Service Component Architecture for Building Cloud Services

by Dr. Muthu Ramachandran, Principal Lecturer in the Computing and Creative Technologies School

Abstract: The emergence of cloud computing has resulted in an increase in data centers, primarily due to rising global demand. The newly emerging cloud paradigm is heavily based on the Software-as-a-Service (SaaS) concept that provides services on-demand, utilising resources more effectively within cloud environments. Service component architecture (SCA) has been developed based on software components to build cloud services and architecture. This article discusses SaaS characteristics and how they can be designed with service components. In this research, we have developed a number of component models that have been designed for supporting cloud design characteristics and their associated architectural layers. This work has also developed a series of best practice design guidelines for service component design that support the componentization of cloud applications.

Introduction

Cloud computing has evolved to address the availability of computing resources which can be accessed from anywhere and anytime. In particular, computing hardware and software often gets outdated and, hence, it is wise to outsource computing resources and to manage their IT infrastructures outside of their company premises, which is more cost-effective than is the case at present. Applications can be leased (like pay-as-you-go service) rather than being purchased, and companies have increased their data centres due to demand (Amazon, Microsoft, and IBM). Cloud computing is heavily based on ‘software as a service’ concept and needs high speed Web access. It provides services on demand utilising resources more effectively within the cloud environment. Cloud architecture, its layers, and its composition of components and services need to be designed for scalability and re-configurability, as they support services and their SLAs (service level agreements). The resource management of cloud computing is the key to achieving its potential benefits. Therefore, it is essential to design cloud applications as Web-based service components further based on well-proven CBSE (component based software engineering) methods and techniques (with appropriate security controls).

Cloud computing is dependent on Internet technology therefore we need to design cloud services which are designed for security. However, this new trend needs to be more systematic with respect to software engineering and its related process that can be customised for cloud service development. Wand and Laszewski [REF-1] defines cloud computing as a set of network-enabled services which provide scalable, guaranteed QoS (Quality of Service), inexpensive computing platforms on demand, is customisable (personalised), and all of which can be accessed in a simple and pervasive way. An overview of the different cloud computing paradigms are discussed and presented with definitions, business models, and technologies by Wand and Laszewski [REF-1] and by many others [REF 1-2]. One of the main reasons for designing services as components is that the expected cloud characteristics such as flexibility, QoS, elasticity can all be achieved by designing service interfaces.

Software components provide a good design rationale supporting various requirements of application development, design flexibility, system composition, testability, reusability, and other design characteristics. Component based designs are customisable and interfaces can be designed supporting SLA (service level agreement). SLAs vary between service providers which need to be customised without much effort. This can only be achieved using a component which has been designed for flexible interface that links to a number of SLAs. Each SLA and business rule can be represented as a set of interfaces that can be mapped onto a knowledge-based database or a data server. This also allows reuse of SLAs for any individual service.
providers. Erl [REF-3] suggests the building blocks of a governance system are precepts, people, processes, and metrics. Some of the important characteristics of cloud computing mentioned are:

- On demand services
- Handling wide area network addresses
- Resource grouping
- Efficient elasticity
- Measurable service delivery

On demand services need to be managed efficiently as part of the cloud infrastructure management system. Most of the cloud users’ computers and devices are behind firewalls which have been assigned special IP addresses that can’t be directly routed over the Internet. Handling such wide area network addresses need to be managed securely in order to protect cloud content and maintain privacy. Resource grouping is a mechanism by which cloud resources are shared to provide efficient services. Elasticity (meaning a resource can be increased and decreased automatically depending on the need) is one of the key promises of cloud computing benefitting business services. Elasticity can be efficiently managed by integrating compute and storage resources into a common framework to eliminate associated cost of managing cloud resources to scale applications. Service delivery is a key concept in making cloud technology a cost effective business solution for IT systems. Therefore, cloud providers need to cost-effectively scale services, resources, maintain profitability, to meet cloud customer requirements for cost, performance, data-integrity, and end-user experience.

Our earlier work described by Ramachandran [REF-4 & REF-5] on component models for Web services and service-oriented architecture (SOA), grid computing, and various other systems can become an integrated aspect of any cloud computing architecture and application design. We also need to understand the basic differences amongst SOA (service-oriented architecture), grid and cloud computing. SOA is to offer services which are based on open standard Internet services, virtualisation technology, and have been running in a different environments. Grid offers services from multiple environments and virtualisation. Cloud can combine both aspects of both grid computing and SOA. We also need to identify a specific development process for capturing requirements, design and implementation strategies, security, and testing cloud applications. The cloud computing paradigm has lots to offer, but at the same time we need to consider building a secured and resilient architecture, and services that are reliable and trustworthy. In this project, a generic component model and a Web service component model has been developed, meeting the design demands for building cloud application architectures. In this research we have also proposed architectural composition strategies which can be customised for various cloud services. A case study on Amazon cloud EC2 has been designed based on a software component model for cloud computing. The results show a number of good practice design guidelines leading to a satisfaction index which is promising.

Design of Service Components

Some of the main reason for the emergence of service components is for customisation through interfaces, supporting reuse through extensibility by applying building-block concepts, and interoperability, for distributed cloud components. Service level components should support communication and exchange of messages to different systems and services on-the-fly, and therefore, componentising services will satisfy those criteria. Web services and SOA have been well established in the past few years with new technologies and architectures supporting service-oriented paradigms explicitly in the process, and they have also been proven to be a good design model. This section discusses service component models for cloud services and also
discusses design rationale. Cloud applications development should primarily focus upon user perspective, their risks, and the design and architectural models should reflect user needs and their risks.

Component models and their architecture provide a framework for system composition and integration. A generic component model that is presented in this article provides a unique concept of two distinct set of services: provide and requires. Software components are the basic unit of artefact that supports service composition with the cloud computing architecture and its environment. However, each development paradigm and application demands customisable and extendable component architectures that suit the needs of their applications. Each Web service component interfaces are mapped onto different ports within architectural layers to request for services and offer services as, and when, required at run-time. Service component architecture (SCA) is a set of specifications which describes a model for building applications and systems using a service-oriented architecture (SOA) [REF-2]. SCA extends approaches on software components and builds on open standards such as Web services.

The aim is to map business requirements onto a service component that can be designed and implemented. A service component can be defined as the one that configures a service implementation. A service component model (UML based service model) is shown in Figure 8 which reflects service component design principle with a number of plugIn type interfaces that allows it to connect other service components, service provider-type of interfaces (IServiceInterface1, 2, etc.) and IServiceContract interface, which is a unique concept in our design that allows you to build and reuse business rules. The other types of interface include EntryPort, RejectedMessagePort, and ExitPort. These interfaces reflect WSDL descriptions and can be automatically generated.

![Service Component Model](image)

Figure 1 – Component model for SaaS.

Service component interface design is a unique solution to much of the larger problem of enterprise spaghetti (connection between system flows across multiple systems thus creating a spaghetti-like connection structure which is unmanageable hence the reason for moving to the concept of Enterprise Service Bus). The ESB builds on the concept of MEM (message-oriented middleware) to provide a flexible, scalable, standards-based integration technology for building a loosely coupled, and highly-distributed SOA [REF-3]. The service contract interface IServiceContract is a complex class as it allows us to build component rules and be able to reuse them in another service implementation where the similar design contract applies. Due to the nature of service complexity, we have designed an independent service component for handling SLAs as shown in Figure 8. The service contract component model provides plugin interfaces IInBoundContracts which allows a service component to take business contracts/rules as input to the component whereas the provider interface such as IoutboundContracts provides business contract services to other service components. The IQoSContracts...
service provides service contracts on quality of service rules that are embedded within the service component implementation.

![Component model for service contract interface](image)

Figure 2 – Component model for service contract interface

The service component modelling and design provides a systematic approach to building cloud service components to allow on-the-fly configuration, to discover new business services, and to be able to connect and disconnect service compositions. Service composition is one of the key principles of service design which can’t be achieved without a component-based approach. The design principle of component interface allows service flexibility, elasticity, and scalability. A service composition is defined as the development of customised services by discovering, integrating, and executing existing services. Design of service composition is not only to consume services but also to provide services. Cloud service orchestration layer and its principle can also be addressed and achieved using service composition when services are designed as components based on the model as shown here.

Service composition and orchestration allows service level reuse to happen. Service reuse is a notion of designing services as generic as possible to be reused in another service invocation. Designing services for reuse is based on SOA design principles:

- Loose coupling is to limit dependency between service consumers and service providers. This can be achieved by service interface design which has been part of a service component model as discussed.

- Autonomy is the key principle that enables service reuse. This can be achieved by designing services that can manage their own resources as database and legacies and to maintain by themselves without depending on other services. Service autonomy facilitates service adoption, scalability, QoS, SLA, and virtualisation.

- Statelessness is the property of a service to have a context but it will not have any intermediary state waiting for an event or a call-back.

- Granularity has been a prominent design principle of reuse. A large granularity of service component which is self-autonomous can yield higher level of service reuse through service composition. However, a balance must be struck when designing service components and interfaces.

- Composability is the process by which services are combined and integrated to provide a comprehensive and composite service. This principle is also the key to achieving cloud orchestration. A composite service consists of an aggregation of services that can produce another reusable service(s).
Discoverability is an important means of mandating service time (design time reuse and runtime discoverability) notion when designing service components so that component can be called on when required. Service component interface concept allows components to be discovered and connected.

Designing reusable services can save cost as it has been a well-known benefit of reuse. Cost reduction is one of the key aspects of cloud computing which aim to reduce cost for consumers by allowing pay-per-use cost model. The design rationale and service component model discussed in this section will help to improve cloud service reuse experiences. A number of SCAs were developed for various layers across three cloud services, SaaS, PaaS, and IaaS, and have been used to analyse the design of a large scale Amazon EC2. For more details on detailed design guidelines and the large scale Amazon EC2 can be found in REF-5.

Based our research experiment, we collected those number of components for Amazon EC2 architecture and services it aims to provide. We have also assessed our components against the set of good design characteristics and guidelines. The result shows a number of component guidelines satisfaction index level (we have introduced the notion of guidelines satisfaction level as the measure in percent of a number of good practice guidelines have been met by a component and its interfaces) as percent for Amazon EC2 components for SaaS, PaaS and IaaS Based on our design guidelines and component models,. This has been illustrated in Chart 1.

![Component guidelines satisfaction index](image)

As shown in Chart 1, best practice design guidelines can help us to improve the implementation of service component quality and thus providing high QoS when delivering services. We believe our approach can be applied across different service components and application domains.

**Conclusion**

Cloud computing has emerged supporting cost effective computing and IT systems. Designing cloud services is complex and need to be crafted systematically with good design principles that support cloud criteria and characteristics. Componentisation of cloud services offers software scalability and resiliency. This work has demonstrated a possibility of designing them systematically with specific process and components which can support effort and cost saving. Our previous work software components and best practice guidelines have
lead to development of embedding best practice and reuse knowledge to be built-in rather than adding them at a later stage in the development. This work has also combined best practice design principles on object-orientation, componentisation, design patterns for composing architectures, service-oriented design strategies, and cloud computing design strategies. All of which has lead to the development of various component architecture for cloud services presented in this article. For cloud computing, the characteristics such as security of services, failover, and availability, can only be achieved if the cloud services are to be developed with those characteristics that can be built-in. Hence, the reason for a number of component based framework and component-based development process for cloud computing architectures have emerged in this project.

References


Dr. Muthu Ramachandran

Dr Muthu Ramachandran is currently a Principal Lecturer in the Computing and Creative Technologies School as part of the Faculty of Arts, Environment and Technology at Leeds Metropolitan University in the UK. Previously, he spent nearly eight years in industrial research (Philips Research Labs and Volantis Systems Ltd, Surrey, UK) where he worked on software architecture, reuse, and testing. Prior to that he was teaching at Liverpool John Moores University and received his PhD from Lancaster University. His first career started as a research scientist from India Space Research Labs where he worked on real-time systems development projects. Muthu is an author of two books: Software Components: Guidelines and Applications (Nova Publishers, NY, USA, 2008) and Software Security Engineering: Design and Applications (Nova Publishers, NY, USA, 2011). He is also an edited co-author of a book, Handbook of Research in Software Engineering (IGI, 2010) and has edited the book KE for SDLC (2011). He has also widely authored published journal articles, book articles and conferences materials on various advanced topics in software engineering and education. He received his Master’s from Indian Institute of Technology, Madras and from Madurai Kamaraj University, Madurai, India. He is a member of various professional organizations and computer societies: IEEE, ACM, Fellow of BCS, and Fellow of HEA.

Muthu can be reached at M.Ramachandran@leedsmet.ac.uk

SOA speaker profile link: http://www.servicetechsymposium.com/speakers2012.php

Contributions

- Service Component Architecture for Building Cloud Services