# A Classification and Comparison Framework for Cloud Service Brokerage Architectures

Frank Fowley, Claus Pahl, Pooyan Jamshidi, Daren Fang, Xiaodong Liu

**Abstract**—Cloud service brokerage and related management and marketplace concepts have been identified as key concerns for future cloud technology development and research. Cloud service management is an important building block of cloud architectures that can be extended to act as a broker service layer between consumers and providers, and even to form marketplace services. We present a 3-pronged classification and comparison framework for broker platforms and applications. A range of specific broker development concerns like architecture, programming and quality are investigated. Based on this framework, selected management, brokerage and marketplace solutions will be compared, not only to demonstrate the utility of the framework, but also to identify challenges and wider research objectives based on an identification of cloud broker architecture concerns and technical requirements for service brokerage solutions. We also discuss emerging cloud architecture concerns such as commoditisation and federation of integrated, vertical cloud stacks.

**Index Terms**—Cloud Broker; Service Brokerage; Architecture Patterns; Cloud Broker Classification; Service Management.

## 1 Introduction

Several organisations active in the cloud technology area, such as Gartner, Forrester and NIST [17], [14], [27], have identified cloud service brokerage as an important business model, but also as an architectural challenge that needs to explore how to best construct broker applications on top of suitable platforms. A cloud service broker manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers [14]. Cloud service management, an important building block of cloud architectures, can be extended to act as a brokerage layer between consumers and providers, and to even form marketplaces. Architecture, development and quality concerns are key enablers of any service brokerage solution that intermediates between different providers by integrating, aggregating and customising their individual services.

Our main contribution is a classification framework for cloud service management, brokerage and marketplace solutions that use some form of intermediation mechanism that goes beyond Gartner, Forrester and NIST and aims to resolve different existing interpretations. In addition to providing a comprehensive definition, our framework allows us to classify and compare broker solutions. We look at architectural concerns, i.e., software-based intermediation solutions that support brokerage as a business model. We distinguish between an application and a platform perspective. Marketplaces, like app stores, are broker applications that assemble and provide end-user services. These broker applications often run on cloud platforms with specific broker-oriented capabilities.

This classification framework is based on a broader classification in terms of capability and feature categories, architectural patterns and a more refined, descriptive scheme specifically looking at architecture, language and quality as technical aspects. The emphasis here is on a systematic determination of classification categories based on different cloud and software bodies of knowledge. We compare selected cloud service management and brokerage solutions to illustrate and evaluate the framework, and also to derive trends and challenges from this comparison. The selection of subjects is again systematic, aiming to identify technically advanced solutions that cover the cloud service broker space comprehensively to ensure that the adequacy and the completeness of the framework is properly evaluated. Based on an identification of cloud broker architecture patterns for service brokerage solutions, we discuss challenges.

Such a dedicated framework does not exist for cloud brokers and goes beyond existing service taxonomies such as [19]. Grozev and Buyya [18] go beyond this and introduce a taxonomy and compare solutions. However, their effort is focussed on a taxonomy for inter-cloud architectures. Their comparison scheme uses 5 basic aspects (type/organisation, architecture, brokering approach, application type, awareness). The definitions provided by Gartner, Forrester and NIST [17], [14], [27] are also high-level and not suitable for a detailed classification and comparison. In [15], the need for a multi-dimensional classification is recognised and a basic multi-facetted framework is proposed. However, it does not clearly distinguish

- *F. Fowley, C. Pahl and P. Jamshidi are with the Irish Centre for Cloud Computing and Commerce IC4, Dublin City University, Dublin 9, Ireland.*
  *E-mail: Claus.Pahl@unibz.it (contact author)*
- *D. Fang and X. Liu are with the School of Computing, Edinburgh Napier University, Edinburgh, UK.*

between application and platform dimensions, lacks a detailed vocabulary at concept instance level, nor does it provide a systematic identification, extraction, modelling and evaluation of the framework.

The paper is organised as follows. Cloud service management and brokerage is introduced and analysed in Section 2. Section 3 defines the comparison framework. In Section 4, we apply the comparison framework to selected solutions and evaluate its fitness for purpose. This investigation leads to a broader research challenges discussion, before ending with conclusions in Section 5.

## 2 Cloud Service Brokerage - Literature Review and Analysis

We start with a review of existing cloud service brokerage definitions to obtain a consolidated list of broker types (Section 2.1) and a list of their typical capabilities (Section 2.2). We revisit the broker types to define the scope of emerging architectural patterns for cloud service brokers (Section 2.3). Then, we investigate the cloud stack deployment models – namely, infrastructure (IaaS) and platform/applications (PaaS/SaaS) - to determine layer-specific concerns (Section 2.4). Finally, we discuss the perspectives of different user types (Section 2.5). We adopt the NIST definitions [27] refering to capabilities provided covering processing, storage, networking resources (IaaS), deployment of applications created using programming languages, libraries and tools (PaaS) and using a providers applications (SaaS).

### 2.1 Brokerage - Definitions and Analysis

We begin by focusing on a brokerage-as-a-service delivery model based on intermediation as a technical principle. Advanced multi-cloud management platforms, as well as brokers as marketplaces, can be considered as specific models in the wider brokerage space. Forrester, Gartner and NIST define Cloud Service Brokerage (CSB) in different ways [14], [17], [27]. Gartner and NIST both follow a three-pronged classification. They define a cloud broker as an entity that manages the use, performance and delivery of cloud services and negotiates between providers and consumers, i.e., intermediates between both [14].

In this initial overview of key concepts, we start with Gartner [17]. For each broker type, we name the typical broker role (agent), the application types (application) and the typical functionality of brokered (or mediated) services (functions).

- Aggregation is about delivering two or more services to possibly many consumers, not necessarily providing new functionality, integration or customisation, but typically offering centralised management of SLAs and security.
  - Agent: distributor
  - Application: marketplace, cloud provisioning
  - Functions: discovery, billing, marketplaces
- Customisation is about altering or adding capabilities in order to improve the service functionality and provide enhanced service analytics.
  - Agent: independent software vendor (ISV)
  - Application: analytics, monitoring, interface
  - Functions: wrapper, adaptivity
- Integration involves making independent services work together as a combined offering. This could be the interworking between layers of the vertical cloud stack, or could involve the data/process integration within a single layer. Techniques such as transformation, mediation and orchestration are the classic solutions.
  - Agent: systems integrator (SI)
  - Application: integrated PaaS
  - Functions: orchestration, mashup, mediation

NIST uses aggregation, arbitrage and intermediation as the three core broker types [27]. NIST and Gartner agree on the importance of aggregation. Some further commonalities exist. NIST intermediation and Gartner customisation focus on enhancing existing services. NIST arbitration and Gartner integration both consider the flexible mediation and integration of different systems. Otherwise, the views diverge. NIST includes arbitrage (which supports dynamic pricing in a cloud service marketplace), whereas Gartner does not. NIST intermediation differs from Gartner integration. Intermediation between consumers and providers, in the NIST understanding, includes a range of capabilities (SLA management, invoicing/billing consolidation) as opposed to a merely technical integration.

- Gartner's classification is building on traditional IT roles, which can be seen as a limitation. The role of the aggregation broker matches the traditional distributor role, the integration broker role aligns with a systems integrator and, finally, the customization broker corresponds to an independent software vendor role – note that we have singled out the agent in the Gartner summary.
- NIST's terminology is not undisputed either. A service intermediary should be a party with no commercial aims, i.e., no consumer cloud policies should be affected by the commercial bias of an intermediary. An intermediary can play the role of a broker (receiving a commission is included in the activity), but might choose not to. Note that NIST defines five actors (including brokers) in its Cloud Computing Reference Architecture, which we will simplify later (see Section 2.4).

Forrester starts with the assumption that the cloud broker model offers IT and telecom service providers and other vendors the opportunity to overcome the rapid commoditisation of their existing services business and build a sustainable service delivery model. A simple broker model compares similar cloud provider

options and dynamically provisions selected services based on the actual spot prices of these resources. A full broker goes beyond this to include cloud bursting, i.e., the dynamic relocation of workloads from private environments to cloud providers and vice versa. Forrester's model is based on three core cloud models [39]: tool vendor (software focus), infrastructure provider (infrastructure focus) and cloud builder (consultancy focus). A cloud broker is defined as the combination of three capabilities arising from three core models:

- SaaS Provider - combines software and infrastructure focus with hosting and management.
- Value-Added Reseller - combines software and consultancy focus with management and also customisation and integration capabilities.
- Integrator - combines infrastructure and consultancy focus with integration and hosting.

We use these definitions to extract five core broker types and a vocabulary of capabilities and features.

### 2.2 Broker Mediation Construction - Capabilities and Features

The broker architecture needs to be divided into two layers – broker platform and broker application:

- The platform is the implementation platform on which a broker application is implemented. This platform can be provided 'as-a-service'. The platform provides a range of services to construct the application through intermediation techniques.
- The application provides a concrete broker – possibly targeting a specific vertical sector or a specific service type (e.g., for a cloud delivery model). The broker application is constructed using the platform services, providing features such as SLA management, a service catalogue, service provisioning, including self-service access, as well as user authentication and authorisation.

Several tools specifically target this architecture. We will review open-source platforms later. The commercial space also provides advanced solutions that validate the relevance of this architectural setting[1].

From the discussion in Section 2.1 based on the different definitions from NIST and Gartner and from an analysis of features that commercial broker platforms provide, we extract five rather than the usual three *core broker capabilities* following [11], see Table 1. We started with capabilities and features from Section 2.1 that we then validated using the descriptions of the commercial tools. These reflect how the different mediated services are constructed, which is part of the platform perspective. The capabilities referred to by Forrester (such as hosting or management) can be mapped to integration and intermediation. We defined these capabilities and associated capability types. *Composition* as a type combines application services or application and platform services. *Management* refers to platform-provided management capabilities. *Adaptation* refers to a specialisation of a service to meet a specific user profile. *Distribution* is involved if services originate from different cloud providers.

To further describe the features of a broker for a multi-cloud or a federated cloud setting beyond the capabilities definition, we need to provide a more exhaustive list of lower-level *features* that are used to create SLA and service management, self-service user access and authorisation functions. The following are extracted from the list given in [39], supplemented by other sources such as the features offered by commercial broker platforms. They are associated with a set of feature categories – which are the relevant broker capabilities they apply to, as well as the management view from the capability types, see Table 2.

### 2.3 Management, Broker Platform and Marketplace - Scope and Patterns

Cloud brokerage applications are built up on existing virtualisation techniques, cloud platforms, and IaaS/PaaS/SaaS offerings. Emerging from the earlier discussion, we can single out three architecture patterns that cover the wider brokerage platform and application space, and that frame the broker capabilities. These can help to put broker applications into context in terms of their key application direction. Here, the description of the architecture patterns does not describe some concrete structural or behavioural architecture, but rather the features and objectives of each cloud service architecture type, see Table 3. These patterns are characterised by technical features. However, they should not be considered as broker layers. For example, the Cloud Management pattern may include a billing integration feature, whereas the Broker Platform pattern may not.

Management brokers could also be called internal brokers as their main purpose is often managing an internal service catalogue. On the other hand, classical brokers typically mediate between customers and externally provided services. This perspective is complementary to the broker platform capabilities, which focuses on the broker construction only, but not the objective embedded in the architecture. The discussion below will show that a fine-grained characterisation of the types of cloud brokerage applications, even beyond these three, is necessary to identify and distinguish specific challenges. We look at open-source solutions (or solutions provided by publicly funded projects) as these are well-documented.

1. Examples, in no particular order, are Jamcracker (www.jamcracker.com/solutions), Vordel (www.axway.com/vordel-products), Gravitant (www.gravitant.com), AppDirect (www.appdirect.com) or ComputeNext (www.computenext.com).

TABLE 1
Broker Capabilities.

| *Broker Capability* | *Capability Type* | *Definition* |
|---|---|---|
| Aggregration | Composition & (internal) Management | delivering combined and centrally managed services to many consumers, not necessarily providing new functionality, integration or customisation. |
| Customisation | Adaptation | altering or adding capabilities to change or improve, enhance and analyse the service function. |
| Intermediation | Composition & (external) Management | intermediation between cloud consumers and providers including advanced capabilities (SLA management, invoicing/billing consolidation). |
| Integration | Distribution & Composition | building independent services and data into a combined offering – often as an integration of a vertical cloud stack or data/process integration within a layer through transformation, mediation and orchestration. |
| Arbitrage | Distribution | flexible composition of service chosen possibly from multiple providers selected statically or dynamically based on technical as well as cost aspects. |

TABLE 2
Broker Features.

| *Feature Category* | *Features*[1] |
|---|---|
| Aggregation Integration Customisation | service orchestration, service catalogue management, provider contracting, resource selection, identity management, user authentication and authorisation, security management, interoperability |
| Intermediation Management Arbitrage | SLA compliance, self-service enablement, metering, dashboards, monitoring, helpdesk support, business transaction monitoring, billing/invocing consolidation, dynamic pricing |
| Dynamic Integration | service operations, dynamic routing, capacity management, dynamic orchestration, performance management, life migration, cloud bursting, replication |
| Workload Management | workload classification, capacity planning, price prediction, metering, billing and chargeback, performance management, configuration management, |

*Note 1*: Extracted from the categorised feature list, Fig. 4, p.9 presented in [39], augmented and validated by feature descriptions used in the selected commercial products listed in Footnote 1.

TABLE 3
Broker Scope and Patterns.

| *Pattern* | *Scope Definition* | *Architecture* |
|---|---|---|
| Cloud Broker Platform | Supports the broker activity types discussed earlier – such as aggregation, customisation, integration etc. – which needs a specific language to describe services in a uniform way and to define the integration mechanism. | Broker Platform: The origin of this is the common intermediary/broker pattern from software design patterns, applied to a cloud setting. |
| Cloud Service Management | Supports the design, deployment, provisioning and monitoring of cloud resources, e.g. through management portals. This is an extension of the core lifecycle management (LCM), adding monitoring features or graphical forms of interaction. Rudimentary features for the integration of compatible services can be provided. | Broker Application: A management layer is often identified in cloud architecture that facilitates efficient and scalable provisioning in a number of the platforms reviewed below. Required capabilities include intermediation and aggregation. |
| Cloud Marketplace | Builds up on intermediary/broker platform to provide a marketplace to bring providers and customers together. Again, service description for core and integrated services plays a role for functionality and technical quality aspects. Consolidated pricing and invoicing should here be included. | Broker Application: Marketplaces for apps are omnipresent and this marketplace pattern is a reflection of upcoming cloud-specific marketplaces, which will be discussed in more detail in Section 5. Required capabilities include customisation, arbitrage and often aggregation. |

### 2.4 Layer-specific Requirements Analysis

We now investigate the possible impact of the different cloud layers, IaaS and PaaS/SaaS, on cloud service broker requirements, based on generic, layer-agnostic concerns arising from the definitions above. Brokers often target service at a specific layer and, consequently, have to deal with various cloud layer-specific concerns [5]. We identify requirements and refer to the earlier broker features such as replication, migration or orchestration here.

Specific concerns for the IaaS layer can be described:

- A key requirement for the IaaS model is elasticity management – see workload management and dynamic integration capabilities. With techniques such as replication, provisioned services can be scaled. Images can be replicated and dynamically migrated to other, interoperable offerings and platforms to create a virtual layered environment.
- Problems arise due to platform engines being proprietary or not replicating fully, unless standards like OVF for VMs are used. Moreover, replicating an image with data needs bandwidth, which requires optimised solutions.
- Image and data management aims to 1) minimise replication and manage deletion, 2) use segmentation for services, 3) differentiate between user-

data and images/services to optimise resources and 4) include intelligent data management such as map-reduce techniques. Horizontal scaling often requires the full dataset to be replicated. Vertical scaling can be based on data segmentation and distribution.

This indicates that automation is of critical importance to IaaS, as the cloud elasticity need is the driver for these techniques. Elasticity requires an *automated management of integration tasks*, specifically dynamic integration and workload management, cf. Table 2.

Specific concerns for the PaaS layer (SaaS is subsumed here as we are considering PaaS-provisioned application services) apart from elasticity are:

- Platforms need to support composition, orchestration and service mashups to aggregate and customise service offerings [13], [4], [6].
- For most applications, base image duplication suffices. However,with many users per application, full replication with customer-specific data and code is generally required. Where base images (e.g., for .NET) are available, we only need to replicate service instances, and not a full image.
- Further problems arise for composition as QoS is generally not compositional, e.g., the security of a composition is determined by its weakest link.

In contrast to IaaS, *automated management of aggregation, customisation and integration* is more of a concern for PaaS. Note that the financial aspects for arbitrage apply to IaaS, PaaS and SaaS equally. Intermediation and arbitrage refer to activities relevant for all layers.

A common concern for all layers is standardisation. *Standardisation* in terms of OVF as an image format, or OCCI as an interface for infrastructure-level resource management, are solutions. Interoperability, which is reflected in the first feature category in Table 2, can be achieved through standardisation based on open and published standards, or on de-facto standards from widely used open-source or proprietary systems. However, standards often do not succeed. Some proposals in the Web services stack (WS-*) are examples. Problems encountered are 1) diversion of specifications, 2) the slow process of standardisation and 3) competing standardisation bodies – the latter is an obvious problem in the cloud domain, where organisations from different areas of IT and computing are active (SNIA, DMTF, OMG, W3C, OGF etc). While some mature standards exist for the services domain in the context of Web Services (W3C, OASIS etc), cloud services are not necessarily WS-compliant. Some solutions exist. IaaS standards, such as OCCI and CIMI, cover service lifecycle management; TOSCA addresses portability and CDMI data management. Open-source IaaS supporting these include Openstack, which is a lifecycle management product in line with the CIMI and OCCI aims, and the mOSAIC API that supports composition and mashups at an infrastructure level [38]. Compliant PaaS systems include Cloudify, a management tool for vertical cloud stack integration, and Compatible One, a broker for horizontal integration [9], [11].

### 2.5 Brokerage - User Perspectives

Brokerage is a mediation process between different parties. It is important to consider different user perspectives, i.e., the consumer and service provider as well as the broker itself. Both consumers, providers and also prosumers (that consume and provide at the same time) have their own lifecycles. They differ in terms of their responsibility for different features and also the relevance of these in the context of their activities. Gartner, for instance, has organised its brokerage model along different provider and prosumer types.

- App Developers and Suppliers: The app developer's work should be supported by abstracting the lower-level work through architecture and programming features, only exposing the business-rule coding interface. This is seen by the progression from API libraries and frameworks, to devops and now PaaS platforms for multi-language, multi-framework development and deployment, to collaborative app composition, and to app lifecycle management. The developer does not need to know about the elasticity of the IaaS, for example. These will become prosumers that construct the broker application on top of a broker platform.
- End Users: Consumers of the app/cloud service just need the interface and use the broker application service. They are arguably the greatest beneficiaries of cloud brokerage.
- Platform Providers: The other main player influenced by these technologies is the new player, the Broker Platform Provider. Their primary task is the platform development addressing both end-user and developer needs. Although, it is often the case that the providers not only develop, but also host and provision the broker platform.

The different user perspectives need to be considered when weighing and interpreting comparison results. Developer features, for instance, are less relevant for end users, but critical for application developers.

## 3 A Classification and Comparison Framework for Cloud Service Broker Architectures

In this section, we introduce a dedicated 3-pronged framework for broker classification and comparison, which we will introduce first (see Fig. 1).

- Broker application dimensions: categorisation in terms of cloud delivery model, broker construction, broker scope

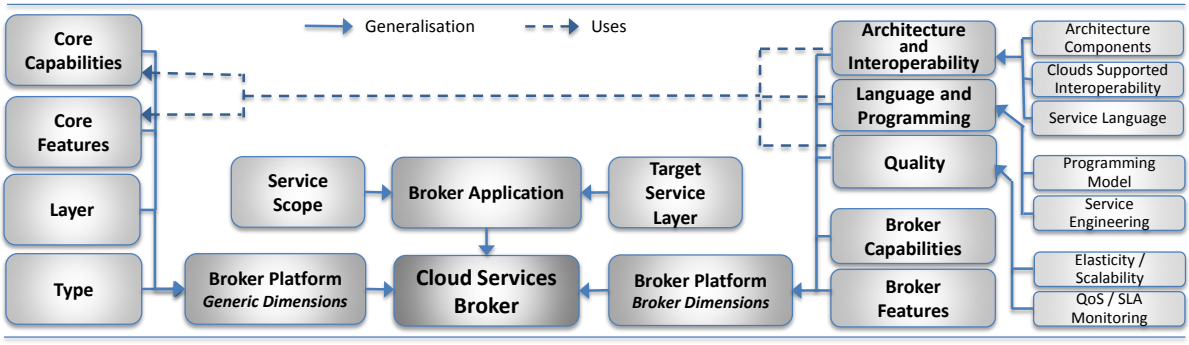



Fig. 1. Cloud Broker Architecture – Basic Ontology.

- Generic platform dimensions: a categorisation schema for a basic cloud platform classification
- Broker platform dimensions: broker-specific categories plus a detailed, descriptive classification

This forms a basic formal ontology, i.e., a taxonomy with defined concepts and instances, that allows the description of broker applications and platforms in terms of three dimensions, each defined through concepts and either predefined instances or textual descriptions. The ontology use a mix of categorical and descriptive elements. The former also combines single and multiple-valued categories. Fig. 2 describes the structure of a description from platform to application using a hierarchy of generic and broker-specific capabilities and features. The classification is a software architecture specification, involving architecture definition and quality assurance concerns – using the features and capabilities of the platform.

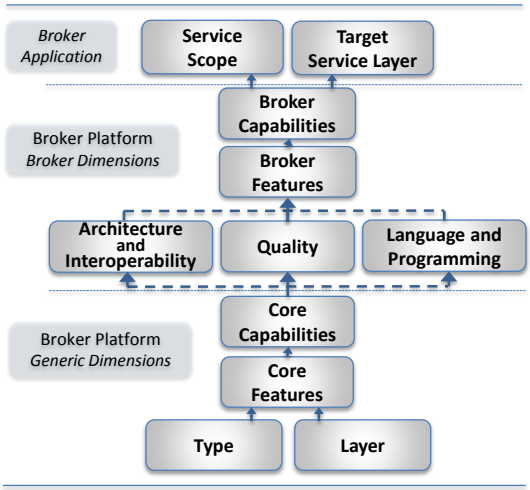



Fig. 2. Cloud Broker Architecture – Structure.

Section 3 is the core contribution section with the classification framework. Section 4 describes its application both to illustrate its utilisation and to validate the quality of the framework. Section 2 plays an equally important role. It analyses the literature on brokerage and extracts key concerns (through Tables 1, 2 and 3). This is then converted into a 3-pronged framework covering application and platform concerns as identified: broker application dimensions, generic platform dimensions and broker platform dimensions. Specifically the platform dimension requires more description mechanisms, covered in Tables 4 and 5 on generic and broker-specific platform concerns. The categories emerge from the analysis. The origin of the concepts/vocabularies are documents as notes to the Tables (e.g., 2, 4 and 5). Core artefacts to provide overall conceptual and architectural views are an ontology capturing key concepts (Fig. 1) and a reference broker architecture (Fig. 2).

### 3.1 Broker Application Dimension - Service Layer and Scope

A cloud service broker application is a cloud-based software system that builds on top of a broker platform (often cloud-based) and provides an application service to providers and end-users as consumers. Based on our requirements analysis from Section 2, two application dimensions can be identified:

- service scope defined through patterns that identify the extent of support: Management (operations), Broker (mediation), Marketplace (open collaboration and competition) (Section 2.3).
- target application services: referring to the cloud delivery models IaaS, PaaS and SaaS (Section 2.4).

The platform dimensions will be addressed now in Sections 3.2 and 3.3.

### 3.2 Generic Platform Dimension - Categorisation-based Taxonomy

We categorise brokerage solutions by the deployment model, but also specific features or functions that each of them provides. We have defined a comparison framework to categorise cloud solutions along these concerns in Table 4. We chose these concerns to,

- firstly, broadly categorise the system in terms of its main function (the system type that indicates its target layer and central function in that layer) and whether it is proprietary or open-source,

TABLE 4
Categorisation Taxonomy – Generic and Broker Platform Properties.

| Category | Concepts | Source |
|---|---|---|
| *Platform - Generic:* | | |
| *System Type* | Multi Cloud API Library, IaaS Fabric Controller, Open PaaS Solution, Open PaaS Provider | typical cloud product description categories[1] |
| *Cloud Stack Layer* | IaaS, PaaS, SaaS | widely accepted cloud delivery models |
| *Distribution Model*[2] | Open Source (for all solutions considered) | typical software product description categories |
| *Core Capabilities* | Multi-IaaS Support, Multi Language / Multi Framework Support, Multi Stack Support | from categorisation of generic cloud terms[1] |
| *Core Features / Components*[2] | Service Description Language, Native Data Store, Native Message Queue, Programming Model, Elasticity & Scalability, QoS/SLA Monitoring | from categorisation of generic cloud terms[1] |
| *Platform - Broker-specific:* | | |
| *Broker Capabilities* | Aggregation, Customisation, Intermediation, Integration, Arbitrage – broad support for broker and marketplace applications | from capability discussion in Section 2.2 (Table 1) |
| *Broker Features* | Service Management, Discovery/Composition, SLA and User Management, Monitoring, Metering, etc. – direct support for broker application features | from feature discussion in Section 2.2 (Table 2) |

*Note 1*: categorisation terms (instances) extracted from these sources: [10], [12], [20], [27], [43].
*Note 2*: Other general cloud platform properties could have been considered. These include auxiliary information for the distribution model, e.g., license model, or properties that describe interoperability aspects, e.g., cloud tools or programming languages supported. We have left out features that are not strictly oriented towards broker construction or that would only add auxiliary descriptive information for core categories (such as languages support if support exists).

- secondly, a range of standard properties and individual components are singled out. Properties chosen here (Core Capabilities) refer to the necessary capabilities for brokers to integrate offerings.

The two features categories organise a number of system components into common and more advanced ones. For the brokerage types introduced earlier, we distinguish support dimensions for service descriptions and the associated manipulation functions: vertical support for the brokerage types customisation and aggregation through e.g., discovery and selection features for services; horizontal support for the brokerage types integration and aggregation through e.g., queuing, storage or elasticity management features (in federated environments) as applications.

There are multiple dimensions for categorizing broker systems. An ontological system allows a product to be tagged with multiple attributes, values, and relationships. The taxonomy is a simple ontology. The aim was to create the right structure of categories and attributes for the systems, assigning multiple categories, attributes, and attribute values.

In Figs. 6, 7 and 8, we will categorise a number of solutions [16], [8], [9], [11], [22], [38], [29], [30], [31].

### 3.3 Broker Platform Dimension - Categorisation and Descriptive Facet-based Description

The broker-specific platform characteristics shall be described in two ways:

- categorisation-oriented: addressing the main objective of the service broker construction (service prosumption - how services/functions are combined and passed on): Integration, Aggregation, Customization, Arbitration, Intermediation. This has been described in Section 2.2.
- descriptive: addressing platform-based broker construction as a software systems development based on three facets of software engineering: architecture & interoperability, languages & programming, and quality.

The second format allows us to drill down and compare the support of broker platforms using a more descriptive format. The aim is to judge the support given by the platform to facilitate broker capabilities such as aggregation, intermediation or arbitration. This second, deeper and more descriptive classification schema is based on three facets derived from generic software engineering concerns – see Table 5.

## 4 Application and Evaluation

We applied the categorisation and comparison framework to a number of cloud brokerage solutions. The primary aim of this application was to evaluate the framework regarding two concerns:

- Adequacy (fit for purpose): can it identify strengths/limitations (compared to self-proclamation) by detecting the presence or absence of common features using a unified terminology?
- Completeness: are major features of tools not covered by the framework?

Compliance with specific definitions such as NIST is not an objective. Neither is the comparison itself meant to be comprehensive in terms of the solutions covered, although we apply a rigorous approach to the selection of the solutions.

### 4.1 Selection of Cloud Solutions

We selected only open-source solutions (to determine the state-of-the-art) and research-oriented projects (to

TABLE 5
Descriptive Comparison – Broker Platform Properties.

| Facet[1] | Sub-Facets | Definition | Source[2] |
|---|---|---|---|
| *Architecture and Interoperability* | - *Architecture Components*<br>- *Clouds Supported / Interoperability* | The solution architecture is a key element in the definition of a broker. Of practical relevance are the existing, typically lower-layer solutions that the system supports. This is an interoperability concern. | Software Design |
| *Languages and Programming* | - *Service Language*<br>- *Programming Model*<br>- *Service Engineering* | Service description and implementation plays a key role for interoperability [33]. For selected solutions, we look at the following three aspects:<br>• service language – the core notation, including the coverage of concerns vertically (PaaS/IaaS integration) and horizontally (full lifecycle management) and how this is manipulated (format and API).<br>• programming model – using the language to program brokerage solutions, linking to SOA principles and other development paradigms.<br>• service engineering – covering wider design and architecture concerns, including monitoring and mashups. | Software Construction |
| *Service and Architecture Quality* | - *Elasticity / Scalability*<br>- *QoS / SLA Monitoring* | A wider range of non-functional software qualities (e.g., based on ISO9126) would need to be considered here. These quality concerns also need to be addressed by the service description notation. Scalability and elasticity are two specific cloud quality concerns generally used to differentiate capabilities of the cloud in comparison to on-premise architectures. Therefore, we mainly focus on these. Load balancers are typically used to control elasticity based on monitored key performance indicators (KPIs). Multi-tenancy, if available, can alleviate elasticity problems. Wider quality/KPI concerns are equally important. Based on specifications, these are looked after by configuration management tools to set up probes and monitoring tools to collect and analyse data. | Software Quality |

*Note 1*: The descriptive elements and their sub-facets should be based on the Vocabulary introduced in the *Generic Capabilities* and *Generic Features* in Tables 1 and 2.
*Note 2*: Based on the IEEE Software Engineering Body of Knowledge SWEBOK [42] and the Joint ACM/IEEE Curriculum Guidelines for Graduate Degree Programs in Software Engineering [1].

detect maturing trends). Commercial solutions have been considered in the analysis to determine the framework in Section 2.2, but have been excluded here to reduce threats arising from more biased and incomplete available information. Sample open-source solutions (OSS) can thus be categorised:

- IaaS: OpenStack, for instance, is a basic IaaS cloud manager that transforms data-centres to become IaaS clouds [31].
- PaaS: OpenShift and CloudFoundry are open PaaS platforms assisting the cloud app developer by commoditising the software stack [8], [30].

The Open IaaS/PaaS solutions can be differentiated from respective IaaS/PaaS brokers. In the following, we will try to point out the salient differences between some cloud brokers that go beyond IaaS/PaaS management solutions. For instance, Optimis and CompatibleOne are IaaS-oriented, and only 4CaaSt targets PaaS and to some extent also the SaaS domain. There is, however, SaaS broker activity in the commercial space – as reviewed in Section 2.2.

Solutions were selected based on key-word searches using the Google search engine following standard selection mechanisms used in systematic literature reviews (SLRs) [24], [21], applied to cloud solutions. A SLR reduces bias and follows a sequence of methodological steps. SLRs rely on well-defined and evaluated review protocols to extract, analyse and document results. We adopted a three-step review process that includes planning (based on Section 3), conducting (Section 4.1) and documenting (Section 4.2, 4.3). We use an explicit characterization framework of the reviewed items. This is the foundation for a comparative analysis based on our analysis dimensions. `Cloud` and `open-source` were search terms included, but `IaaS` and `PaaS` were distinguished:

- Terms `"cloud paas open-source"` resulted in
  `OpenShift, CloudFoundry, Cloudify, Stratos, Pivotal, Tsuru, Paasmaker, CloudBees, Cocaine`
  – from which we selected the first three.
- Terms `"cloud iaas open-source"` resulted in
  `Openstack, OpenNebula, jcloud, mOSAIC`
  – from which we selected the first three again.

Searching for `"cloud paas open-source service broker management"` aimed at covering dedicated brokerage solutions, but resulted only in `CompatibleOne` as a new, not yet covered solution.

To select relevant research projects, we matched

`"cloud project service broker management"`

against project summary information provided by the EU (full documentation available), which resulted in

`4CaaSt*, Broker@Cloud, Cloud4SOA, Modaclouds, mOSAIC*, PaaSage, Ocean, Optimis*, Reservoir*,`

from which we selected those that were finished at the time of this research (September 2014) – marked with an asterisk *. We focussed on the EU context as the reporting and documentation mechanism is

consistent for the selected projects. In [18], a more global review of research projects is provided – which has overlapping EU projects and similar results.

An observation is that the broker pattern receives attention and that reusable solutions are in development, starting with the IaaS layer, but including PaaS and SaaS over time. The existence of marketplaces, which are interesting for the diverse SaaS space, indicates the existence of broker solutions. A wide range of commoditised broker platforms can be expected in the future to service the different broker types defined, but also provide a fuller range of features.

### 4.2 Categorisation - Platform Dimension

In Tables 6 to 8, we categorise the solutions selected in Section 4.1 [8], [9], [22], [29], [38], [30], [31], [16], [11] according to the defined classification taxonomy. As we review broker platforms, only the two platform dimensions are covered – although it needs to be noted that some like CompatibleOne or 4CaaSt have application features as well. Compatible One provides, for instance, platform features like SLA management (SLAM) and intermediation and integration features (PROCCI), and also application functions for service brokering (BROKER). Note that in Table 8, instead of a detailed feature list, we only summarise this as 'Y' if a sufficient number of features is covered per category[2].

### 4.3 Descriptive Facet-based Comparison

In the following, we review solutions with respect to three facets defined in Table 5: architecture & interoperability, languages & programming, and quality. We do not consider all selected solutions in the comparison, but only select the most advanced ones for each aspect as the objective here is framework evaluation only. For this, a smaller number of feature-rich, advanced cases are sufficient. A comprehensive state-of-the-art review is not aimed at here. A detailed presentation of the comparison results is presented in Tables 9 to 11. Here, we single out key observations.

**Architecture and Interoperability.** In Table 9, a number of PaaS-level solutions are summarised in terms of these two aspects. Common are the utilisation of configuration management solutions, such as Chef or Git. The deployment is managed through consoles or APIs, mapping PaaS-level requests down to IaaS operations. As many IaaS solutions are often supported, interoperability is a critical concern. CompatibleOne is OCCI-compatible in its support for VM management, i.e., it provides dynamic architecture management and interoperability. For instance, Mosaic assumes a Linux OS, which runs Mosaic App Components (called CloudLets). A number of common commercial cloud solutions are supported by Mosaic, including Amazon and Rackspace products.

**Languages and Programming.** Solutions are compared in Table 10, e.g., Cloudify uses application recipes and resource node templates (Groovy scripts) as the programming model. A service recipe contains LCM scripts, monitoring probes and IaaS resourcesrequirements. Mosaic uses an ontology as the notation and a component-based application programming model for portability of apps across compliant clouds. Patterns emerge as solutions to compose, connect and manage clouds in distributed contexts.

**Quality.** Table 11 covers quality. Common components such as monitoring, load balancers and scalability engines are the key capabilities required here.

### 4.4 Evaluation of the Framework

We have categorised broker platforms based on the platform taxonomy aspects. Specifically, looking at the perspective of the user types is now valuable. For instance, developers are supported in three categories: a) API library: jcloud, b) Devops: Cloudify and c) Full PaaS: CloudFoundry, OpenShift. A trend goes from provider-oriented solutions to developer-oriented solutions to end user-oriented cloud management [3] – 4Caast being an example of the latter. A deeper discussion of trends and challenges emerging from this discussion shall follow in the next section. We now turn to the evaluation of the framework itself in terms of adequacy and completeness.

**Adequacy.** The framework is adequate by providing a sufficiently complete framework.

- Broker Definition. This was achieved by using the three most cited brokerage definitions as the starting point. We define five types that subsume each of the three literature sources.
- Capabilities and Features. We identified fine-granular capabilities and features from the literature. Additionally, we used common categorisations from the cloud computing community (e.g., delivery models) and software engineering (specification and design) to add further dimensions.

While only 10 solutions have been used here for validation, initially 13 open-source and 6 commercial products were considered and documented in [15].

**Completeness.** The completeness has been empirically validated through an application for a state-of-the-art comparison. For this, the cases have been systematically selected to reflect a broad range of advanced systems. An investigation of the concrete descriptions has not resulted in any relevant concepts lacking in the framework. Some table parts (Tables 7 and 8) are sparsely populated, but still several tools per feature can be identified, which means that the respective feature is present in state-of-the-art systems and, thus, the respective feature category is valid.

2. A figure of 35% of features covered was considered as sufficient. The figure was derived from the percentage of core features that are on average covered by the selected tools.

TABLE 6
Broker Platform - Generic Category and Type.

| | LAYER and MAIN TYPES | | | | | |
|---|---|---|---|---|---|---|
| Name | Type | Cloud Layer | Multi Cloud API Library | IaaS Fabric Controller | Open PaaS Solution | Open PaaS Provider |
| OpenNebula | Cloud Fabric Controller | IaaS | | Y | | |
| OpenStack | Cloud Fabric Controller | IaaS | | Y | | |
| jclouds | API Library | PaaS | Y | | | |
| Cloudify | Cloud Devops & LCM | PaaS | Y | | Y | |
| CloudFoundry | PaaS | PaaS | | | Y | Y |
| OpenShift | PaaS | PaaS | | | Y | |
| CompatibleOne | IaaS Broker | PaaS | | | Y | |
| Mosaic | PaaS | PaaS | Y | | Y | |
| 4Caast | Service Broker | PaaS | | | | |
| Optimis | IaaS Broker | PaaS | | | Y | |

TABLE 7
Broker Platform - Generic Capabilities and Features/Components.

| | CORE CAPABILITES | | | CORE FEATURES | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | Multi IaaS Support | Multi Language / Multi Framework | Multi Stack | Service Description Language | Native Data Store | Native Message Queue | Programming Model | Elasticity Scalability | QoS / SLA Monitoring |
| OpenNebula | | | | | | | | | |
| OpenStack | | | | | Y | Y | Y | | Y |
| jclouds | Y | | | | | | | | |
| Cloudify | Y | Y | Y | Y | | | Y | Y | Y |
| CloudFoundry | Y | Y | Y | | Y | | Y | | Y |
| OpenShift | | Y | Y | Y | | | | Y | Y |
| CompatibleOne | Y | | | Y | | | Y | Y | Y |
| Mosaic | Y | | | Y | Y | Y | Y | Y | |
| 4Caast | | | | Y | Y | Y | | | Y |
| Optimis | | | | Y | Y | | Y | Y | Y |

TABLE 8
Broker Platform - Broker-specific Capabilities and Features.

| | BROKER CAPABILITES | | | | | BROKER FEATURES | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | Aggregation | Intermediation | Integration | Arbitrage | Customisation | Aggreg/ Integr/ Custom | Intermed/ Mgnt/ Arbitrage | Dynamic Integration | Workload Management |
| OpenNebula | | | | | | | | | |
| OpenStack | | | | | | | | | |
| jclouds | | | | | | | | | |
| Cloudify | Y | | Y | | | | | Y | |
| CloudFoundry | Y | | Y | | | | | Y | |
| OpenShift | | | | | | | | | |
| CompatibleOne | Y | Y | Y | Y | | | Y | Y | Y |
| Mosaic | Y | | | | | Y | Y | | |
| 4Caast | Y | Y | | Y | Y | Y | | | Y |
| Optimis | Y | Y | | | | | Y | | Y |

As a result, the solution is adequately fit for purpose. It allows a neutral classification along dimensions originating from different fields. It is therefore consistent with a common understanding of key concepts. It also serves as an analysis tool. From the results in Tables 9 to 11, e.g., the ongoing development of scripting languages emerges as the focus in the programming context, or work on scalability engines as a quality concern, emerges as indicators of activities.

### 4.5 Discussion and Challenges

The need for management support and interoperability becomes apparent in the context of cloud service brokerage, where independent actors in the ecosystem integrate, aggregate/compose and customise/adapt existing services [17], [27]. End-to-end personalisation becomes achievable. Prosumers may create mashups from existing services.

From the above comparison between various cloud solutions, we can note a difference between the needs of cloud brokerage and cloud marketplaces. We did already introduce them as different patterns above. Service management is already a common solution, often provided as a domain-independent platform.

- A *brokerage* solution needs to automate the process of matching service requirements with resource capacity and capabilities [40]. The ideal would be a total commoditisation of IaaS so that any compute resource could be plugged into a user's compute capacity. Therefore, interoperability is of importance. There are areas of compatibility that should be considered in match-

ing that are not handled by brokers currently. For example, none of the solutions that were assessed considered data integrity as a matching criterion; however, they all included performance in their criteria. Security is another aspect that has a technical nature, but is also abstract insofar as it can be implemented by a cloud provider. Data integrity and security policy enforcement, if considered as criteria when evaluating cloud interoperability, may need to be formalised using a standardised language to describe common aspects, similar to the languages that have been created to model other cloud entities.

- A *marketplace* needs to additionally focus on the possibly distributed architecture of the applications as well as the cloud. The appstore model appears to be the de-facto model of choice for the marketplace, but this seems more an admission of the success of the Apple initiative rather than research. There is a potential to explore other forms of the online marketplace suitable to cloud apps and their composition [26], [13]. This can also be extended to an even more commodity-based scenario where all services could be registered on a wide-area multi-marketplace scale.

Table 7 shows that broker-specific core and advanced features are not comprehensively supported in available tools. 4Caast and Optimis cover these, but are as research projects not available as products. A wider discussion, arising from the concrete comparisons above, shall now follow in terms of *commoditisation*

The *commoditisation* of cloud services is an emerging need arising from the discussion – specifically from the language and programming facet. A trend is to move from the lower IaaS layer to PaaS and onwards to encompass SaaS, aiming to integrate lower layers – 4CaaSt is an example. To make this work, services at all layers need to be available for a uniform way of processing in terms of selection, adaptation, integration and aggregation. Commoditisation is the concept to capture this need. Some concrete observations related to the reviewed open-source solutions are: (i) fully functional image and vertical stack building capabilities (CompatibleOne leadership), (ii) programming and operational support of service composition (4CaaSt leadership) and (iii) graphical manipulation of service abstractions (Optimis leadership). Commoditisation can be supported through a uniform representation based on description templates. The programming model with its supporting programming/scripting languages is crucial here. These need to cover the architecture stack and meet language and quality concerns discussed in Section 4.

Commoditisation is an enabler of functions on top of a broker platform. Thus, additional challenges and requirements, for marketplaces in particular, are:

- data integration and security enforcement as non-functional requirements,
- social network functions allowing service ratings by the communities,
- SLA management to be integrated, e.g., in terms of monitoring results.

Commoditisation can be facilitated by an operational development and deployment model to act as an enabler. This ties in with another observation. A proliferation of cloud capacity clearing-houses, which operate similar to a spot market to allow clouds to buy and sell spare cloud capacity on a very short-term basis, has only started. It is less clear what is needed to facilitate this from a construction perspective.

*Federation* is the second trend for brokerage solutions [7], i.e., to work across independently managed and provided cloud solutions of an often heterogeneous nature. Some challenges and requirements arising from the architecture and interoperability discussion can be identified:

- Reference architectures – cf. NIST cloud brokerage reference architecture [27].
- Scope of control – the management of configuration and deployment based on integrated and/or standards-based techniques [25].
- Federation and syndication – as two forms of distributed cloud architectures [37].

## 5 Conclusions

We have introduced the main concepts of service brokerage for clouds, using some concrete systems and platforms to identify current trends and challenges and compare current, primarily open-source solutions. Brokerage relies on interoperability, quality-of-service and other architectural principles. Brokers and marketplaces can play a central role for new adopters migrating to the cloud or between providers [21], [34]. Brokers will act as first points of call.

Our classification and comparison framework identifies three dimensions – two platform and one application oriented. Taxonomy-based categorisation helps to characterise solutions in terms of type, common components and features. The second mechanism is a more descriptive, layered taxonomy starting with architecture and interoperability, languages and programming, and quality as facets. Our objectives have been to clarify the conceptualisation of cloud service brokerage by taking more architectural concerns into account, but also to provide a framework for industry and academia to classify and compare solutions.

An observation of our comparison based on the framework is the emergence of cloud broker solutions on top of cloud management. A further separation of marketplaces, often in the form of appstores, is necessary. A number of activities work in this direction. Compatible One is an example showing how OCCI is used as an infrastructure foundation and built upon to provider PaaS-level brokerage. 4CaaSt

in a similar vein aims to integrate the layers and move toward a marketplace solution. Commercial broker applications, that have already supported the framework construction here in Section 2.2, show already existing brokerage and marketplace solutions ranging from images to software services, essentially commoditising the respective cloud resources.

Service description mechanisms discussed in [28], [36], [32] (as manifests, recipes and blueprints), but also in standards and languages like TOSCA and CloudML, can serve to abstract, manipulate and compose cloud service offerings in an effort to commoditise the cloud. These description mechanisms, based on an abstract model serve two purposes: Firstly, to abstractly capture, present and manipulate cloud resources. Secondly, to serve as a starting point to link to configuration and other deployment concerns in federated clouds. Thus, commoditisation and federation emerge as challenges from our discussion.

Future work includes trust as an equally important concern that is more difficult to facilitate technically than commoditisation. A mechanism is needed for not only vetting individual providers, but also to allow this to happen in layered, federated and brokered cloud solutions. Furthermore, end-users, intermediaries such as brokers and providers – the key roles we identified – have different lifecycles that need to be integrated. Forrester Research [14] has already provided basic ideas, but a deeper investigation into lifecycle and workflow integration would be necessary.

## ACKNOWLEDGMENTS

This research has been supported by the Irish Centre for Cloud Computing and Commerce and the Royal Irish Academy/Royal Society Intl Cost Share Grant IE131105.

## REFERENCES

[1] ACM/IEEE. Area Definitions – Joint ACM/IEEE Curriculum Guidelines for Graduate Degree Programs in Software Engineering GSwE2009. http://www.gswe2009.org. 2009.
[2] 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. Pages 129-136. Palo Alto, US. ACM Press. 2006.
[3] T. Benson, A. Akella, S. Sahu, A. Shaikh. Peeking into the Cloud: Toward User-Driven Cloud Management. CloudS 2010 Conference, Sydney, Australia. 2010.
[4] D. Benslimane, S. Dustdar, A. Sheth. Services Mashups: The New Generation of Web Applications. Internet Computing, vol.12, no.5, pp.13-15, 2008.
[5] D. Bernstein, E. Ludvigson, K.Sankar, S. Diamond, M. Morrow Blueprint for the Inter-cloud: Protocols and Formats for Cloud Computing Interoperability. Intl Conf Internet and Web Appl and Services. 2009.
[6] R. Buyya. Compatibility-Aware Cloud Service Composition under Fuzzy Preferences of Users. IEEE Transactions on Cloud Computing, 2(1):1-13. 2014
[7] R. Buyya, R. Ranjan, R.N. Calheiros. Intercloud: Utility-Oriented Federation of Cloud Computing Environments For Scaling of Application Services. Intl Conf on Algorithms and Architectures for Parallel Processing , LNCS 6081. 2010.
[8] Cloud Foundry. Open Source PaaS Cloud Provider Interface. http://www.cloudfoundry.org/. 2015.
[9] Cloudify. Cloudify Open PaaS Stack. http://www.cloudifysource.org/. 2015.
[10] Cloud Standards. http://cloud-standards.org/. 2015.
[11] CompatibleOne. Open Source Cloud Broker. http://www.compatibleone.org/. 2015.
[12] ETSI Cloud Standards. http://www.etsi.org/news-events/news/734-2013-12-press-release-report-on-cloud-computing-standards. 2015.
[13] C. Fehling, R. Mietzner. Composite as a Service: Cloud Application Structures, Provisioning, and Management. Information Technology 53:4, pp. 188-194. 2011.
[14] Forrester Research. Cloud Brokers Will Reshape The Cloud. 2012. http://www.cordys.com/ufc/file2/cordyscms_sites/download/09b57cd3eb6474f1fda 1cfd62ddf094d/pu/
[15] F. Fowley, C. Pahl, L. Zhang. A Comparison Framework and Review of Service Brokerage Solutions for Cloud Architectures. 1st International Workshop on Cloud Service Brokerage (CSB 2013). Springer. 2013
[16] S. Garcia-Gomez et al. Challenges for the comprehensive management of Cloud Services in a PaaS framework. Scalable Computing: Practice and Experience 13(3). 2012.
[17] Gartner - Cloud Services Brokerage. Gartner Research, 2013. http://www.gartner.com/it-glossary/cloud-services-brokerage-csb
[18] N. Grozev, R. Buyya. InterCloud architectures and application brokering: taxonomy and survey. Software: Practice and Experience. 2012.
[19] C.N. Höfer, G. Karagiannis. Cloud computing services: taxonomy and comparison. Journal of Internet Services and Applications, 2(2), 81-94. 2011.
[20] IEEE Cloud Standards. http://cloudcomputing.ieee.org/standards. 2015.
[21] P. Jamshidi, A. Ahmad, C. Pahl. Cloud Migration Research: A Systematic Review. IEEE Transactions Cloud Computing. 2013.
[22] Jclouds. jclouds Java and Clojure Cloud API. http://www.jclouds.org/. 2015.
[23] A. Juan Ferrer et al. OPTIMIS: A holistic approach to cloud service provisioning. Future Generation Computer Systems, 28(1):66-77. 2012.
[24] B. Kitchenham and S. Charters. Guideline for Performing Systematic Literature Reviews in Software engineering. Keele University and University of Durham, 2007.
[25] A.V. Konstantinou, T. Eilam, M. Kalantar, A.A. Totok, W. Arnold, E. Sniblel. An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds. Intl Workshop on Virtualization Technologies in Distr Computing. 2009.
[26] R. Mietzner, F. Leymann, M. Papazoglou. Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns. Intl Conf on Internet and Web Applications and Services. 2008.
[27] NIST. Cloud Computing Reference Architecture. http://dl.acm.org/citation.cfm?id=2385915. 2012.
[28] D.K. Nguyen, F. Lelli, Y. Taher, M. Parkin, M.P. Papazoglou, W.-J. van den Heuvel. Blueprint Template Support for Cloud-Based Service Engineering. Proceedings ServiceWave11, Poznan, Poland, October 2011.
[29] OpenNebula. OpenNebula - Open Source Data Center Virtualization. http://opennebula.org/. 2015.
[30] OpenShift. Cloud computing platform. https://openshift.redhat.com/. 2015.
[31] OpenStack. OpenStack Open Source Cloud Computing Software. http://www.openstack.org/. 2015.
[32] C. Pahl. Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture. European Conf on Model-Driven Architecture ECMDA2005. 2005.
[33] C. Pahl, S. Giesecke and W. Hasselbring. Ontology-based Modelling of Architectural Styles. Information and Software Technology (IST). 51(12): 1739-1749. 2009.
[34] 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 2013. 2013.
[35] C. Pahl, H. Xiong, R. Walshe. A Comparison of On-premise to Cloud Migration Approaches. Europ Conf on Service-Oriented and Cloud Computing ESOCC. 2013.

TABLE 9
Architecture and Interoperability.

| | *Cloudify* | *CloudFoundry* | *OpenShift* | *CompatibleOne* | *4Caast* |
|---|---|---|---|---|---|
| *Architecture / Components* | - Console for platform commands. Web management console for monitoring. - Service Manager uses scripting (recipe) to cater for middleware stack - Cloud Controller is REST endpoint to manage app deployment & control; injects agent on VM to install & orchestrate app deploy / monitor/ scale - Cloud Driver: VM templates for different IaaS clouds in configuration. Triggers host provisioning | - Console pushes app to cloud; deployment management / configuration through console. - Controller runs as a VM on the target IaaS; controls all Cloudfoundry (CF) spawned cloud VMs. Does not manage any IaaS layer functions. - IaaS provider must support CF. Apps created using CF are deployed to CF VMs controlled by a Cloud Controller on CF-compliant IaaS clouds. | - Divided into control plane (Broker) and msg / hosting infrastructure (nodes). - Controller is command CLI shell, used to create apps. GIT for app management / deployment. - Gear is application container and a virtual server/node accessed via ssh. Cartridge service runs on a Gear. App LCM scripts allow for post-deployment action hooks to run on VMs. | - ACCORDS exposes features through REST API. - Parser validates Manifest against CORDS schema and maps elements to valid OCCI categories which are then instantiated. - Publisher provides which endpoint serves which categories. Parser runs and produces a plan of OCCI instances for resolution (instance can receive/send data). - Broker processes plan and invokes instances. | - Exec Container REC runs instances. - Deployment Manager maps deployment model (service template, QoS constraints) to OVF. Service Manager deploys images using Claudia. - REC includes an agent (application LCM, control) and a server (storage, config data). Deployment Server (Chef) talks to Service & REC Manager. OVF Manager creates extended OVFs from resolved BluePrints. |
| *Clouds Supported / Interoperability* | Supports Azure, OpenStack, CloudStack, EC2, Rackspace, Terramark (buildable for any of the jclouds above) | Supports AWS, Openstack, Rackspace, vCloud, vSphere. Hosted as public PaaS. | Uses DeltaCloud; app runs on RedHat certified public cloud (needs deltacloud support). | OCCI provider interfaces (PROCCIs) for OpenStack, OpenNebula and Azure (also SlapOS and SlapGrid). | FlexiScale driver provided. OpenNebula supported. Generic IaaS Cloud API through Tcloud. |


[36] M.P. Papazoglou, W.J. van den Heuvel. Blueprinting the Cloud. IEEE Internet Computing, November 2011.
[37] A. Paya, D.C. Marinescu. Clustering Algorithms for Scale-free Networks and Applications to Cloud Resource Management. 2013.
[38] D. Petcu et al. Portable cloud applications - from theory to practice. Fut. Gen. Computer Systems 29(6):1417-1430. 2013.
[39] S. Ried. Cloud Broker A New Business Model Paradigm. Forrester. 2011.
[40] L. Rodero-Merino, L.M. Vaquero, V. Gil, F. Galan, J. Fontan, R.S. Montero, I.M. Llorente. From Infrastructure Delivery to Service Management in Clouds. Future Generation Computer Systems, vol. 26, pp. 226-240. 2010.
[41] L. Sun, H. Dong, and J. Ashraf. Survey of Service Description Languages and Their Issues in Cloud Computing . Eighth International Conference on Semantics, Knowledge and Grids (SKG) 2012. pp. 128-135. IEEE. 2012.
[42] IEEE. Software Engineering Body of Knowledge SWEBOK. http://www.computer.org/portal/web/swebok. 2004.
[43] Wikipedia - Cloud_API,Wikipedia. 2015.




**Frank Fowley** is a Senior Research Engineer at the Irish Centre for Cloud Computing and Commerce IC4. His research interests include cloud computing and security. Frank holds an M.Sc. from Dublin City University in Forensics and Secure Computing.



**Claus Pahl** is a Senior Lecturer at Dublin City University, where he is a principal investigator of the Irish Centre for Cloud Computing and Commerce IC4. His research interests include software engineering in service and cloud computing. He holds a Ph.D. in computing from the University of Dortmund and an M.Sc from the University of Technology in Braunschweig.



**Pooyan Jamshidi** is a research associate at Imperial College Dublin. He holds a PhD from Dublin City University. He received the BS and MS degrees in computing from Amirkabir University of Technology. His general research interests are in software engineering and his focus lies predominantly in the areas of self-adaptive software, software architecture and cloud computing.

TABLE 10
Service Language, Programming Model and Service Engineering.

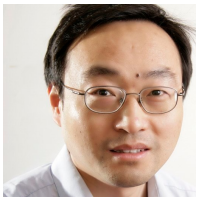
| | *Reservoir* | *CompatibleOne* | *4Caast* | *Optimis* |
|---|---|---|---|---|
| *Service Language* | Service Definition Manifest for metadata; software stack (OS, middleware, app, config, data) in virtual image; service descriptions for contracts between service provider SP and infrastructure provider IP. Manifests (OVF) relate abstract entities and LCM of services. Feedback between SP and IP allows IP to scale and monitor. | Units of Service Manifest: Image & Infrastructure. Image: System (base OS) & Package (stack config); Infrastructure: Storage, Compute & Network. Image is description of manual app build. Image has agent that is embedded in VM & runs on startup. Agent is script to run required configuration, set up monitoring probes, or download components. | 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 (is service orchestration feature). | Service Manifest includes sections per component per VM. Service Register has sections for SP requirements and IP capabilities, VM abstract description, TREC (trust, risk, eco-efficiency, cost), elasticity, data protection. Has provider description schema for a SP to provide its capabilities in an XML Optimis-compliant format. |
| *Programming Model* | Elasticity is defined using ECA rules to scale infrastructure dynamically based on application KPI metrics. Rules in OCL. | PaaS4Dev: Java EE services (EE5/6 web profile) & Enterprise OSGi services (http, jndi, transaction) for development | Uses Active MQ, postgresql, jonas, ow2orchestra, apache serv bus. Ontology-based BP schema using Jena, SPARQL. | Java schemas, jaxb, xmlbeans, REST, monitor; also jax-ws, cxf, javagat. IDE is Eclipse with plugin for Optimis core classes. |
| *Service Engineering* | Service provisioning described in Deployment Descriptor. Service configuration automation based on Xen configuration. Service Elasticity is achieved through mapping Manifest KPIs with run-time metrics gathered by app monitoring agents. | - Nested manifests support service composition. - COSACS module embeds in VM image mechanisms to manage lifecycle, e.g., post-creation monitoring setup and appliance config, in conjunction with image production module. | - Request Language BRL & request patterns create Blueprint BP service specification - mapped to operations and mgmt API calls. Mashup for composition. - BP consists of BP images, contains functional, KPI & policy parameters. | Toolkit provides image mgmt, context manager injects context information to VMs and Elasticity Engine to add/remove resources. Service Deployment Optimiser optimises placement of services. Configuration using the Toolkit IDE. |

TABLE 11
Quality: Scalability/Elasticity and SLAs.

| | *CloudFoundry* | *OpenShift* | *CompatibleOne* | *4Caast* | *Optimis* |
|---|---|---|---|---|---|
| *Elasticity / Scalability* | Can add/remove instances for scalability and change CPU & memory limits on VMs | Automatic gears add/remove as load changes. Multi-tenancy using multi-gears on VMs | Elasticity is provided by the load balancer module for the IaaS resources. | Not in current release. | The toolkit includes an Elasticity Engine to add / remove resources. |
| *QoS / SLA Monitoring* | There is only a basic logging facility with Cloud foundry but there are many third-party Cloud Foundry monitoring plug-ins can be used to provide application monitoring, such as Hyperic. | The application scaling, when automatic, is based on concurrent application request thresholds. The resources consumed by an application can be monitored and viewed from the Console. | via COMONS Monitoring module. | Monitoring based on probe injection on PICs via REST. Modified JASMINe provides dynamic probe deployment & config. Chef recipe configs VM probes for REC manager. Monitoring based on collectd stats for forecasting. | Framework uses REST to get CPU/disk usage from monitoring which resides on nodes and run as scripts to feed data to monitor store. SLA Manager built using WSAG4J based on OGF WS-Agreement. |



**Daren Fang** Daren Fang is a Research Associate at Edinburgh Napier University, UK. His research interests include cloud service modelling, green service optimization, service adaptation, and service evolution.



**Xiaodong Liu** Xiaodong Liu is a Reader and the Director of Centre for Information & Software Systems at Edinburgh Napier University. His research interests include context-aware adaptive services, cloud service evolution, pervasive computing. He holds a PhD in software engineering from De Montfort University, a MSc and BEng from Renmin University and Xi'an Jiaotong University.