is the process of selecting resources based on application requirements. Traditional resource matching, as exemplified by the Condor Matchmaker [9] or Portable Batch System [10] are considered as inflexible and difficult to extend to new characteristics or concepts. In their works unlike the traditional Grid resource selectors that describe resource/request properties based on symmetric flat attributes, separate ontologies are created to declaratively describe resources and job requests using an expressive ontology language. Instead of exact syntax matching, the ontology-based matchmaker performs Semantic matching using terms defined in those ontologies.

C. Virtual Appliance

In recent designs and implementations of virtualization systems, virtual appliances get the most attention. The idea has been initially presented [11] to address the complexity of system administration by making the labor of applying software updates independent of number of computers on which the software is run. Overall, the work develops the concept of virtual networks of virtual appliances as a means to reduce the cost of deploying and maintaining software. VMware [12] introduces new generation of virtual appliances which are pre-installed, pre-configured, and ready to run. However, in practical scenarios, pre-configured solutions can not satisfy varying requirements of users. In addition, those preconfigured virtual appliances occupy huge storages, if the system supports variety of operating system and software combinations. And it is not feasible for all range of users to have huge storage devices to store all those appliances shaped based on their configuration.

D. Open Virtualization Format (OVF)

The Open Virtualization Format (OVF) [13] is a hypervisor-neutral (the OVF doesn't rely on the use of specific hypervisor or virtualization platform), and open specification for the packaging and distribution of virtual appliances composed of one or more VMs. It aims to facilitate the automated, secure management of not only virtual machines but the appliance as a functional unit. OVF is virtualization platform neutral, while also enabling platform-specific enhancements to be captured. This makes it a proper format for Cloud computing where we have to deal with diversity of virtualization platforms.

III. PROPOSED ARCHITECTURE

In this work a unified architecture which can be seen in Fig. 1 is presented which utilizes appliance and Cloud computing to satisfy user requirements. It is useful to allow Cloud's users to deploy specific appliances which are not directly provided and supported by a Cloud IaaS provider. Therefore, the architecture helps clients to discover suitable appliances form different providers and then deploy it on IaaS providers.

A. Architecture Components

The Architecture main components are explained below:

1) Web portal: All services provided by the system are presented via the Web portal to service requesters. In addition, this component provides proper graphical interfaces to capture user's requirements such as software, hardware,

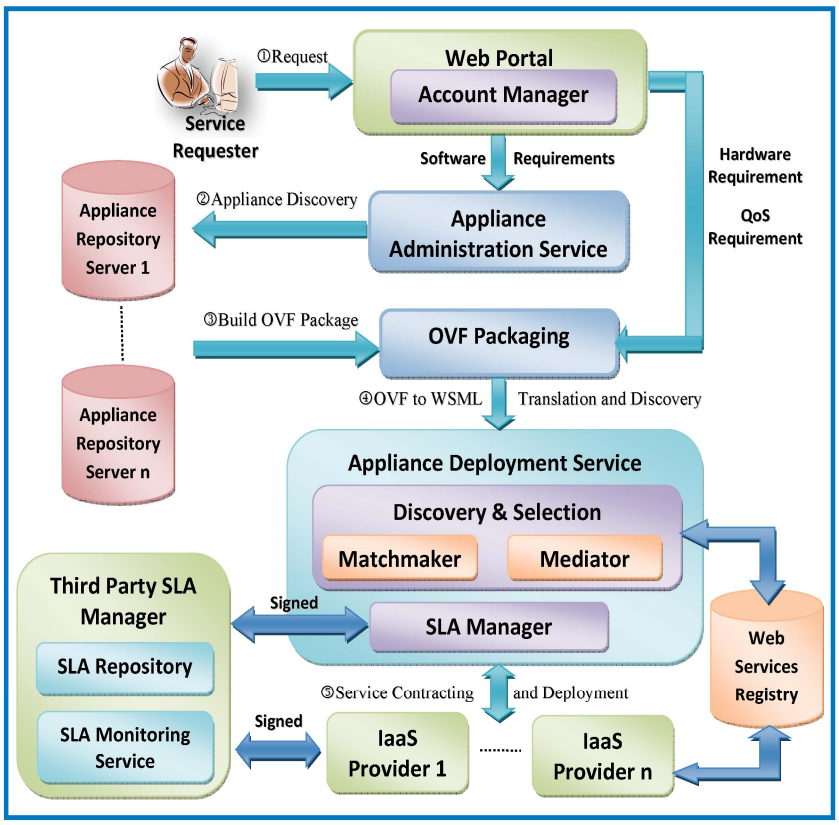

Figure 1. Phase of appliance deployment in Cloud based environment

and QoS requirements and contains account manager which is responsible for user management. It provides authorization and authentication for users and keeps the history of all users activities in the system.

2) Appliance Administration Service: This component provides desirable execution environment based on user requirements and providing necessary disk images and required information for running the application on the IaaS service provider side. After deployment phase, the component helps end users to manage their appliances (for example start or stop them).

3) OVF Packaging: After acquiring required disk images or their external URL addresses, we need to pack them along with other user requirements such as operating systems and hardware requirements into a standard format. The OVF standard is chosen for this purpose which has been largely adopted by the industry (VMware, Citrix and rPath) [14]. The OVF structure can be used to describe which software and from which appliance provider can be used. The possibility of using external references helps us to add disk images from different appliance providers and enhance flexibility of our approach.

4) Appliance Deployment Service: Since WSMO is used for discovery, user requirements are translated by the mediator into WSML format. Next, the Deployer Service which is in the platform layer of Cloud [2], maps appliance requirement to resources using the ontology-based discovery technique. It acts in user's interest to satisfy her quality of service (QoS) requirements by selecting the most desirable IaaS provider. Translation OVF metadata format to WSML notation is done by the mediator in this component.

5) Web Service Registry: It allows IaaS providers to advertise their virtual units. The advertisement of virtual unit contains descriptions of their features, costs, and the validity time of the advertisement. From standardization perspective, a common metamodel that describes IaaS providers' services has to be created. However, due to the lack of the standard, we have developed our own metamodel based on previous works and standards in this area using WSMO which explained in Section II.A.

6) SLA Manager: As Buyya et al. [29] mentioned to reach profit-making mainstream, it is essential to strengthen the role of Service-Level Agreements (SLAs) between the IaaS providers and the service requestor. Consequently, for taking care of quality of service requirement and Service level agreement, a framework for QoS-based Web service contraction is adopted for the architecture [24]. The service advertisement published by the IaaS providers in the registry can be considered as open SLA. This open SLA is

used by SLA manager to achieve an enforceable SLA by negotiating on QoS dimensions. The SLA is then signed by service provider and requestor and sent to a third party SLA manager to be kept and used for monitoring purpose.

7) Third party SLA Manager: A monitoring system is provided by this component for fairly determining to which extent an SLA is achieved as well as facilitating a procedure taken by a user to receive compensation when the SLA is violated. The monitoring is based on the copy of signed SLA which is kept in SLA repository. Third party monitoring results can be similar to what the CloudStatus [15] service reports. Hyperic's CloudStatus BETA is the first service to provide an independent view into the health and performance of the most popular Cloud services, including Amazon Web services and Google App Engine. CloudStatus gives users real-time reports and weekly trends on infrastructure metrics including service availability, response time, latency, and throughput which affect the availability and performance of Cloud-hosted applications.

8) IaaS Providers: They are in both fabric and unified resource level [2] and contain resources that have been virtualized as virtual units. Therefore, they expose their services as virtual units which can be a virtual computer, database system, or even a virtual cluster. IaaS providers advertise their virtual units as Web services in the Web service registry according to WSML notation. Among IaaS providers, Amazon Elastic Compute Cloud (Amazon EC2) has attracted considerable attention. Amazon EC2 [16] provides the flexibility to choose from a number of different instance types to meet various computing needs. Each instance provides a predictable amount of dedicated compute capacity and is charged per instance-hour consumed. Fig. 2 shows how an instance type of Amazon EC2 is modeled as a Web service.

B. *Matchmaker Architecture*

The matchmaker consists of two components: 1) Ontologies, provide the domain model and vocabulary for expressing virtual unit advertisements and service requestors requirements. 2) Matchmaking algorithm, determines when an advertised virtual unit matches a requester requirement description.

1) Ontologies. In this work two ontologies have been developed using WSML. We use WSMO studio [30] to create our ontologies. WSMO Studio is an open source Semantic Web service and Semantic Business Process modeling environment for the Web Service Modeling Ontology. WSMO Studio is available as a set of Eclipse plug-ins. The most useful WSMO Studio features include: Ontology editor with integrated WSML Reasoner (for consistency checks and querying of ontologies Editor for WSMO elements (Web services to advertise virtual units, goals to define user requirements, mediators). Each of ontology domains described by WSMO studio defines functional and non-functional properties and their elements. These two ontologies are:

a) Requirements ontology. The ontology as depicted in Fig. 2 captures a requester virtual unit requirements which are defined as functional properties (e.g., number of CPU, memory size) and non-functional properties (e.g., budget, location) which represent QoS requirements. The greater parts of our notation are taken from Common Information Model (CIM) [31] and in particular OVF for describing resource management which is impartial to IaaS providers and implementation.

b) Virtual unit ontology. The virtual unit ontology provides an abstract model for describing virtual units and their capabilities to let IaaS providers advertise their services. Our initial model concentrates on modeling of computational virtual unit services as depicted in Fig. 3.

2) Matchmaking algorithm: In order to consider whether a goal G which represents user requirements and a Web service W that represents advertised virtual units match on a semantic level, the sets G and W describing

```
/*****************************
* GOAL
*****************************/
goal _"http://example.org/GetVirtualUnit"
    nonFunctionalProperties
        dc#title hasValue "Goal of getting a virtual unit"
        dc#type hasValue _"http://www.wsmo.org/TR/d2/v1.2/#goals"
        wsml#version hasValue "$Revision: 1.1 $"
    endNonFunctionalProperties
    importsOntology { _"http://www.example.org/ontologies/OpSys",
            _"http://www.example.org/ontologies/VirtualHardware",
            _"http://www.wsmo.org/ontologies/location",
            _"http://www.wsmo.org/ontologies/cost"}

    capability _"http://example.org/GetVirtualUnit#capl"
        sharedVariables {?dep, ?budget, ?loc}
        effect havingAVirtualUnit
            nonFunctionalProperties
                VU#DeploymentLatency hasValue ?dep
                VU#Budget hasValue ?budget
                VU#Location hasValue ?loc
            endNonFunctionalProperties
            definedBy
                [Info hasValue "Guest Operating System",
                 Description hasValue "Unix"]
                  memberOf VU#OperatingSystemSection[hasId hasValue "99"] and
                [ [ElementName hasValue "Virtual Hardware Family",
                   InstanceID hasValue "0",
                   VirtualSystemType hasValue "vmx-04" ] memberOf VHS#System and
                  [Description hasValue "Number of virtual CPUs",
                   ElementName hasValue "1 virtual CPU",
                   ResourceType hasValue "3",
                   VirtualQuantity hasValue "1"] memberOf VHS#Item and
                  [AllocationUnits hasValue "byte * 2^20",
                   Description hasValue "Memory Size",
                   ElementName hasValue "256 MB of memory",
                   InstanceID hasValue "2",
                   ResourceType hasValue "4",
                   VirtualQuantity hasValue "256"] memberOf VHS#Item and
                  [AutomaticAllocation hasValue "true",
                   Connection hasValue "VM Network",
                   ElementName hasValue "ethernet adapter on 'VM Network'",
                   InstanceID hasValue "3",
                   ResourceType hasValue "10"] memberOf VHS#Item and
                  [Elementname hasValue "SCSI Controller 0 - LSI Logic",
                   InstanceID hasValue "4",
                   ResourceSubType hasValue "LsiLogic",
                   ResourceType hasValue "6"] memberOf VHS#Item and
                  [ElementName hasValue "Harddisk 1",
                   HostResource hasValue "ovf:/disk/lamp",
                   InstanceID hasValue "5",
                   Parent hasValue "4",
                   ResourceType hasValue "17"] memberOf VHS#Item and
                ] memberOf VU#VirtualHardwareSection [hasInfo hasValue
                  "Virtual Hardware Requirements: 256MB, 1 CPU, 1 disk, 1 NIC"]
```

Figure 2. Requirements Ontology

```
/*****************************
* WEBSERVICE
*****************************/
    nonFunctionalProperties
        dc#title hasValue "A virtual unit servcie"
        dc#type hasValue _"http://www.wsmo.org/TR/d2/v1.2/#services"
        wsml#version hasValue "$Revision: 1.1 $"
    endNonFunctionalProperties
    importsOntology { _"http://www.example.org/ontologies/OpSys",
            _"http://www.example.org/ontologies/VirtualHardware",
            _"http://www.wsmo.org/ontologies/location",
            _"http://www.wsmo.org/ontologies/cost"}

    capability _"http://example.org/VirtualUnit#cap1"
        sharedVariables {?dep, ?price, ?loc}
        effect
            nonFunctionalProperties
                VU#DeploymentLatency hasValue ?dep
                VU#Budget hasValue ?budget
                VU#Location hasValue ?loc
            endNonFunctionalProperties
            definedBy
                [Info hasValue "Supported Operating System",
                 Description hasValue "Red Hat Enterprise Linux",
                 Description hasValue "Windows Server 2003",
                 Description hasValue "Oracle Enterprise Linux",
                 Description hasValue "OpenSolaris",
                 Description hasValue "openSUSE Linux",
                 Description hasValue "Ubuntu Linux",
                 Description hasValue "Fedora",
                 Description hasValue "Gentoo Linux",
                 Description hasValue "Debian"]
                  memberOf VU#OperatingSystemSection and
                [ [ElementName hasValue "Virtual Hardware Family",
                   InstanceID hasValue "0",
                   VirtualSystemType hasValue "Small Unit" ] memberOf VHS#EC2Type and
                  [Description hasValue "Number of virtual CPUs",
                   ElementName hasValue "1 virtual CPU"] memberOf VHS#Item and
                  [AllocationUnits hasValue "byte * 2^30",
                   Description hasValue "Memory Size",
                   ElementName hasValue "1.7 GB of memory"] memberOf VHS#Item and
                  [AutomaticAllocation hasValue "true",
                   Connection hasValue "VM Network",
                   ElementName hasValue "ethernet adapter on 'VM Network'"]
                     memberOf VHS#Item and
                  [ElementName hasValue "Storage",
                   StorageCapacity hasValue "byte * 160 * 2^30",
                   HostResource hasValue "ovf:/disk/lamp"] memberOf VHS#Item and
                  [ElementName hasValue "Platform",
                     PlatformType hasValue "bit * 32"] memberOf VHS#Item and
                   [ElementName hasValue "I/O",
                   Performance hasValue "Moderate"] memberOf VHS#Item
                ] memberOf VU#VirtualHardwareSection [hasInfo hasValue
                  "Offering Virtual Hardware: 1.7 GB, 1 CPU, 1 disk, 1 NIC"]

    interface _"http://example.org/VirtualAppliance#cap1"
        choreography _"http://example.org/tobedone"
        orchestration _"http://example.org/tobedone"
```

Figure 3. Virtual units Ontology

these elements have to be interrelated somehow; precisely speaking, we expect that some relationship between G and W has to be exist. The service matching in the proposed architecture is based on Description Logics (DLs) [38] which are a family of knowledge representation formalisms that are able to represent the structural knowledge of an application domain through a knowledge base including a terminology and a world description. The basic formalism of a DL system comprises three components: 1) Constructors which represent concept and rule, 2) *Knowledge base* (KB) which consists of the TBox(*terminology*) and the ABox(*world description*). The TBox presents the vocabulary of an application domain, while the ABox includes assertions about named individuals in terms of this vocabulary, 3) Inferences which are reasoning mechanisms of Tbox and Abox.

Before we proceed to define discovery, we need to introduce the goal and five matching operations described below:

Definition 1 (Goal) *Let $\mathcal{T}$ be an acyclic Tbox. A Goal G for $\mathcal{T}$ is defined in the form of $G=(C_G, I_G, N_G)$, where:*

- *C_G is the set of capabilities of Web services including goal constraints, which the user would like to have.ψ*
- *I_G is the set of interfaces of Web service, which the user would like to have and interact with.ψ*
- *N_G is the set of Nonfunctional properties, which is similar to that attached to Web services.ψ*

Definition 2 (Exact matching) *Suppose that a requested capability of a Goal $C_{G_1} \in G_1$ is given. Let a capability of a Web service $C_{W_1} \in W_{1\psi}$and Nonfunctional properties of a Web service $N_{W_1} \in W_1$. If $(N_{G_1} \equiv N_{W_1}) \sqcap (C_{G_1} \equiv C_{W_1})$ then G_1 can "exactly" match W_1,i.e. $G_1 \equiv W_1$.*

Definition 3 (PlugIn matching) *Suppose that a requested capability and Nonfunctional properties of a Goal $C_{G_1} \in G_{1\psi\psi\psi}$, $N_{G_1} \in G_{1\psi}$are given. Let a capability of a Web service $C_{W_1} \in W_1$ and Nonfunctional properties of a Web service $N_{W_1} \in W_1$. If $(C_{G_1} \sqsubseteq C_{W_1}) \sqcap (N_{G_1} \sqsubseteq N_{W_1})$ then $G_1 \sqsubseteq W_1$. This match is called "PlugIn".*

Definition 4 (Subsumption matching) *Suppose that a requested capability and Nonfunctional properties of a Goal $C_{G_1} \in G_1$,$N_{G_1} \in G_1$ are given. Let a capability of a Web service $C_{W_1} \in W_1$ and Nonfunctional properties of a Web service $N_{W_1} \in W_1$. If $(C_{W_1} \sqsubseteq C_{G_1}) \sqcap (N_{W_1} \sqsubseteq N_{G_1})$ then $W_1 \sqsubseteq G_1$. This match is called "Subsumption".*

Definition 5 (Intersection matching) *Suppose that a requested capability and Nonfunctional properties of a Goal $C_{G_1} \in G_{1\psi\psi\psi}$, $N_{G_1} \in G_{1\psi}$are given. Let a capability of a Web service $C_{W_1} \in W_1$ and Nonfunctional properties of a Web service $N_{W_1} \in W_1$. If $\neg(C_{G_1} \sqcap C_{W_1} \sqsubseteq \bot) \sqcap \neg(N_{G_1} \sqcap N_{W_1} \sqsubseteq \bot)$ then $\neg(G_1 \sqcap W_1 \sqsubseteq \bot)$. This match is called "Intersection".*

Definition 6 (Non-matching) *Suppose that a requested capability and Nonfunctional properties of a Goal $C_{G_1} \in G_{1\psi\psi\psi}$, $N_{G_1} \in G_{1\psi}$are given. Let a capability of a Web service $C_{W_1} \in W_1$ and Nonfunctional properties of a Web service $N_{W_1} \in W_1$. If $(C_{G_1} \sqcap C_{W_1} \sqsubseteq \bot) \sqcap (N_{G_1} \sqcap N_{W_1} \sqsubseteq \bot)$ then $G_1 \sqcap W_1 \sqsubseteq \bot$. This relationship is called "Non match".*

Based on the above definitions, we propose the Web service discovery algorithm which can be seen in Fig. 4 and also define it as follows:

Definition 7 (Discovery) *Suppose that a requested capability and Nonfunctional properties of a Goal $C_{G_1} \in G_{1\psi\psi\psi}$, $N_{G_1} \in G_{1\psi}$are given. Let a capability of a Web service $C_{W_1} \in W_1$ and Nonfunctional properties of a Web service $N_{W_1} \in W_1$. Discovery is defined as to find a set of Web services W_i such that:*

$$((C_{G1} \equiv C_{W1}) \leftarrow \sqcap (N_{G1} \equiv N_{W1})) \sqcup ((C_{G1} \sqsubseteq C_{W1}) \leftarrow \sqcap (N_{G1} \sqsubseteq N_{W1})) \sqcup ((C_{W1} \sqsubseteq C_{G1}) \leftarrow \sqcap (N_{W1} \sqsubseteq N_{G1})) \sqcup ((\neg(C_{G1} \sqcap C_{W1} \sqsubseteq \bot)) \leftarrow \sqcap (\neg(N_{G1} \sqcap N_{W1} \sqsubseteq \bot))).$$

Algorithm 1 WSDiscovery ($\mathcal{G}_{user}$, $\mathcal{G}_{exist}$, $\mathcal{W}$)

Inputs: $\mathcal{G}_{user}$ is the user's goal, $\mathcal{G}_{exist}$ is set of existing goals, $\mathcal{W}$ is set of WSMO Web services

1: **for all** $\mathcal{G} \in \mathcal{G}_{exist}$ // $\mathcal{G}$ is an existing WSMO Goal
2: **if** $\mathcal{G}_{user} \equiv \mathcal{G}$ **then**
3: **return** ($\mathcal{G}_{user}$, matchtype($\mathcal{G}$))
4: **else**
5: **for all** $w \in \mathcal{W}$ **do**// w is a Web service
6: **if** $C(\mathcal{G}_{user}) \equiv Cw$ **and** $N(\mathcal{G}_{user}) \equiv Nw$ **then**
7: //C is capability and N is non-functional properties
8: **return** (w, *Exact*)
9: **else if** $C(\mathcal{G}_{user}) \sqsubseteq Cw$ **and** $N(\mathcal{G}_{user}) \sqsubseteq Nw$ **then**
10: **return** (w, *PlugIn*)
11: **else if** $Cw \sqsubseteq C(\mathcal{G}_{user})$ **and** $Nw \sqsubseteq N(\mathcal{G}_{user})$ **then**
12: **return** (w, *Subsumption*)
13: **else if** $\neg(C(\mathcal{G}_{user}) \sqcap Cw \sqsubseteq \perp)$ **and**
14: $\neg(N(\mathcal{G}_{user}) \sqcap Nw \sqsubseteq \perp)$ **then**
15: **return** (w, *Intersection*)
16: **end for**
17: **end for**
18: **return** ($\mathcal{G}$, *NonMatch*)

Figure 4. Web service discovery algorithm

The architecture operations are described in following sections. First initial steps are presented, and then details of execution phases which are desired for satisfying user requests are explained. To enable execution phases, the first step is to build a proper environment which is constructed during initial pashas.

C. Initial phases

In this section, phases which have to be done prior to the execution phases are discussed.

- First, each service requestor has to have an account in the system. The account is used for user's authentication and authorization; besides, it stores all user information regarding their previous requests. This information can help the systems to offer better quality of service to the user. For example assume that requestor face the failure in deploying his appliances on specific Cloud in previous interaction with system, this information will pass to the deployment service to avoid similar situation in later requests.
- It is necessary to have Web Services Registry which contains semantic description of Web services, such as their capabilities (pre-conditions, post-conditions, assumptions and effects), interfaces (choreography) and non-functional properties. This is the place for all IaaS providers to advertise their virtual units as a service. A sample of advertisements in the Web service registry is shown in Fig. 3.
- Ontology repository is built up to contain ontologies for describing semantics of particular domains. Any components might wish to consult ontology, but in most of the cases ontologies will be used by the mediator related components to overcome data and process heterogeneity problems. In our case, semantic has to be described for operating systems, virtual hardwares, and other QoS domains.
- A Trusted Third Party for keeping SLA contracts and their monitoring has to be in place. The idea was proposed by [17] and adopted by our architecture. That's because we are going to provide a SLA monitoring system, which is capable of fairly determining to which extent a SLA is achieved and facilitating a procedure taken by the user to receive compensation when the SLA is violated.

D. Execution phases

The following detailed execution phases are done during appliance deployment in Cloud-based environment.

- In ① Service requestor specifies certain requirements such as hardware requirements like CPU, storage, and memory. A client request may just describe some of needed resources for example only CPU and storage. In this situation, default values for other requirements are assigned by the portal. These default values are presented by the portal and could be assigned according to the software requirements and previous requested virtual units of users.
- In phase ②Software requirements used as an input for searching the best suited appliances among various repositories of virtual appliance providers [19] which named as virtual market place [18] by VMware. VMware offers the industry's largest virtual appliance marketplace, gives users the quickest way to browse and try applications designed to run best in a virtual machine [18]. 3Tera [20] and rPath [21] are other appliance providers to be considered.
- Phase ③ is dealing with building OVF package and its metadata based on discovered virtual appliances from external appliance providers.
- Since our discovery is working based on ontology, therefore during phase ④ the mediator translates OVF metadata (only metadata portions which is required for discovery) to WSML notation. Therefore service requestor hardware, operating system, and QoS requirements are modeled as a goal as illustrated in Fig. 2. Then discovery service uses ontology-based matchmaker to select best available virtual unit represented by a Web service in the registry.
- Finally in phase ⑤ enforceable SLA achieved by negotiation on QoS dimensions between the SLA manager and IaaS providers. Enforceable SLA will be signed by both parties and the obtained Web service contract is kept and continuously monitored by the third party.

IV. CASE STUDY

In this section our approach is validated on a case study to show the effectiveness of the proposed work. To show its applicability, it has been tested in Web Service Modeling Toolkit (WMST) [25].

The Audio Video Devices (AVD) online store has a powerful Website for selling digital gadgets. Their business is initially based on Europe and they have just expanded it to US. They have leased a dedicated server from a data center in US. Nevertheless, due to a business plan for announcement of twenty percent discount in some category of items, an exceptional load is predicted to build up on the server. As they have doubted about the ROI they are not going to lease another server. Recently, they have been informed about Cloud computing and its pay-as-you-go [31] manner and found it very useful for their case. That is because they can lease a virtual server even for an hour and terminate it when the load back to the normal level. However, they are seeking for a solution to deploy their application automatically on the most suitable IaaS provider.

We show how our work can help them to achieve their goal. Their virtual unit requirements are depicted in Table I and virtual units' specification which have been advertised by providers are shown in Table II. Operating systems supported by each IaaS providers are depicted in Table III.

First the AVD IT officer connects to the portal and expresses her software, hardware, and other requirements. She needs an Apache server to be installed on a virtual unit with specification illustrated in Table I. Consequently, Appliance Administration Service discovers a suitable Apache appliance which will be packed according to OVF standard. Next, the matchmaker as explained in Section III.B checks the capabilities of both virtual units Web services against the resource requirements. Since the knowledge base (KB) specifies that both "Linux family" and "OpenSolaris" are types of "Unix", therefore not only A but also C IaaS provider, pass the operating system requirement criteria.

Both Providers A and C services (which is located in US) pass functional requirements criteria based on Definition 3. As it shows in Fig. 5, providers C in EU and A match type with user requirements is Subsumption as they cannot satisfy location and deployment time criteria correspondingly. However, the provider C in US is the most preferable as it can satisfy all requirements and its match type is PlugIn. Next, as described in phase 5 in Section III.D the signed SLA will be sent to the third party for monitoring. In this case, the third party realized that deployment time was 80 seconds which is 3.5 seconds more that what both parties have been agreed on. Therefore, the third party informs them, and AVD is found eligible for receiving compensation as the SLA was violated.

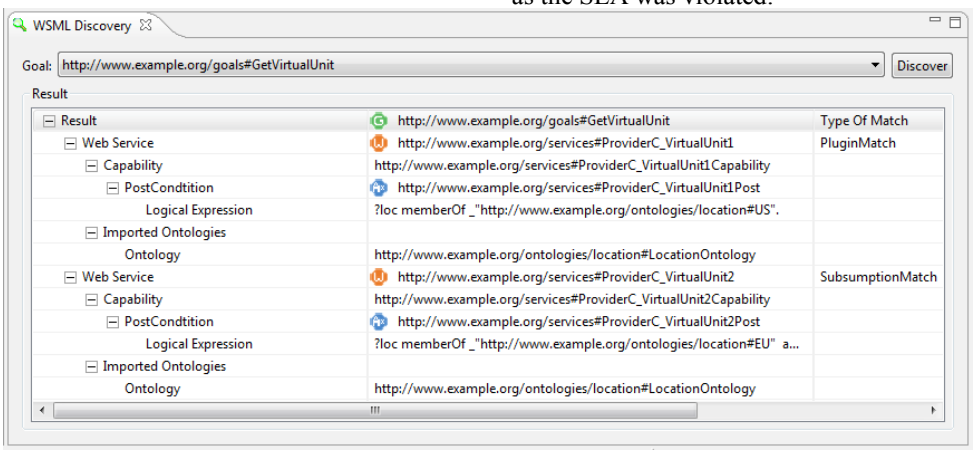

Figure 5. Case study validation in WSMT environment

TABLE I. REQUEST

Requestor	CPU(core MHz)	Memory	Storage	Platform	Budget	Location	D-time (Sec)	OS
AVD	800	1.5 GB	140 GB	32-bit	0.15 $	US	79	Unix compatible

TABLE II. VIRTUAL UNITS

Providers	CPU(core MHz)	Memory (GB)	Storage(GB)	Platform	Price per hour	Location	D-time (Sec)
A	1000	1.7	160	32-bit	0.12$	US	79.5
B	1000	2	120	64-bit	0.38$	US	80
C	1000-1200	1.7	170	32-bit	0.10$	US	76.5
C	1000-1200	1.7	170	32-bit	0.11$	EU	76.5
A	4000	7.5	850	64-bit	0.138$	US	78

TABLE III. SUPPORTED OPERATING SYSTEM

Providers	Operating Systems
A	UNIX, Debian 5.0, 4.0 ; Ubuntu 9.04, 8.10 ; Windows Web Server 2008
B	Windows Server 2008 ; Windows Server 2003
C	OpenSolaris ; OpenSUSE Linux ; Ubuntu Linux ; Windows Server 2003

V. IMPLEMENTATION

After doing implementation feasibility study, tools demonstrated in Table IV show capabilities to represents components in the architecture. However, with the best of our knowledge, none of the current tools in the market support automated Cloud discovery.

In this section, we just focus on two of mentioned tools in Table IV which are used in Appliance administration and OVF components in the proposed architecture.

TABLE IV. ARCHITECTURE IMPLEMENTATION

Component	Tools
Appliance Administration Service	v kernel [33], rBuilder [21]
OVF Packaging	Kensho [14], VMware OVF Tool [34]
Third Party Monitoring Service	CloudStatus [15], Monitis [35], Nimsoft [36]
Appliance Deployment service	VMware Studio [37]
Web Portal	Life ray Enterprise Open Source Portal [32]

First is the rPath Appliance Platform Agent (rAPA) which is an extensible application framework that provides Web-based remote administration for appliances [21]. It can be applied to view a description of the appliance as well as some basic appliance status information and logs. Furthermore, it is capable of configuring the HTTP and HTTPS proxies used by the appliance. The second tool is the Project Kensho OVF Tool which uses the OVF standard for export and import of virtual appliances and also Common Information Model (CIM) industry standards developed by the Distributed Management Task Force (DMTF).

VI. RELATED WORK

Recently, many works have targeted satisfying end user requirements using virtualization approach [4, 3, 22]. In this section, two of most recognized works are reviewed.

Keahey et al. [22] presented the idea of virtual workspace (VW) which allows users to define an environment in terms of their requirements (such as resource requirements or software configuration), manage and then deploy it in the Grid and Cloud. They have their own Cloud for deployment of VW which named Nimbus Cloud. It provides virtualization in the form of Xen virtual machine and can be used to make a request to deploy a workspace based on a specified VM image. Finally, it has to be mentioned that they have not considered the user QoS requirements and in general SLA. In addition, Cloud discovery and selection is missing from the work.

A phenomenal related work has been done in North Carolina State University. The project name is Virtual Computing laboratory (VCL) [23] which was originally described in February 2004. VCL claims that it is an ideal product to support all kind of Cloud solution. VCL services vary from virtual computer laboratory seats or desktops, to single applications on demand, to high-performance computing services and clusters. VCL is now one of the most well-known virtualization management systems in the world, particularly in academia. However, VCL support for images is limited to few types of software and it cannot adequately capture users' varying requirements.

VII. CONCLUSION

In this paper, an effective architecture for appliance deployment is presented. The presented approach includes three main improvements: converting user requirements to OVF to be a standard package format for Cloud deployment, proposing an advertisement approach for IaaS providers, and applying ontology-based discovery to find the best suited providers. One of desired attributes of our architecture is to allow users to present their requirements in terms of high-level and general software and hardware characteristics, which will be mapped to appliances and virtual units. In addition, we plan to investigate integration of SLA-based appliance discovery to our system to further enhance QoS for end users.

REFERENCES

[1] L. Vaquero, L. Rodero-Merino, J. Cáceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition", SIGCOMM Comput. Commun. Rev., Vol. 39, No. 1, 2009, pp. 50-55.

[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared", IEEE Grid Computing Environments Workshop, 2008, pp. 1-10.

[3] Z. Cheng, Z. Du, Y. Chen, and X. Wang, "SOAVM: A Service-Oriented Virtualization Management System with Automated Configuration", IEEE Int'l Workshop on Service-Oriented System Engineering, 2008, pp. 251-256.

[4] P. Anedda, M. Gaggero, and S. Manca, "A general service oriented approach for managing virtual machines allocation", ACM Symposium on Applied Computing, ACM, 2009, pp. 2154-2161.

[5] C. Sun, L. He, Q. Wang, and R. Willenborg, "Simplifying Service Deployment with Virtual Appliances", IEEE Int'l Conf on Services Computing, 2008, pp. 265-272.

[6] H. Tangmunarunkit, S. Decker, and C. Kesselman, "Ontology-Based Resource Matching in the Grid – The Grid Meets the Semantic Web", Int'l Semantic Web Conf, Springer-LNCS, 2003, pp. 706-721.

[7] S. Pahlevi, A. Matono, and I. Kojima, "SPARQL-Based Set-Matching for Semantic Grid Resource Selection", Business Process Management Workshops, 2007, pp. 461-472.

[8] S. Pahlevi and I. Kojima, "Ontology-Based Grid Index Service for Advanced Resource Discovery and Monitoring", European Grid Conf, 2005, pp. 144-153.

[9] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", The 7th IEEE Int'l Symp on High Performance Distributed Computing, 1998, pp. 140-147.

[10] The Portable Batch System. [Online]. Available: http://pbs.mrj.com

[11] C. Sapuntzakis, et al., "Virtual Appliances for Deploying and Maintaining Software", The 17th USENIX Conf on System Administration, 2003, pp. 181-194.

[12] J. Staten, E. Brown, F. Gillett, and W. Saleh, "The Case For Virtual Appliances How Hypervisors Can Simplify Software Distribution", Forrester Research, 2007. [Online]. Available: www.vmware.com/go/vam_va_fd_forresterreport

[13] Open Virtualization Format White Paper, June 2009. [Online]. Available: www.dmtf.org/standards/published_documents/DSP2017_1.0.0.pdf

[14] Project Kensho v1.1 Technology Preview. [Online]. Available: http://community.citrix.com/display/xs/Kensho

[15] Enterprise Monitoring and Management for Cloud Services. [Online]. Available: http://www.hyperic.com/products/Cloud-monitoring.html

[16] Amazon Elastic Compute Cloud (Amazon EC2), [Online]. Available: http://aws.amazon.com/ec2/

[17] H. Akio. Monitoring of service level agreement by third party. US Patent 7007082, February 28, 2006.

[18] Virtual Appliance Marketplace. [Online]. Available: http://www.vmware.com/appliances/learn/overview.html

[19] M. Wilson, "Constructing and Managing Appliances for Cloud Deployments from Repositories of Reusable Components", Workshop on Hot Topics in Cloud Computing, 2009.

[20] 3Tera AppStore. [Online]. Available: http://www.3tera.com/AppStore/

[21] rBuilder Online. [Online]. Available: https://www.rpath.org/ui/

[22] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Achieving Quality of Service and Quality of Life in the Grid", Scientific Programming Journal, Vol. 13, No. 4, 2005, pp. 265-276.

[23] M. Vouk, et al. "Powered by VCL - Using Virtual Computing Laboratory (VCL)", The 2nd Int'l Conf on Virtual Computing, 2008, pp. 1-10.

[24] M. Comuzzi and B. Pernici, "A framework for QoS-based Web service contracting", ACM Trans. On the Web, Vol. 3, No. 3, 2009, pp. 1-52.

[25] WSMO Working Group. [Online]. Available: http://www.wsmo.org

[26] D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF", Electronic Commerce: Research and Applications, Vol. 1, No. 2, 2002, pp. 113-137.

[27] Fensel D., et al., Enabling Semantic Web Services: Web Service Modeling Ontology, Springer, 2006.

[28] WSMX Working Group. [Online]. Available: http://www.wsmx.org

[29] R. Buyya, C. Shin-Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility", Future Generation Computer Systems, Elsevier Science, Vol. 25, No. 6, 2009, pp. 599-616.

[30] Common Information Model (CIM) Standards. [Online]. Available: http://www.dmtf.org

[31] M. Armbrust, et al., "Above the clouds: A Berkeley view of cloud computing", Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[32] Life ray Enterprise Open Source Portal. [Online]. Available: http://www.liferay.com/web/guest/home

[33] Virtual Appliance Management Suit. [Online]. Available: http://www.vkernel.com

[34] VMware OVF Tool. [Online]. Available: http://communities.vmware.com/community/developer/ovf

[35] Monitis Cloud Performance Monitoring Service. [Online]. Available: http://monitorCloud.com/monitorCloud

[36] Nimsoft Cloud Monitoring. [Online]. Available: http://www.nimsoft.com/solutions/Cloud-monitoring/index.php

[37] VMware Studio. [Online]. Available: http://www.vmware.com/appliances/learn/vmware_studio.html

[38] Baader, F., D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[39] S. Grimm, "Intersection-Based Matchmaking for Semantic Web Service Discovery", The 2nd Int'l Conf on Internet and Web Applications and Services, 2007, pp. 14-14.