A Template Description Framework for Services as a Utility for Cloud Brokerage

Li Zhang[1], Frank Fowley[2], Claus Pahl[2]
[1]*Northeastern University, Software College, Shenyang, China*
[2]*IC4, Dublin City University, Dublin, Ireland*
zhangl@swc.neu.edu.cn, [cpahl|ffowley]@computing.dcu.ie

Keywords: Cloud Broker; Service Brokerage; Service Description; Reference Architecture

Abstract: Integration and mediation are two core functions that a cloud service broker needs to perform. The description of services involved plays a central role in this endeavour to enable services to be considered as commoditised utilities. We propose a conceptual framework for a cloud service broker based on two parts: a reference architecture for cloud brokers and a service description template that describes the mediated and integrated cloud services. Structural aspects of that template will be identified, formalised in an ontology and mapped onto a set of sublanguages that can be aligned to the cloud development and deployment process.

1 Introduction

Organisations like Gartner and NIST (Gartner, 2012; NIST, 2013) have identified cloud service brokerage as a key concern for future cloud technology development and research. Cloud brokerage refers to the customisation, aggregation and integration of cloud services, possibly by third parties. Here, we investigate some conceptual problems towards the implementation of a cloud brokerage solution. Integration and mediation are two core functions that a cloud broker needs to perform. We propose a conceptual framework for a cloud broker based on two parts: a reference architecture for cloud brokers and a service description template that describes mediated, i.e. customised, aggregated or integrated cloud services.

Our conceptual reference architecture will extend the NIST cloud architecture specification, taking on board the NIST and Gartner categorisations. The NIST architecture is not sufficient as a common reference framework, as it does not detail how integration between services – horizontally between different providers and vertically between the different cloud layers – can be achieved. Equally do current service modelling and specification approaches (Pahl 2005) not cover adequately the vertical integration across the cloud layers infrastructure (IaaS), platform (PaaS) and software (SaaS) (Konstantinou 2009; Mietzner 2008; Rodero-Merino 2010). We define here a conceptual model for a service description template that would allow the two dimensions to be covered. Structural aspects of that template will be identified, formalised in an ontology and mapped onto a set of sublanguages that can be aligned to the cloud development and deployment process.

The paper is organised as follows. The next Section 2 defines the reference architecture. The following Section 3 introduces and defines the service template. As this concept plays a crucial role, it will be investigated in more depth. Furthermore, we outline a solution architecture for an implementation based on open-source cloud platforms in Section 4 and also discuss some trends, before ending with related work (Section 5) and some conclusions (Section 6).

2 Architecture Framework

As already explained, we propose a conceptual framework for an automated architecture for a cloud service broker based on two parts: a reference architecture for cloud brokers compliant with the NIST proposal and a service description template model that describes mediated cloud services. While descriptions for Web services exist in the format of WSDL, or any semantic extension like SAWSDL or WSMO, some specific requirements for the cloud context need to be addressed. Enabling cross-layer description of concerns and facilitating cloud provisioning and configuration is required (Fehling 2011). Service description templates (and their instances) are

the core artefacts that are used to describe, select, deploy and manage the different cloud resources, which are made available as services in an independent way.

2.1 Architecture and Broker Types

Figure 1 describes the high-level architectural setting. We follow Gartner here in the separation of integration, customisation and aggregation functions of a broker. The broker sits between the consumer and a range of different, independent providers. The role of the service templates is indicated:

- forming the basis of requests by consumers,
- collected from providers and made available for customisation, integration, aggregation by broker.

The detailed structure of the templates (operation, quality, resources, policies as the main description categories) will be explained on later. Note that while the structure of the reference architecture might be easier to agree upon as it brings expected features into a common framework, requirements for the service templates are more difficult to establish as they not only have to describe 'what' is in the architecture, but also 'how' the architectural framework can be converted into an effective solution. Cross-layer support in a federated context (i.e. vertical and horizontal integration support) is the critical challenge (Barrett et al., 2006; Buyya 2010). The reference architecture thus aims to provide integration in two dimensions:

- Horizontal integration across federated clouds
- Vertical integration across the cloud stack layers (IaaS, PaaS, SaaS)

We follow Gartner in their terminology. The Gartner categorisation is illustrated in Fig. 1. NIST follows a similar three-pronged classification. They define a cloud broker as an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers. According to NIST, a cloud broker can provide services in three categories:

- Service Intermediation: A cloud broker enhances a given service by improving some specific capability and providing value-added services to cloud consumers. The improvement can be managing access to cloud services, identity management, performance reporting, enhanced security, etc.
- Service Aggregation: A broker combines and integrates multiple services into one or more new services (Benslimane 2008). The broker provides data integration and ensures secure data transfer between consumer and multiple providers.
- Service Arbitrage: Service arbitrage is similar to service aggregation except that the services being aggregated are not fixed. Arbitrage means a broker has the flexibility to choose services from multiple agencies (Benson 2010). The cloud broker, for example, can use a credit-scoring service to measure and select an agency with the best score.

2.2 Cloud Broker Interfaces and Access

We envisage the cloud consumer to interact with the cloud broker in two ways. Firstly, request submission and cloud service construction: this is pre-deployment and provisioning. Secondly, service monitoring and management: this is post-deploying during the provisioning of the service. Fig. 2 illustrates the two layers:

- An Integration Layer allows the construction (aggregation) of a complex or customised cloud service based on individual, independent services. Data and information that is communicated needs to be integrated through appropriate mediation and transformation techniques. In addition, the integration of wider processes is often necessary, similar to work on document-based service process integration in frameworks such as ebXML.
- A Management Layer allows the monitoring and analysis of broker-provided services. The broker is often the direct contract partner, resulting in the need to deal with service-level agreements (SLAs) and billing/payment issues at that level. Monitoring provides the quality and usage data for both quality and payments concerns.

2.3 Broker Features and Requirements

In order to clarify the requirements for a service description notation for cloud service brokerage, we detail the functions we expect to be performed by the broker across the cloud layers IaaS, PaaS and SaaS – see Figure 3. For the pre-deployment, the following aspects are relevant. Integration, interoperability and portability are the key concerns to support the consumer in creating multi-vendor applications and to move or migrate between vendors (Bernstein 2009; Pahl 2013a; Pahl 2013b).

- Portability can be supported and automated at lower layers by adopting OVF or TOSCA – the latter a standard which specifically supports portability between public and private clouds.
- Expected workload is a concern that should be considered at design/built-time, as service selection and later provisioning need to take this into

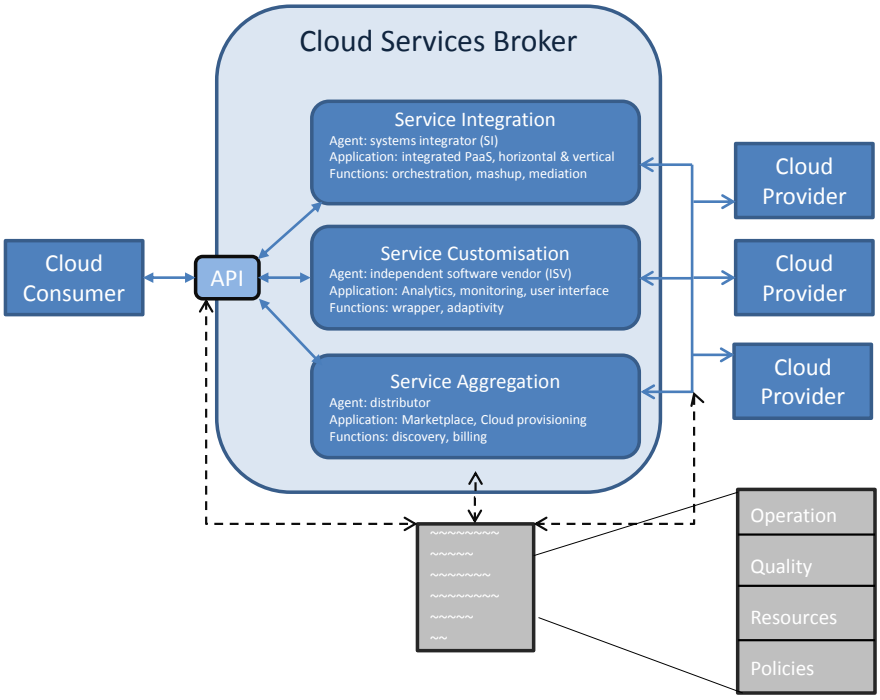

Figure 1: High-level Architectural Cloud Broker Setting.

account. This requires workload to be made explicit at all layers.

- Horizontal interoperability refers to the integration of services within a cloud layer, requiring common interfaces. Standards compatibility is a solution. OCCI is a lifecycle management standard describing APIs for service management.
- Vertical interoperability requires the integration of infrastructure, platform and application needs. Key performance indicators (KPIs) differ across the layers – while at lower infrastructure layers network bandwidth and latency are concerns, further up it is response time or availability per service. However, higher-level ones need to be mapped down to lower layers, and vice versa. Thus, formalised mappings need to be defined.

We see the vertical cloud stack as an implementation of layering as a SOA principle, resulting in a reference architecture with applications (SaaS in cloud), middleware (PaaS in cloud) and infrastructure services (IaaS in cloud). Critical is in vertical cloud stack is the integration of non-functional properties (QoS), i.e., mappings between the layers. For SaaS and PaaS, a relevant metric is for instance the service response time. For IaaS, metrics are based on compute (CPU load), storage (data size) and network (throughput or latency). While not covered here in detail, mappings between layers can be statically fixed. As input, we can used our work on predicting and deriving e.g. PaaS-response time from IaaS-metrics like CPU utilisation or network bandwidth (Zhang, 2013).

For post-deployment management, life cycle support is crucial, covering the configuration and provisioning as well as on-going monitoring and analytics.

- Resource need to be configured, based usually on quality requirements such that the resources match the needs of the consumer. Selection, matching and configuration support is necessary.
- The user requirements and the resulting abstract configuration have to be correlated with the available resources provided at the time of provisioning the services. In addition to the configuration concerns, this requires lower-level input from resource managers and workload balancers.
- Monitoring is an essential part of a provisioning platform. Monitoring solutions are available – commercial and open-source. However, due to the requirement of supporting federated clouds, the need to collect and integrate monitored data from different sources arises.
- Like monitoring, the analysis of data gathered requires interoperability between monitoring sources and the analysis tool. The requirements are similar with respect to standardisation and compatibility, e.g. SLA-related data.

This discussion provides a first requirements list for the service template, discussed in the next section.

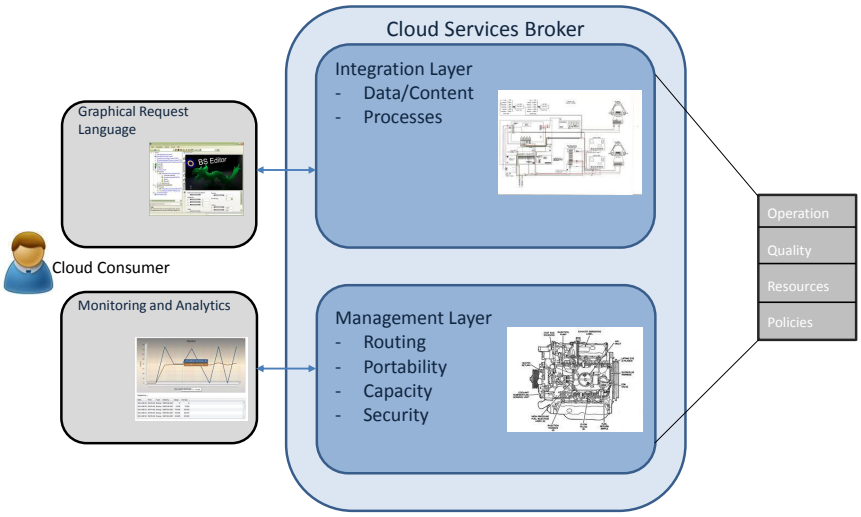

Figure 2: Cloud Broker Interfaces and Access to Functions.

3 Service Description Templates – Structure and Formalisation

A service template is an abstract description of a service's capabilities, in terms of functional and non-functional aspects. Different types of templates to describe the capabilities of cloud resources have been proposed, often under a different name such as Recipes (Chef, Cloudify) (Cloudify, 2012), Blueprints (4CaaSt) (4Caast, 2013) or Manifests (OVF, CompatibleOne) (Compatible One, 2013). In Section 5, see Fig. 7 below, we summarise the language definition challenges and requirements presented in (Pahl et al, 2013). We categorise the service description languages, such as (CloudFoundry, 2013; DeltaCloud, 2013; Jclouds, 2013; Libcloud, 2013; Mosaic, 2013; OpenShift, 2012; Optimis, 2013; simpleAPI, 2013), in terms of the cloud layer support and specific features they provide. We categorise the solutions along the following concerns, which form requirements for a template solution:

- System Type: Multi Cloud API, IaaS Fabric Controller, Open PaaS Solution, Open PaaS Provider.
- Distribution Model: Open Source (for all solutions considered)
- Core Capabilities: Multi-IaaS Support, Multi Language/Framework Support, Multi Stack
- Core Features/Components (development and deployment time): Service Description Language, Native Data Store, Native Message Queue, Programming Model, Elasticity & Scalability, QoS/SLA Monitoring.
- Advanced Features/Components: Service Discovery/Composition, Broker, Marketplace – towards broker and marketplace features.

We can see a trend towards the upper broker and marketplace solutions (Mietzner 2008; Rodero-Merino 2010). These demonstrate to some extent a multi-faceted description with functional and quality aspects. However, a comprehensive coverage is not available. A richer description notation is needed, resulting in additional complexity due to the required multi-layer support, leaving both vertical integration and comprehensive to be addressed.

3.1 Template Structure

We propose a 4-part structure for the service description template: operation, quality, resources and policies. The first two cover service-internal aspects – the functional and non-functional aspects of the service. The remaining two cover more the service-external perspective with the resources required and also the security and compliance requirements. We identify a 4-part hierarchy of concepts, which we will later formalise as the core taxonomy of a service description ontology (Pahl 2007; Papazoglou 2011b).

- Operation – the functional service interface. Its subconcepts follow WSDL in its structure:
  - Service type/interface defines the interface as an abstract data type
  - Operations that provide service functionality
  - Messages and Types to communicate data to and from the service for operation invocations
- Quality – the non-functional properties. Its subconcepts are performance-related concerns:
  - Availability of the service in question
  - Latency/Response Time of service/operations

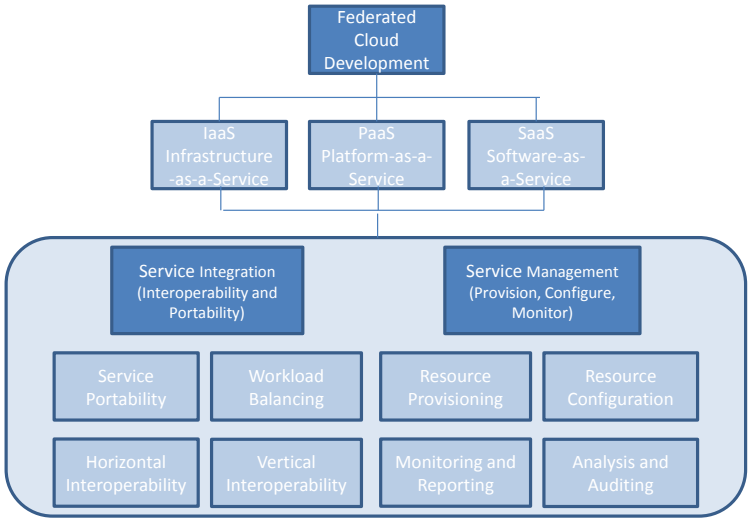

Figure 3: Cloud Broker Features and Capabilities.

  - Bandwidth as another quality concern
- Resources – deployment requirements. Its subconcepts are aspects specific to the provisioning of a service in a PaaS or IaaS environment:
  - Average/peak workload requirements is a typical example for a configuration requirement
- Policies – the business agreements and constraints. Its subconcepts are:
  - Security and Privacy concerns in terms of confidentiality, integrity or authenticity
  - Compliance and Governance as further rules to describe to correct alignment of a service to its execution environment

3.2 Formalisation

We propose to formalise the service template as an ontology. An ontology is a conceptual model formulated in a formal description language. We utilised in this context the core of the Semantic Web technology, the Web Ontology Language OWL. This language is based on description logics, i.e. has a formal definition and is supported by various tools including reasoners that can even be used at runtime. Using semantic technologies brings us closer to interoperability of meaningful descriptions in an Internet context.

An ontology is a representation of concepts (or classes) that are defined in terms of their properties, i.e. relationships to other concepts. Concepts can be instantiated into concrete values. Properties can be object or data valued, depending on whether they refer to another complex object (concept – an example is the relationship of a service to another service through invocation or inheritance) or data (an example is the performance of a service as a numeric value). Concepts can be arranged into a hierarchy using a sub/superconcept relationship. This forms the taxonomy of the ontology. Our ontology has a root concept, which is 'ServiceTemplate'. Using an 'isPartOf' relationship, the aspects Operation, Quality, Resources and Policies are connected to ServiceTemplate. Individual concerns, such as ServiceType or Message for Operation or Availability for Quality, are the subconcepts of the respective four core aspects. These properties define a basic hierarchy. However, we need a richer ontology that addresses the requirements outlined earlier, see Figure 4.

- In terms of interfaces and operations, we need to distinguish the different layers – as ontology subconcepts of the Operation concept. For IaaS-Operation, we define a service API with the corresponding operations that is OCCI-compliant. While VM image management is relatively uniform (thus, OCCI is possible as a standard), at the PaaS layer, we need to provide for standard tools like databases or application server. For instance a Tomcat server could be modelled as an instance of a Web application server concept. Standardised solutions in terms of APIs/operations would then only be specific to a tool category.
- Quality concerns (KPIs for the different layers) need to be differentiated. We assume each quality criterion to be qualified by an attribute (data-valued property) that indicates the relative cloud layer to which it applies.
- Required resources can be explicitly or abstractly specified by referring to other templates that capture the required resource. This is facilitated by a special resources relationship. The need for formality becomes clear as circular resource relation-

ships cannot be tolerated and need to be detected.

- The Policies part needs to link to a special kind of entities: rules. We propose to utilise the SWRL language (Semantic Web Rule Language). The specified rules can be evaluated at build time and dynamically verified.

The ontology is meant to be extensible for particular circumstances. For concrete resources, instances are attached. This clarifies the definition of a template – a template is concept-level in terms of the ontology and can be instantiated by providing instance-level values.

Templates can be generated from other resources and provide input to further processing. Based on SLAs or other legal agreements, some operational, quality, and policy requirements can be derived that would define a partial template. Service integration plans need to be generated in many cases where a single service will not satisfy the need. A service process needs to be orchestrated in a classical Web service manner, associated to an integration process connecting abstract resources.

3.3 Sublanguages

While we have proposed a comprehensive ontology capturing all conceptual aspects of the service template language framework, in order to tailor this to the needs of different actors and components of the reference architecture, a facilitation through different languages would be beneficial. As core technical languages (Figure 5) we can identify:

- Service Template Definition Language (Operation and Quality) addresses service internal aspects
- Service Template Configuration/Instantiation Language (Resources) addresses first type of external aspects
- Service Template Constraints Language (Policies) also addresses external aspects, but, as explained, a rule language rather than an ontology language needs to be facilitated here.

These core languages can be complemented by two template manipulation operators – core composition and non-compositional manipulation (Benslimane 2008), both possibly supported by a graphical request language:

- Service Template Composition Language (Integration Processes) is singled out to deal with service aggregation by supporting a range of standard composition operators to allow services to be orchestrated into a process across different cloud providers (Pahl 2007).
- Service Template Manipulation Language provides non-compositional template manipulation. This could include the refinement/specialisation of a template, or the merger or restriction.
- Graphical Request Specification Language. Requests for a specific service are expressed by end-users. To facilitate access for different users with different degrees of technical knowledge, we suggest a graphical request language visualising the template structure.

The implementation shall follow the ontology operator calculus for architecture manipulation from (Pahl 2007) on ontology-based architecture and pattern languages that combine composition and abstraction mechanisms. The conceptual core of the languages, particularly for the composition and manipulation language is an operator calculus, including operations such as match/select, merge, restrict, refine, behaviourally compose, abstract/instantiate, to support the following core manipulation and composition activities:

- Refinement, i.e. the derivation of a related service description that preserve functional properties (the scope of the service can be tailored through restrict and union operations).
- Instantiation, i.e. providing concrete values for an abstract, concept-level template in order to denote a concrete service.
- Composition, i.e. the use of aggregation operators such as sequence, selection and iteration to orchestrate a selection of services to form a composite service process serving a more complex goal.

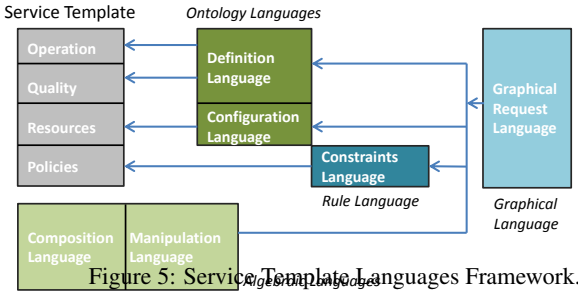

Figure 5: Service Template Languages Framework.

4 Architecture and Validation

An implementation architecture for the service templates is the brokerage test-bed currently developed at IC4 (a cloud research centre at our university), where a combination of OpenStack as the IaaS solution and CompatibleOne as the core PaaS broker

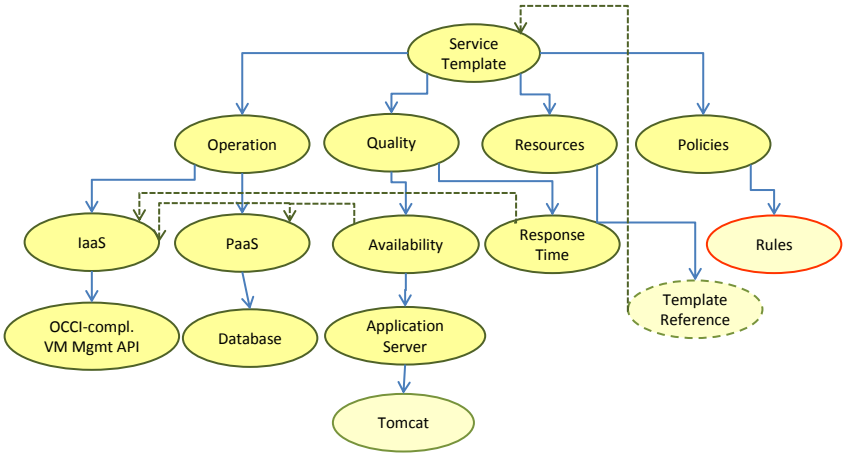

Figure 4: Service Template Ontology.

solution is being developed. This serves us as an implementation to investigate the vertical integration of IaaS, core PaaS and broker layers. While CompatibleOne is a broker, the support implemented does not satisfy the requirements of our needs here – see Sections 2.1 for a CompatibleOne review and 2.3 for a requirements discussion. The following architecture (Fig. 6) describes the architectural setup of OpenStack and Compatible One that we propose to be used as the experimentation environment.

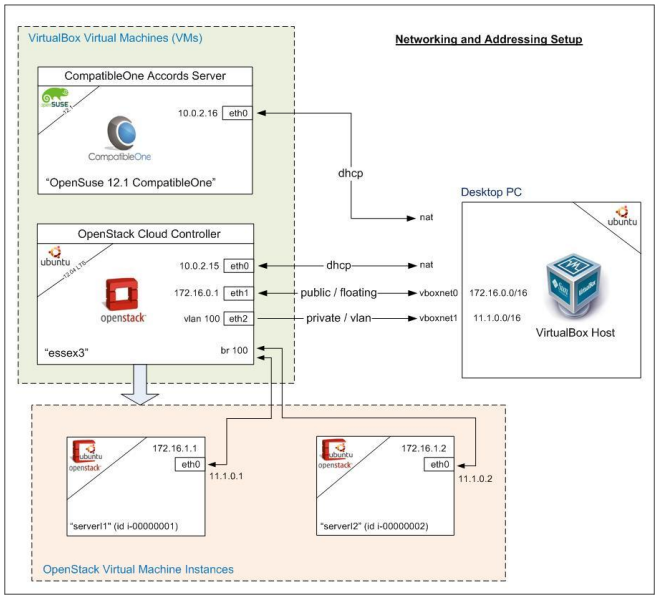

Figure 6: OpenStack and CompatibleOne Architecture

CompatibleOne (Compatible One, 2013) is an academic and industry consortium that set out to provide a framework to allow for portability and interoperability of cloud software. The project focusses on producing a broker platform following the Gartner definition of a cloud broker to provide Intermediation, Aggregation and Arbitration of cloud services. Such a broker could allow for cloud service consumers requirements to be mapped to cloud vendors' resources. The project is committed to the OCCI standard. To date, the project has produced the CORDS information model and the ACCORDS software platform.

- The CORDS model is as an open, object-oriented, OCCI-compliant representation of cloud computing entities and can be used to build complex cloud middleware.
- The ACCORDS platform is a suite of cloud middleware programs that provide broker services. Using ACCORDS, a customer can describe service requirements in a CORDS-compliant format. These can be matched to available provider offerings and then ultimately provisioned at run-time by interacting with the actual vendor resources.

The model includes representations for service level agreements, pricing structures and transaction management for usage billing.

OpenStack (OpenStack, 2012) is a cloud enabling solution that transforms a physical hardware infrastructure into an Infrastructure as a service (IaaS). The platform is a Cloud Platform or IaaS Cloud Fabric controller. The platform can be used to run cloud compute and storage infrastructure. The OpenStack open source siolution makes its services available through Amazon EC2/S3 compatible APIs and hence client tools written for AWS can be used with OpenStack clouds. Five main services are exposed by OpenStack: Compute (Nova), Storage (Swift), Imaging (Glance), Identity (Keystone) and Network management (Quantum).

The adequacy of the proposed service template model can be validated by discussing some scenarios as implementations in the above testbed architecture:

- Customisation, Aggregation, Integration: these are the three broker types, which need to be supported by adequate operators.

  We have proposed an ontology-based operator calculus that provides suitable support. Refinement allows customisation. The composition operators that we propose allow aggregation. And integration is facilitated by allowing semantic integration through a rich ontology and a rules-based approach to control the integration into specific environments. Thus, specifically the manipulation and composition operators are necessary to fully support service brokerage. An ontology-based solution helps with integrating multi-facetted aspects into a coherent manipulation and reasoning framework.

- CORDS and ACCORDS integration: the CORDS model with its OCCI-based description layer provides a standards-based platform to enable utilisation in a concrete OCCI-compliant IaaS solution.

  The CORDS model relating to the ACCORDS platform features can be fully represented in terms of the service template notation. ACCORDS services are instantiated service templates. The OCCI-compliant OpenStack is the execution platform. With CompatibleOne, we have used one of the most advanced broker platforms available (we discuss 4CaaSt, another advanced solution, in the next section) based on our comparison in Fig. 7 earlier. Thus, we can be confident that the vertical layering problem can be addressed for a rich, multi-facetted description template.

While we have not fully implemented the proposed service template model as an extension of the CORDS matching, our experiments with CompatibleOne support the feasibility of a full integration of the two cloud solutions. The experiments validate the scope that a service description template has to cover for vertical integration as proposed in Section 4.

5 Related Work

We select 13 specification notations here for a more in-depth description including (CloudFoundry, 2013; DeltaCloud, 2013; Jclouds, 2013; Libcloud, 2013; Mosaic, 2013; **?**; OpenShift, 2012; Optimis, 2013; simpleAPI, 2013). Fig. 7 summarises a range of solutions ordered from bottom to top based on the focus in the cloud technology stack (from basic IaaS to advanced SaaS marketplace support). We compare these in terms of their capabilities and components as comparison categories as discussed in Section 3. A more detailed, general discussion can be found in (Grozev, 2012), which also covers systems like mOSAIC or jclouds. Our work here is not specific to any of the intercloud architecture types identified there. Our discussion is also more specific to languages and service description, whereas (Grozev, 2012) covers wider concerns like location-awareness or pricing.

An ontology-based foundation enables reasoning support. It also acts as an integration, e.g. via ODM framework with MOF/UML-style notations and via the predicate logic link to other logic-based languages. In this respect, our template approach is a step ahead compared with e.g. the Blueprint approach of 4CaaSt or the recipes techniques in Cloudify. Ongoing work in the EU Broker@Cloud project uses an extension of the RDF-based version of USDL (Kourtesis, 2013), which demonstrates the relevance of semantic technologies for integration and reasoning across platforms, but here our full OWL goes beyond the RDF-based USDL formalisation.

The two most advanced ones in terms of brokerage – 4CaaSt and CompatibleOne – shall be discussed. The 4CaaSt Blueprint approach (Nguyen 2011; Papazoglou 2011b) supports through its different languages a range of cloud brokerage activities. Blueprints are initially abstract specifications, which can be resolved, i.e. mapped to lower cloud stack layers until the specifications are finally deployed using a standard configuration and deployment manager. Resources and Services are described in a Blueprint (BP), which is an abstract description of what needs to be resolved into infrastructure entities. BPs are stored and managed in a BP repository via a REST API. A BP is resolved when all requirements are fulfilled by another BP, via the Resolution Engine. This is a service orchestration feature. A number of XML blueprint languages are supported: BDL Description Language (for cloud operations and capabilities), BCL Constraint Language (for SLA parameters definition), BML Manipulation Language (for operations, e.g. service match, merge, compose, delete), BRL Request Language (for user and developer request definitions). Blueprints are suitable to support simple marketplace offering single, isolated services as a product, but lack the range of brokerage activities in their toolkit. This entails a further complication – compositional brokerage needs to be mapped down the layer, which is non-trivial as particularly quality aspects are non-compositional in nature.

The CompatibleOne manifests suffer from similar limitations. The units of a CompatibleOne Service Manifest are Image and Infrastructure. An Image consists on a System (the base OS) and a Package (reflecting the software stack configuration). Infras-

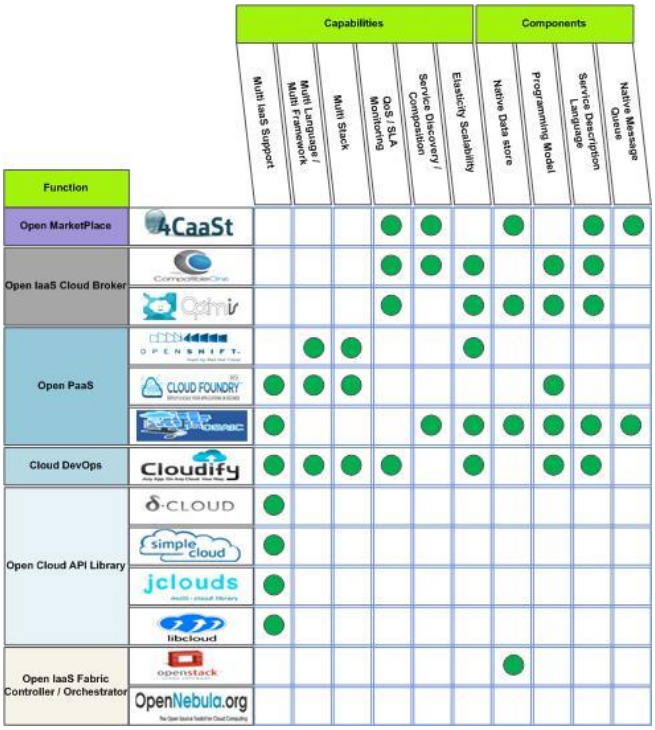

Figure 7: Summary Service Description Comparison.

tructure covers the classical IaaS concerns Storage, Compute and Network. An Image is a description of a manual build of an application, used by development and configuraton tools (devops, e.g. puppet or chef). An Image has an agent that is embedded in a VM and runs on startup. Agent is a script to run required configurations, set up monitoring probes, or download required components. Cordscript is proprietary CompatibleOne scripting language for expressing configuration actions. These support metadata handling within VMs, managing required linkages between VMs, and setting up monitoring probes in VMs. Cordscript is a statement-oriented language. A feature under development in CompatibleOne is a cloud-implementable graph of OCCI category instances. Particularly well solved is the standards-compliancy at the lower layer (making it a good testbed), but the manifests only cover core concerns.

The consumer expresses provisioning requirements by creating a CORDS manifest. This contains attributes like ID and name. Key element is a provisioning plan involving nodes and regions affected. The node configuration topology includes image and infrastructure requirements (compute, storage, network). Actions for configurations and releases can be specified. Other information concerns a) account, pricing and invoicing and b) security settings. CORDS is configuration, provisioning and deployment oriented, where our template adds quality beyond security and governance through policies. The templates can be seen as higher-level (abstract) specifications on top of e.g. a CORDS manifest.

6 Conclusions

Standards and compatibility are effectively the only way to make a truly interoperable cloud service broker work. We can note a trend towards cloud marketplaces utilising brokerage facilities. The proposed solution in this paper is a first step in this direction that needs to facilitate language support in terms of request specification languages and also template manipulation operators that will support central marketplace functions (like matching and selection) – based on a the reference architecture is the other solution that defines how architectural components interact and how the service templates are processed in this context. Abstract descriptions of cloud services are the key to a service brokerage solution allowing to consider services as commoditised utilities that can be brokered. We presented a conceptual model for the service templates. We have defined the core properties as an ontological model. We have suggested a range of languages to support this feature.

The proposed OpenStack and CompatibleOne ar-

chitecture should serve as an adequate basis to develop a solution based on the languages for service templates as broker implementation, which remains future work. On the theoretical side, the full specification of the operator calculus needs to be completed, but can follow (Pahl 2007) here.

Acknowledgments.

This research has been supported by the Irish Centre for Cloud Computing and Commerce, an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

REFERENCES

4Caast. 4CaaSt PaaS Cloud Platform. http://4caast.morfeo-project.org/. 2013.

R. Barrett, L. M. Patcas, C. Pahl and J. Murphy. Model Driven Distribution Pattern Design for Dynamic Web Service Compositions. International Conference on Web Engineering ICWE'06. ACM Press. 2006.

T. Benson, et al. Peeking into the Cloud: Toward User-Driven Cloud Management. Clouds 2010, 2010.

D. Benslimane, S. Dustdar, A. Sheth. Services Mashups: The New Generation of Web Applications. Internet Computing, vol.12, no.5, pp.13-15, 2008.

D. Bernstein et al. Blueprint for the Inter-cloud: Protocols and Formats for Cloud Computing Interoperability. Intl Conf Internet and Web Appl and Services. 2009.

R. Buyya, R. Ranjan, R.N. Calheiros. Intercloud: Utility-Oriented Federation of Cloud Computing Environments For Scaling of Application Services. Intl Conf on Alg and Arch for Parallel Processing. 2010.

Cloud Foundry. Open Source PaaS Cloud Provider Interface. http://www.cloudfoundry.org/. 2013.

Cloudify. Cloudify Open PaaS Stack. http://www.cloudifysource.org/. 2012.

CompatibleOne. Open Source Cloud Broker. http://www.compatibleone.org/. 2013.

DeltaCloud. Deltacloud REST cloud abstraction API. http://deltacloud.apache.org/. 2013.

C. Fehling, R. Mietzner. Composite as a Service: Cloud Application Structures, Provisioning, and Management. IT - Information Technology: Vol. 53, No. 4, pp. 188-194. 2011.

F. Fowley, C. Pahl and L. Zhang. Cloud Brokerage Architecture – State-of-the-Art and Challenges. Workshop on Cloud Service Brokerage CSB2013. 2013.

Gartner - Cloud Services Brokerage. Gartner Research. http://www.gartner.com/technology/research/cloud-computing/cloud-services-brokerage.jsp. 2012.

N. Grozev and R. Buyya. Inter-Cloud architectures and application brokering: taxonomy and survey. Softw: Pract. Exper.. doi: 10.1002/spe.2168. 2012.

Jclouds. jclouds Java and Clojure Cloud API. http://www.jclouds.org/. 2013.

A. V. Konstantinou et. al. An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds. 3rd International Workshop on Virtualization Technologies in Distributed Computing. June 2009.

D. Kourtesis, K. Bratanis, A. Friesen, Y. Verginadis, A.J.H. Simons, A. Rossini, A. Schwichtenberg, P. Gouva. Brokerage for Quality Assurance and Optimisation of Cloud Services: an Analysis of Key Requirements. ICSOC Cloud Service Brokerage Workshop CSB2013. Springer. 2013.

Libcloud. Apache Libcloud Python library. http://libcloud.apache.org/. 2013.

R. Mietzner, F. Leymann, M. Papazoglou. Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns. 3rd Internet and Web Appl and Services. 2008.

Mosaic. mOSAIC Multiple Cloud API. http://www.mosaic-cloud.eu/. 2013.

NIST Cloud Computing Reference Architecture. NIST. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505. Sept. 2011.

D.K. Nguyen, et al. Blueprint Template Support for Cloud-Based Service Engineering. In Proceedings of the Service-Wave'11, Poznan, Poland, October 2011.

3 OpenNebula. OpenNebula - Open Source Data Center Virtualization. http://opennebula.org/. 2012.

OpenShift. Cloud computing platform as a service. https://openshift.redhat.com/. 2012.

OpenStack. OpenStack Open Source Cloud Computing Software. http://www.openstack.org/. 2012.

Optimis. Optimis - Optimized Infrastructure Services. http://www.optimis-project.eu/. 2013.

P. Jamshidi, A. Ahmad, C. Pahl. Cloud Migration Research: A Systematic Review IEEE Transactions on Cloud Computing. 2014.

C. Pahl, S. Giesecke and W. Hasselbring. Ontology-based Modelling of Architectural Styles. Information and Software Technology (IST). 1(12): 1739-1749. 2009.

C. Pahl, H. Xiong. Migration to PaaS Clouds - Migration Process and Architectural Concerns. IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA 2103. IEEE. 2013

C. Pahl, H. Xiong, R. Walshe. A Comparison of On-premise to Cloud Migration Approaches. European Conference on Service-Oriented and Cloud Computing ESOCC 2013. Springer LNCS. 2013

M. P. Papazoglou, W.J. van den Heuvel. Blueprinting the Cloud. IEEE Internet Computing, November 2011.

L. Rodero-Merino et al. From Infrastructure Delivery to Service Management in Clouds. Future Generation Computer Systems, vol. 26, pp. 226-240. Oct. 2010.

simpleAPI. Simple API for XML. http://en.wikipedia.org/wiki/Simple_API_for_XML. 2013.

L. Zhang, B. Zhang, C. Pahl, L. Xu, Z. Zhu. Personalized Quality Prediction for Dynamic Service Management based on Invocation Patterns. Intl Conference on Service Oriented Computing ICSOC 2013. 2013