Specifying Cloud Application Requirements: An Ontological Approach

Yih Leong Sun, Terence Harmer, and Alan Stewart

Queen's University of Belfast,
University Road, Belfast, BT7 1NN, UK
{ysun05,t.harmer,a.stewart}@qub.ac.uk
http://www.qub.ac.uk

Abstract. Increasingly business organisations are deploying service applications onto cloud infrastructures. Given the available range of infrastructure providers and products, it is a challenging task to select the most appropriate set of cloud resources for a given application. Cloud providers offer resources in various formats using different pricing structures. There is a mismatch between the terminology used to specify an application's requirements and that used to describe provider resources. In this paper, a resource allocation approach based on mapping application requirements onto cloud infrastructure products is proposed. Two domain-specific ontologies for media transcoding and financial services are used to illustrate how application requirements can be modelled. It is then shown how requirements can be mapped onto a general ontological description of cloud resources. The resource ontology is provider-agnostic and provides a framework for searching the cloud market for a set of products that meet an application's requirements.

Keywords: cloud computing, cloud programming model, ontology.

1 Introduction

Many organisations are utilising cloud infrastructures as a flexible and cost-effective platform on which to execute business applications. Increasingly providers offer infrastructure resources or services in the cloud market for hosting cloud-based applications. Different providers offer different ways of leasing their cloud resources using different pricing structures. In order to achieve a high degree of business continuity, it is important that cloud-based applications can operate even under adverse conditions. In the event of service interruptions caused by a provider's resources malfunctioning, the organisation should have the option to migrate their applications elsewhere and avoid a vendor lock-in situation. From the application developer's point of view, finding an appropriate set of infrastructure resources that meet an application's requirements in a multi-provider cloud environment is a challenging task because of the range of products available as well as the dynamic nature of the market.

Typically developers analyse an application's requirements and then select a suitable set of infrastructure resources on which to execute the application.

M. Yousif, L. Schubert (Eds.): Cloudcomp 2012, LNICST 112, pp. 82–91, 2013.

Consequently, there is a need for a programming model which facilitates the specification and construction of cloud-based infrastructures. Such a programming model should enable application developers to build cloud-based infrastructures easily and rapidly. In order to develop such a programming model, it is desirable to have a means for mapping application requirements onto infrastructure products (offered by multiple providers). Many different frameworks have been proposed for discovering and utilising cloud resources. Typically cloud requirements are analysed from the provider's point of view, based on the resource capabilities offered by the provider. However, there is a mismatch between the perspective of application developers and the low-level details supplied in a typical resource specification. Currently, there are no suitable mechanisms for describing requirements from an application's viewpoint. This paper proposes an application-centric, multi-layer ontology as a way of describing application requirements in the cloud context. The ontology allows application developers to formulate high-level domain-specific application requirements and subsequently to use these descriptions to search for the most appropriate set of resources in a multi-provider cloud environment. There is potential for application developers to utilise the proposed ontology so as to automate the resource discovery process.

This paper is organised as follow. Section 2 provides a brief summary of a provider-agnostic programming model. Section 3 discusses related research and defines a multi-layer model. Section 4 describes ontology translation using two examples while conclusion are drawn in Section 5.

2 A Provider-Agnostic Programming Model

An overview of a programming model for selecting and utilising cloud-based infrastructures in a multi-provider cloud environment is given below.

The model is wrapped in a provider-agnostic API [7] (see Figure 1) and incorporates a set of cloud providers that make up the cloud market. The selection of a particular provider depends on user preferences and provider's financial models. This set of providers is associated with a pool of available resources. In practice, an application is mapped onto a set of resources from the resource pool that meets the application's requirements. Multiple types of suitable resources may be discovered. An initial result set can be filtered using heuristics in order to find the most suitable set of resources for a particular application. After a best fit resource is identified, it can be reserved for future use. When resources are needed, they are instantiated and user's application is deployed. After the application finishes, the underlying resources can be discarded.

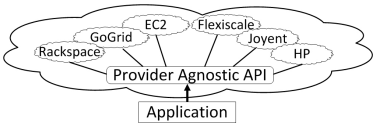

Fig. 1. A provider-agnostic programming model

This model allows developers to acquire infrastructure resources without knowledge of the internal implementation details of the underlying providers. This provides an abstract view of infrastructure resources and insulates the application from API changes arises from the underlying providers. Applications can be deployed and scaled according to system constraints, within a given budget, and have portability across multiple providers. This paper proposes the use of ontologies to formulate application requirements. These requirements are subsequently used in the resource mapping process of the proposed model.

3 A Multi-layer Ontological Model

An *ontology* is a formal description of the entities and the relationships between them within a particular knowledge domain. Ontologies have been used to describe cloud resources elsewhere. Most cloud-related ontologies are resource-centric and give definitions from the perspective of the capabilities of a particular resource provider.

Bernstein et al [2] proposes an ontology-based catalog which describes the resource capabilities offered by cloud providers, such as CPU, storage and compliance capabilities. Reservoir [4] proposes a service specification mechanism which includes VM details, application and deployment setings. Mosaic [8] introduces an ontology to describe cloud resources with a set of functional and non-functional properties. LoM2HiS [3] proposes a framework for mapping low-level resource metrics to high-level SLA parameters which focuses on hardware or network attributes. This paper proposes an application-centric multi-layer ontology that focuses on user requirements rather than just the cloud resources.

In this paper, a three-layer model is used to describe the process of mapping application requirements onto cloud resources (see Figure 2). The top layer uses a *domain-specific ontology* to express high-level application requirements semantically using application specific terminology. Two examples of such domain-specific requirements are: (i) application data must be processed within UK in order to be compliant with the UK Data Protection Act; (ii) media file must be transcoded into *wmv* format and played on *windows phone* device.

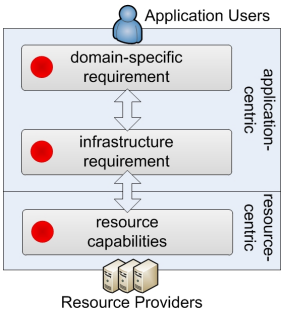

Fig. 2. A multi-layer ontology model

The middle layer of the proposed model is an *infrastructure requirement ontology*. This layer describes provider-agnostic infrastructure constraints that are needed to deliver the application requirements. In going from the top to the middle layer, high-level *domain-specific* requirements are mapped to *infrastructure level* requirements. The bottom layer in the model is the *resource* layer which specifies the resource capabilities offered by various cloud providers. This paper focuses on the *domain-specific* and *infrastructure requirement* layer. The *resource* layer has been widely investigated elsewhere [2,8].

3.1 Domain-Specific Ontology

A domain-specific ontology can be used to capture high-level application constraints. The ontological layer is application-centric, focused on user needs and expressed using domain specific terminology. Two examples of domain-specific ontologies are given in order to illustrate the model.

Media transcoding is the process of converting media files (video or audio) from one format to another. Transcoding is computational intensive and requires high storage and fast bandwidth [6,10]. Often users impose a budget for the provisioning of transcoding infrastructure. Consider a media company that broadcasts a series of animation videos. The video sources use *avi* format and are made available 5 hours before the broadcast schedule. For certain applications, these need to be transcoded into *windows media* format at a frame rate of *30 frame per second* and delivered to *Windows Phone* devices via *http streaming*. The company has a budget of *£100* for the transcoding operations. The application's requirements may be specified in a high-level notation as:

> Video conversion : AVI to Windows Media
> Mobile encoding : Windows Phone
> Delivery deadline : 9am next morning
> Encoding features : frame-rate conversion; http adaptive streaming
> Budget : £100

More generally, the following domain-specific ontology is used for specifying media transcoding requirements:

Budget requirements specify monetary constraints for running a transcoding task. These can be specified as the maximum amount that a user is prepared to spend per day or per hour.

Format requirements specify the container format of the source and transcoded media; for example, transcoding a video from *avi* format to *flv* format.

Codec requirements are the audio or video codecs of the media; for example, *mpeg4*, *h264*, *mp3*, *aac*.

Device requirements refer to the destination devices that the transcoded media will be played on; for example, *iPhone*, *Windows Phone*, *PC*.

Processing Filter requirements are advanced video and audio filters, including both pre-processing and post-processing filters; for example, frame rate conversion, de-interlacing, watermarking, audio resampling, etc.

Content Sensitivity requirements refer to the sensitivity of the media content. A sensitive media content is associated with high security capabilities; for example, secure transfer channel or encryption.

Output Storage requirements refer to the storage destination of transcoded media; for example, cloud storage, such as Amazon S3.

Delivery Time requirements refer to the time when the transcoded media will be ready for delivery (i.e. when the transcoded media is needed for use).

Delivery Channel requirements refer to how the transcoded media is delivered; for example, to be delivered as a downloaded file, HTTP live streaming over the internet, or mobile streaming over 3G network.

The proposed transcoding ontology adopts the terminology used by the media industry. Many media people would be familiar with such high-level specifications rather than the details of hardware or software resources that need to be deployed to perform transcoding tasks. Moreover, there are many different solutions which can be used to implement the same transcoding task: developers can use a *basic resource* facility and additionally install the transcoding software themselves; alternatively, developers could use a *packaged resource* with preinstalled software, such as the AWS resource pre-built with Wowza Transcoder [1]. By using the proposed model, media users can specify an application's requirements using appropriate terminology; this in turn allow developers to search for the most suitable set of resources. It is up to developers to define a mapping algorithm which relates the domain-specific ontology to the infrastructure constraints. Historical database can be used to facilitate the mapping process by providing actual performance data. For example, in [6] several jobs are run concurrently on a multi-core resource in order to meet a delivery deadline.

Financial service sector is a challenging domain for infrastructure deployment [9]. It is highly regulated and demands high availability of resources. One way to increase infrastructure resilience is to deploy mirror infrastructures located in different geographical locations. Consider a financial company which needs to revalue their customers' portfolios on a daily basis [9], the company must comply with the UK Data Protection Act. The application service must be made available 12 hours a day. Extra application services are required if several large portfolios are to be revalued at the same time. These requirements can be specified as:

Compliance: UK Data Protection Act 1998
Service hours required: 8am to 8pm, weekdays
Service performance: High response time; high availability;
high scalability; high disaster recovery
Portfolio valuation software: Supplied by users

A generic domain-specific ontology for financial services application is proposed as follows:

Budget requirements are the monetary constraints for running the financial application, typically specified as a maximum spend per month.

Compliance requirements are the rules, regulations, legislation or laws that need to be comformed with in the financial service domain, such as UK Data Protection Act 1998.
Security requirements refer to how financial data is accessed and transferred.
Data requirements refer to the quality and integrity of the financial data. For example, financial data must be verified and audited by a third-party data verification service.
Performance requirements indicates the uptime requirements or guaranteed response time for a certain time range in a day, for example, 99.99% uptime and 100ms response time between 8am to 8pm, Monday to Friday.
Availability requirements refer to the capabilities of a financial application to continue operate without service interruption in the event of component failures, for example, availability level can be categorised as high, medium or low; a high availability requirements indicates that a mirror infrastructure must be provisioned in different geographical locations.
Scalability requirements refer to how flexible the infrastructure can grow or shrink when demands fluctuate.

Financial companies require quick turn-around time to deploy applications in order to remain competitive in the fast-paced financial market. The proposed ontologies provide an easy and quick mechanism for financial users to specify high-level requirements using appropriate terminology. Developers can translate these high-level specifications to lower level infrastructure constraints. For example, *low availability* means that infrastructure replication is not required, whereas, *high availability* means that an application must be deployed on a mirror infrastructure located in different geographical locations.

3.2 Infrastructure Requirement Ontology

In the infrastructure requirement ontology, a **requirement** specifies the capabilities or qualities that are necessary (or desired) for an infrastructure. **Infrastructure requirements** are divided into different categories (see Figure 3):

Cost requirement is the budget for deploying cloud infrastructure.
Performance requirement refers to effectiveness and quality of the infrastructure. **Network latency performance** is the delay incurred in the processing of data across the network; **bandwidth performance** is the speed of the network including **incoming** and **outgoing bandwidths**.
Resource requirement refers to the specification of individual resources (hardware, software or operating system). Four categories are identified: **hosting environment** defines the operating system requirement of the host, such as Windows 7; **hardware capability** refers to the hardware components, such as CPU, RAM, storage space; **software stack** indicates the list of software or services that need to be installed on a resource.
Geographical requirement refers to location of resources (including data).
Compliance code requirement refers to regulatory, industry or security standard that the infrastructure needs to comply with, such as ISO27002.

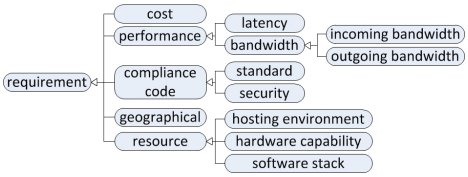

Fig. 3. Hierarchy of infrastructure requirements

A **requirement** can be either **hard requirement** or **soft requirement**. A **hard requirement** is a compulsory requirement which remains invariant over the application's lifecyle – an example might be legislation regulation; a **soft requirement** is a desired requirement which can change or be re-prioritised – for example, it might be budget or performance related. This concept is represented using the **hasRequirementType** property and the **requirementType** emumeration property. A **priority level** data property is defined to indicate the importancy of a **requirement**. This property can be used to measure and calculate weightings during requirement prioritisation and resource filtering process. **Requirements** may depend on each other. For example, UK Data Protection Act (**compliance requirement**) indicates that no data can be processed or stored outside the UK boundary. This translates to a dependency relationship on **geographical requirement**. Figure 4 illustrates the **requirement** ontology.

Infrastructure and **requirement** are core entities in the infrastructure requirement ontology. Every **infrastructure** has at least one **site**. A **site** has one or more **resource groups**. A **resource group** is a set of **resources**. **Requirements** can be applied at different levels of the infrastructure layout: **infrastructure**, **site**, **resource group** or **resource** level. The relationship between **infrastructure** and **requirement** is expressed using the **hasRequirement** property (see Figure 5).

A **restriction** class is defined to identify the conditions or constraints associated with a **requirement**. Each **requirement** has at least one **restriction** which is expressed using **isConstrainedBy** property (see Figure 4).

Cost requirement is constrained by **cost restriction**. A **cost restriction** can be a **total cost** or it can be divided into **compute cost**, **software cost**, **storage cost**, or **bandwidth cost**. Each **cost restriction** is associated with **cost frequency** (per hour, per day) and **cost money** (amount, currency).

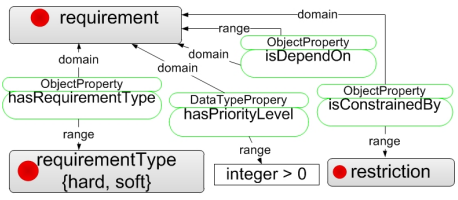

Fig. 4. Requirement

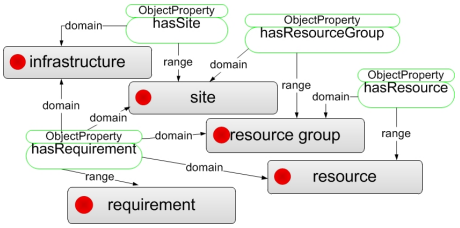

Fig. 5. Infrastructure and requirement

Performance requirement is constrained by performance-related restriction. **Latency performance** is constrained by **latency restriction** (expressed via **hasLatency** property). **Bandwidth performance** is constrained by **bandwidth restriction**. **Bandwidth restriction** indicates minimum amount of bandwidth required using the **hasMinBandwidth** property.

Resource requirement is constrained by resource-related restriction. **Hosting environment** is constrained by **operating system restriction** which indicates the operating system types (via **hasOS** property). **Hardware capability** is constrained by **hardware restrictions**, such as **cpu core restriction** (via **hasMinCPUCore** property), **cpu speed restriction** (via **hasMinCPUSpeed** property), **cpu architecture restrictioin** (via **hasCPUArchitecture** property), **RAM restriction** (via **hasMinMemory** property), and **storage space restriction** (via **hasMinStorageSpace** property). **Software stack** is constrained by **software restriction** which indicates the list of softwares or services that need to be installed on the resource (via **hasSoftware** property).

Geographical requirement is constrained by **location restriction**. **Location restriction** is associated with **hasLocation** property that indicates the location (country or data center location).

Compliance code requirement is constrained by **compliance restriction**, which can be **standard code restriction** – contains the standard's code format or **regulatory restriction** – contains the name of regulation.

3.3 Resource Ontology

The resource ontology, the bottom layer of the proposed model, defines the properties of the resources offered by cloud providers. This layer has been widely investigated elsewhere – see [2] and [8]. These existing ontologies can be applied as the resource ontology in the proposed model. The mapping of infrastructure requirements to the resource ontology can be achieved by using query language [5] – this topic is outside the scope of this paper.

4 Translation from Domain-Specific Ontology

Here consideration is given as to how a domain-specific ontology can be translated to the infrastructure requirements ontology.

For the media transcoding example, application developers need to provision an infrastructure which fulfils the transcoding requirements as well as balances the budget and delivery time constraints. Transcoding requirements, such as video format conversion, frame rate conversion or http streaming, indicate the features or capabilities of a transcoding software that need to provide. Particular software, such as FFmpeg or Rhozet, can be used to perform the transcoding task. However, each software has different system requirements. For example, Rhozet must be run on Windows operating system, whereas FFmpeg can be run on Linux. Using the proposed multi-layer ontology model, domain-specific transcoding requirements can be translated into software requirements, where

each software has associated operating system or hardware dependancy requirements. Moreover, tight delivery deadline requirement may necessitated the use of high-cpu resources. Figure 6 illustrates how the transcoding requirements are translated into infrastructure requirements.

Examples of the ontology relationships between domain-specific and infrastructure layer for the media transcoding application are given as follow: ***Format, codec, device, processing filters*** and ***delivery channel*** *isDependOn* **software stack** which indicates the transcoding software's capabilities or features; ***delivery time*** *isDependOn* **network latency**, **bandwidth** and **hardware capability** as it requires fast bandwidth and high cpu for fast processing.

For the financial services example, the UK Data Protection Act regulatory requirement indicates that the infrastructure resources being provisioned must be located within UK. Two identical mirror infrastructure must be provisioned at different geographical location in order to meet the high availability and high disaster recovery requirements. High bandwidth usage is required as the application service needs to utilise stock market values. The demand of high response time requires high-cpu and high-memory resources for faster computation. The formulation of such requirements are illustrated in Figure 7.

Examples of the ontology relationships for the financial services application are given below: ***Compliance*** *isDependOn* **geographical** because a 'UK Data Protection Act' indicates that no data can be processed outside the UK boundary; ***data*** *isDependOn* **software stack** as it requires third party verification services; ***high availability*** and ***high scalability*** are complex requirements and depend on how infrastructure resources are provisioned. The concepts of **infrastructure**, **site**, **resource group** and **resource** are used to indicate that different geographical sites must be provisioned, and each site *hasRequirement* **latency**, **bandwidth** and **geographical**.

The middle layer of the proposed model serves as an agent that translate the domain-specific ontology onto related infrastructure requirements ontology. This provides an abstract view of high-level requirements from the application's perspective. Infrastructure requirements ontology can then be mapped to resource ontology using ontology query language [5].

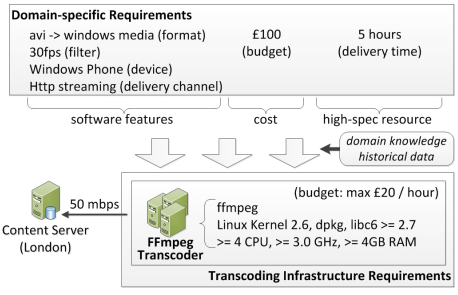

Fig. 6. Media transcoding application

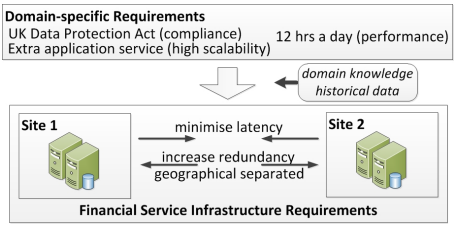

Fig. 7. Financial services application

5 Conclusion and Future Work

The cloud market is developing rapidly with a dynamic environment of providers and products. Searching for suitable resources in such a dynamic environment is challenging. Little attention has been paid to describe a cloud application's requirements at an appropriate level of abstraction. In this paper, an application-centric multi-layer ontology for describing cloud application requirements is proposed. This ontology provides a semantic mechanism for capturing application needs in a language familiar from users' application domains. Two examples are used to illustrate the formulation of application requirements. We hope to enhance the ontology by studying other application domains. We also hope to develop techniques for mapping user requirements into infrastructure constraints. We believe that our approach offers an effective mechanism to compare and select resources from a multi-provider cloud market.

References

1. Wowza Transcoders, `http://www.wowza.com/forums/content.php?23-pre-built-amis-amazon-machine-images` (last accessed: June 30, 2012)
2. Bernstein, D., Vij, D.: Using Semantic Web Ontology for Intercloud Directories and Exchanges. In: International Conference on Internet Computing (2010)
3. Emeakaroha, V.C., Brandic, I., Maurer, M., Dustdar, S.: Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In: 2010 International Conference on High Performance Computing and Simulation, HPCS (July 2010)
4. Galán, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., Vaquero, L.M.: 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, COMSWARE 2009. ACM (2009)
5. Haase, P., Motik, B.: A mapping system for the integration of OWL-DL ontologies. In: Proceedings of the First International Workshop on Interoperability of Heterogeneous Information Systems, IHIS 2005, pp. 9–16. ACM (2005)
6. Harmer, T., Wright, P., Cunningham, C., Hawkins, J., Perrott, R.: An application-centric model for cloud management. In: Proceedings of the 2010 6th World Congress on Services, SERVICES 2010. IEEE Computer Society (2010)
7. Harmer, T., Wright, P., Cunningham, C., Perrott, R.: Provider-independent use of the cloud. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 454–465. Springer, Heidelberg (2009)
8. Moscato, F., Aversa, R., Di Martino, B., Fortis, T., Munteanu, V.: An analysis of mOSAIC ontology for Cloud resources annotation. In: 2011 Federated Conference on Computer Science and Information Systems, FedCSIS (September 2011)
9. Sun, Y.L., Perrott, R., Harmer, T., Cunningham, C., Wright, P.: An SLA Focused Financial Services Infrastructure. In: Proceedings of the 1st International Conference on Cloud Computing Virtualization, Singapore (2010)
10. Wright, P., Harmer, T., Hawkins, J., Sun, Y.L.: A Commodity-Focused Multi-cloud Marketplace Exemplar Application. In: 2011 IEEE International Conference on Cloud Computing (CLOUD) (July 2011)