A CCRA based Mass Customization Development for Cloud Services

Conference Paper · June 2013

DOI: 10.1109/SCC.2013.113

6 authors, including:

Bo Hu
Wuhan University
11 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Yutao Ma
Wuhan University
43 PUBLICATIONS **246** CITATIONS
SEE PROFILE

Liang-Jie Zhang
Institute of Electrical and Electronics Engineers
147 PUBLICATIONS **1,386** CITATIONS
SEE PROFILE

A CCRA based Mass Customization Development for Cloud Services

Bo Hu[1,2], Yutao Ma[3*], Liang-Jie Zhang[1], Chunxiao Xing[2], Jun Zou[1], Ping Xu[1]

1. Kingdee Research, Kingdee International Software Group Co. Ltd., Shenzhen 518057, China
2. Research Institute of Information Technology, Tsinghua University, Beijing 100084, China
3. State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China
*E-mail: ytma@whu.edu.cn

***Abstract*—With the incredible popularity of cloud computing, the adoption of mass customization (MC) is significant for building a cloud computing system that could provide services provisioning in a manner of multi-tenancy. Because of lack of a standard architecture that supports MC development for cloud services, the existing metadata or model driven approaches have insufficient abilities to realize personalized requirements with mass production when applied to product development in large-scale enterprises. Aiming at these problems, this paper presents a novel MC-based development approach for enterprise-level business cloud services based on the specification of the Cloud Computing Reference Architecture (CCRA), and shares the practice about how the approach is applied to building Kingdee K/3 Collaboration Development Cloud (CDC). Successful practice has proved that by adopting our MC development approach, we can develop platforms and tools on the cloud at a low cost and more effectively.**

Keywords-reference architecture; mass customization; cloud computing; domain modeling; collaborative development

I. INTRODUCTION

Mass customization (MC) of software is analogous to its ancestor in marketing and manufacturing industries, which focuses on the means of efficiently producing and maintaining multiple similar software products, exploiting what they have in common and managing what varies among them [1]. It is a powerful engineering model that enables software organizations to manage their software product lines more effectively. However, for the early attempts to realize MC of software, benefits were not highlighted immediately, on the contrary costs kept increasing to millions of dollars, mainly because technical and architecture barriers of adoption always made adoption time often measured in terms of years [2].

From the perspective of software development, the research and practice of MC sprang up after the rise of SOA (Service-Oriented Architecture). It includes a set of principles and methodologies for designing and developing software in the form of services. These services have well-defined business functionalities that are built as standardized software components which can be reused for different purposes. Hence, MC of software would profit from SOA with the objective to integrate software systems of different partners in a loose and interoperable manner through the use of standard interfaces and services. Until now, some context-specific IT services have successfully been provided for various industries with SOA-based approaches to MC of software [3], which offers us valuable experiences and lessons learned from a large number of practical case studies.

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet) [4]. With the incredible popularity of cloud computing, the adoption of MC is a core principle for building a cloud computing system that could provide services provisioning in a manner of multi-tenancy. MC in cloud computing is more natural and flexible, and offers more potential competitive advantages than traditional object-oriented software development. It enables mass standardized production of cloud services while customized delivery in order to meet individual customer's personalized requirements. For example, Salesforce.com has been deemed as a model that takes MC to such a new height with innovative offerings such as a Platform as a Service (PaaS) named Force.com.

However, only a few development platforms such as Force.com could be recognized within the community of IT industry as well as academia, implying that the successful practices of Salesforce.com may be unsuitable for other enterprises with different sizes. We argue that the dilemma emerges for the reason of lacking industrial standards of fundamental architecture supporting the MC of cloud services, and due to that current metadata or model driven methods for MC of software also have shortcomings for large-scale enterprise cloud service development [5]. For example, how to make a rational tradeoff between customers' personalized requirements and general SOA-based solutions so as to achieve the flexibility of on-demand services with multi-tenancy [6] remains challenging for software engineers.

Aiming at the above-mentioned problems, the goal of this paper is to present a novel MC-based development approach for enterprise-level business cloud services based on the specification of the Cloud Computing Reference Architecture (CCRA) drafted by The Open Group (TOG), and to share our practice about how this approach was applied to building Kingdee K/3 Collaboration Development Cloud (CDC). The contributions of this paper could be summarized as follows:

978-0-7695-5026-8/13 $26.00 © 2013 IEEE
DOI 10.1109/SCC.2013.113

(1) It proposes the framework and primary components of a cloud computing architecture, which has been adopted by TOG as a standard reference architecture (CCRA) for enterprises to facilitate constructing their open cloud ecosystems;

(2) It presents a CCRA-based domain modeling method that engineers a general cloud service model, which could implement cloud services customization with less development costs;

(3) It introduces the practice of developing an enterprise-level business cloud services platform (Kingdee K/3 CDC) in the light of our method, which could be a sound example of case study for other enterprises.

The remainder of this paper is structured as follows: Section II introduces related work; Section III presents the framework and primary components of a reference architecture for cloud computing, which underlies the MC-based development method for cloud services; Section IV depicts the method in detail, and elaborates on how service customization could be implemented by means of domain models; The implementation of our method will be briefly presented in Section V; In the end, Section IV concludes this paper and puts forward future work.

II. RELATED WORK

A. Architecture for Cloud Computing

Although the concept of cloud computing has widely been recognized within the community of IT industry and academia, the fundamental architecture of cloud computing is still under discussion. As far as we know, a few famous IT companies only released their top-level or conceptual architecture for cloud computing in consideration of commercial competition. However, developing a cloud ecosystem rather than a single SaaS (Software as a Service) or PaaS application would demand for a reference architecture and detailed guidance.

In order to address the deficiencies in the current state of the art, some researchers began to propose different architectures for cloud computing with intrinsic support for cloud services provisioning and delivery. Tsai *et al.* proposed a Service-Oriented Cloud Computing Architecture (SOCCA) [7] so that clouds can interoperate with each other, and presented high level designs to better support multi-tenancy feature of cloud computing. However, the feasibility of this architecture hasn't been proven in practice. Rochwerger *et al.* proposed a modular and extensible cloud architecture for open federated cloud computing [8], which would enable providers of cloud infrastructure to dynamically partner with each other to create a seemingly infinite pool of IT resources. However, because the Reservoir project is in its early stages, the usefulness of this architecture actually remains still unclear.

Zhang *et al.* proposed a reusable and customizable cloud computing open architecture (CCOA) [9] by bridging the power of SOA and virtualization in the context of cloud computing ecosystems, and presented seven architectural principles and ten interconnected architectural modules of the architecture. Although CCOA was applied only to two case studies on infrastructure and business cloud, it laid a solid foundation for our work. Then, Liu *et al.* proposed a reference architecture of cloud computing [10] based on CCOA and TOG SOA reference architecture, and defined its objective and primary principles. Compared with the initial draft of CCRA, this paper will present an improved version that focuses on the refinement of CCRA's primary components by combining cloud computing best practices in Kingdee, one of the biggest software vendors in the Asian-pacific region.

B. MC of Service-oriented Software

Component-based software development (CBSD) is a reuse-based approach to defining, implementing and composing loosely coupled independent software components into systems with lower costs [11]. As we know, it becomes mature after almost 20 years of development. So, software component-based MC is not an initiate innovation any more. In the paradigm of MC production in terms of CBSD, components are deemed as the critical elements so that resources and functions of products are always encapsulated as atomic, standard and modularized components, and these components can be assembled as much more coarse-grained components or products following a specific type of MC methodology, e.g. software product line (SPL).

Software service industries are also waking up to the power of a MC orientation. Business needs for SaaS and PaaS applications entail a higher requirement for production and deployment, which is as essential to a service vendor's success as system security and the scalable, modular, and multi-tenant architectural imperative. In contrast to older approaches to delivering component-based software, cloud services with new delivery model such as SaaS, PaaS, and Infrastructure as a Service (IaaS) promote multi-tenancy as a tool to exploit economies-of-scale. Hence, how to achieve the flexibility of multi-tenancy cloud services in the light of MC-based development approach became a popular research topic within the community of services computing.

Ruehl *et al.* [12] analyzed how SPLs can be utilized to create highly customizable SaaS applications, and proposed an architecture model to support such a development process. However, the adaptation and extension of SPL to this new field of applications remains unsolved in this paper. In [13], they improved the work presented in [12] with respect to functional and deployment variability, and the proposed description approach for deployment variability allows the customer to choose a policy from four different deployment models, which they may apply to each component of a SaaS application they use. Hosono *et al.* proposed a more detailed solution that has 4 steps toward MC, including domain development, pattern development, asset retention, and asset reuse [14]. Unfortunately, no reports showed that these methods have successful or partially been applied to the development of SaaS applications in practice.

In addition to SaaS applications, MC on PaaS applications has also attracted the attention of researchers [15]. Because a PaaS application provides a computing platform and a solution stack as a service, the consumer

could create a SaaS application using tools and/or libraries from a service provider and control software deployment and configuration settings. Che *et al.* proposes the framework and core technologies of a cloud-based service platform for MC by the advantages of cloud computing [15]. Actually, it is more likely to be a vision rather than a practical method. From the perspective of configuration management, Schroeter *et al.* analyzed the requirements for applying methods from SPL engineering to configuring cloud-based multi-tenant aware applications, and proposed a dynamic configuration management method [16] that allows for reconfiguration of variants as stakeholders' objectives change. Furthermore, Hosono *et al.* proposed an application lifecycle kit [17] that embraces both design and operation for MC on PaaS applications, which has yet been realized.

III. CLOUD COMPUTING REFERENCE ARCHITECTURE

A. Overview of CCRA

The specification of CCRA is now a base document of TOG standard for cloud computing architecture we have contributed. Experience and lessons of cloud computing research and development in IT industry indicate that technical standards are essential for constructing cloud ecosystems or open platforms. CCRA is one of cloud computing standard initiatives from the perspective of architecture, and it is designed to facilitate and ultimately enable automation and streamlining the process of modeling and documenting, making architectural design, and implementation decisions in the implementation of a flexible, extensible and reusable architecture when constructing cloud computing offerings (e.g. cloud services such as SaaS, PaaS, IaaS, and Business Process as a Service (BPaaS)), value added services and other cloud-based solutions.

According to the definition of cloud computing, we recognized that cloud computing involves a set of key characteristics to address resources sharing on the Internet or an intranet in terms of business requirements, including on-demand self-service, ubiquitous network access, location independent resource pooling, rapid elasticity and pay per use. Within the context of these key characteristics, we believe that virtualization and SOA would be two critical and essential enabling technologies for cloud computing [9]. The former could make resources sharing independent from specific infrastructure, available and elastic, and the latter ensures on-demand accesses to shared resources through the network. In our opinion, cloud computing inherits and keeps most of the techniques in SOA, with a more and extended emphasis on virtualization, multi-tenancy, on-demand service provisioning and subscription, and interoperability. And what's more, services offered in cloud computing environments have diversified delivery styles including IaaS, PaaS, SaaS and even BPaaS.

Therefore, the SOA Reference Architecture (SOA RA) international standard [18], also initiated by TOG, was chosen to serve as a good starting point to build CCRA. SOA RA is a layered architecture from the perspectives of both consumer and provider with cross-cutting concerns describing those Architecture Building Blocks (ABBs) and principles that support the realizations of SOA. It is often used for understanding the different elements of SOA, deployment of SOA in various enterprises, the basis for an industrial or organizational reference architecture, implications of architectural decisions, and the positioning of vendor products in SOA context. Layers of SOA RA include five horizontal layers (i.e. operational systems, service components, services, business processes and consumer interfaces), and four vertical layers (i.e. governance, information, Quality of Service (QoS) and Integration). The five horizontal layers are more functional in nature and relate to the functionality of SOA solutions, while the vertical layers are supportive of cross-cutting concerns that span the functional layers but are clustered around independent notions themselves as cross-cutting concerns of the SOA architectural style. In SOA RA, ABBs are the atomic elements that can be selected and combined to form an architecture to satisfy the specific enterprise requirements. Each layer contains a set of ABBs that define the key responsibilities of that layer. In addition, ABBs are connected to one another across layers and thus provide a natural definition of the association between layers.

Because of the above considerations, CCRA has also been designed as a layered architecture based on SOA RA. As depicted in Figure 1, the architecture consists of nine layers, and each layer represents a key cluster of considerations and responsibilities that typically emerge in the process of designing a cloud computing architecture. According to the different roles each layer has played, these nine layers could be classified into horizontal (or functional) layers and vertical (or supporting) layers. On one hand, the five horizontal layers are designed to enable the business capabilities required by cloud services running on the architecture. On the other hand, the four vertical layers sustain the functionalities provided by those horizontal layers.

B. Primary Components of CCRA

Due to space limitations, here we would like to give a brief introduction to the five horizontal layers, and much more details relevant to this paper will be dwelled on in the next section. The five horizontal layers are listed as follows:

(1) Cloud Infrastructure Layer: prepare underlying resources for a cloud and provide capabilities that make these resources available, dynamic and elastic

Obtaining resources on demand through the network with low price should be one conclusive factor when consumers begin to consider the adoption of cloud computing. These resources can be hardware infrastructures (including computing, storage, network, and other resources), software infrastructures (including operating systems, database management systems, middle-ware software, etc.), and legacy systems or applications. On the other hand, resource providers also want to deliver their offerings as services with high quality to consumers at a low cost, and virtualization and its management techniques could translate these promises into reality. Virtualization techniques provide resource providers powerful means to set up a flexible environment which can be configured and expanded much more easily than a physical system, and virtualization

management techniques are able to ensure the delivery of robust and cost-effective services to customers.

Therefore, the Cloud Infrastructure Layer provides the abilities to set up hardware and software infrastructures, to make these physical infrastructures virtualized, consolidated and treated as resource pools, and to monitor and manage these virtualized resources to ensure an available and dynamic provisioning.

(2) Cloud Component Layer: implementation for cloud services and their operations

Virtualized resources should be encapsulated as cloud components and implemented as cloud services before they can be provisioned to customers. In CCRA, cloud components are the implementations of cloud services, and a cloud component could realize at least one or more cloud services in terms of functionality and quality, and bind a service contract/specification to the resources of the service in the Cloud Infrastructure Layer. According to the different types of resources, a cloud component always falls into three categories, namely technical component, functional component and data component. A technical component provides the abstraction of an infrastructure to support other categories of components. A functional component provides business functionality and aids in the realization of a cloud service, and it may be composed of technical or other functional components. A data component provides necessary data information transmitted among different functional components.

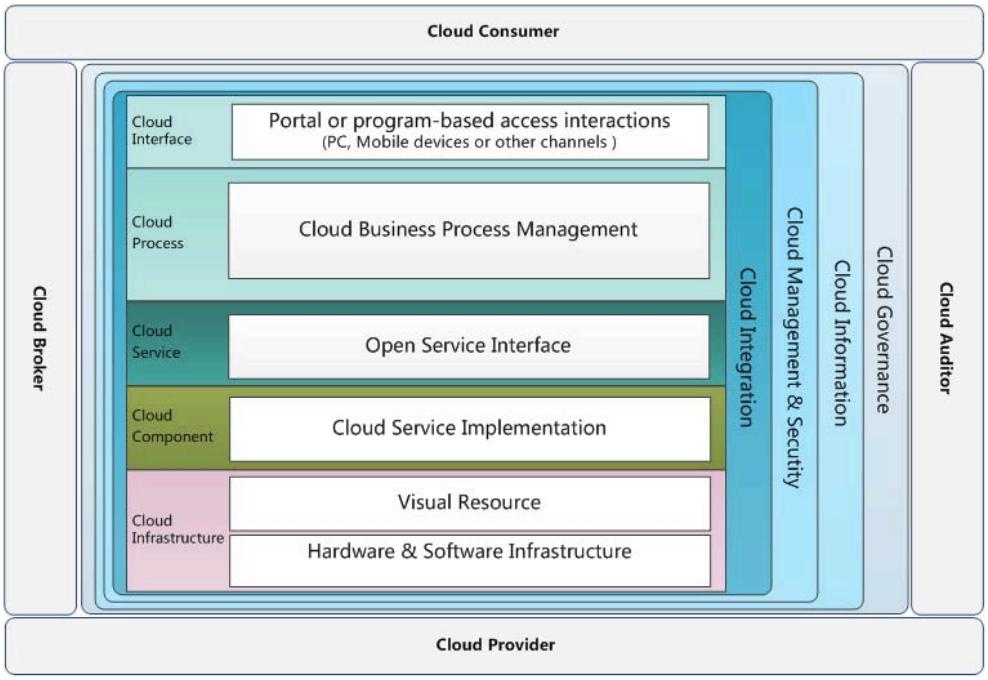

Figure 1. The framework of CCRA

Therefore, the Cloud Component Layer is a junction layer which binds the contract or specification of a cloud service in the Cloud Service Layer to the infrastructure of the cloud service in the Cloud Infrastructure Layer. It provides the abilities to implement one or more cloud services and their operations, to publish and expose those implemented cloud services, and to bind protocols to a cloud service.

(3) Cloud Service Layer: programmable interfaces enable cloud ecosystems open and prosperous

Different from the implementation of a cloud service based on cloud components in the Cloud Component Layer, the Cloud Service Layer is used to make these implemented cloud services available through application programming interfaces (APIs) and graphic user interfaces (GUIs). As it is known to all that APIs are one of the key technical facilitators of cloud computing, and most well-recognized providers of cloud computing have provided relevant APIs as well as GUIs for customers to access their services. Taking Amazon Simple Storage Service (S3) as an example, Amazon Elastic Compute Cloud (EC2) uses its interface to offer graphical interfaces for customers, while Twitter and Dropbox can take advantage of it to create new mashups or more powerful APIs. Just like the classification of cloud components, APIs of cloud services also fall into different categories, namely technical API, functionality API, data API and control API. Control APIs are special APIs which allow cloud services to be added, reconfigured, or removed in real time, either by human control or programmatically based on traffic, outages, or other factors.

The Cloud Service Layer provides the abilities that could publish and govern a cloud service, manage its lifecycle, route invocation requests when it encounters heavy traffic or overload, monitor its performance and usage, and calculate service billing.

(4) Cloud Process Layer: highly flexible cloud processes allow consumers to quickly and efficiently adapt to their changing business

Along with the development of cloud computing, it is inevitable that cloud computing will be widely applied in various industries, which presents a major problem of these cloud computing applications under changing business contexts. Like the counterpart in the SOA RA, the Cloud Process Layer is used to create a lot of highly flexible cloud processes that allow consumers' business processes to quickly and efficiently adapt to a rapid changing and increasingly complex marketing environment.

Therefore, the Cloud Process Layer provides the ability to integrate, monitor and manage business processes on the cloud.

(5) Cloud Interface Layer: it enables a cloud service to support a client-independent, channel-agnostic set of functionality

The Cloud Interface Layer is the access point for stakeholders of a cloud ecosystem, and it enables a cloud service to support a client-independent, channel-agnostic set of functionality, which is separately consumed and rendered through one or more channels (client platforms and devices).

A cloud ecosystem includes cloud services and stakeholders such as vendors, partners, and clients who provide or consume shared resources in cloud computing environments. Cloud vendors expose the interaction interfaces of their internal operations and product development capabilities to the cloud. Cloud partners provide different types of components for cloud vendors or serve as agents to provide value-added services for cloud clients. Cloud clients are the users of cloud services that offer business goal driven resources sharing. Therefore, for different stakeholders, dashboards should be provided in this layer, and each dashboard would provide a portal or a program-based access point.

IV. A MODEL DRIVEN MASS CUSTOMIZATION APPROACH

Nowadays, the emphasis of software development has shifted from coding to modeling. Model Driven Architecture (MDA) is a software design approach for the development of software systems, and supports model-driven engineering of software systems. It has widely been recognized as an efficient approach to software development within the community of software engineering. To achieve the best tradeoff between high productivity of mass production and customized services that meet personalized requirements, we need a model-driven method for implementing MC of software and even cloud services.

Considering the primary ideas of MC and MC-based development process for cloud services [14,17], based on TOG CCRA a model driven approach is presented in this section. The key points of this approach include: 1) a standard reference architecture for building enterprise-level cloud ecosystems, 2) a modular, flexible and meta-level cloud services model, and 3) customized implementations of this model for personalized requirements. In this approach, a common cloud services model was built based on domain models, and then it could be translated into different implementation instances of cloud services with simple configuration of variable parameters derived from customers' personalized requirements.

A. Dynamic domain modeling for Cloud Services

Domain Modeling is a common approach for software design and development, which could create knowledge bases comprised of a series of conceptual models of all the topics related to a specific domain. A domain model describes various entities, their attributes, relationships and other key elements within the scope of the problem domain, allows software developers and business customers to share a common basis for discussing requirements and possible solutions, and provides a convenient way to establish an application-specific model based on the modification and configuration of domain models, for instance adding or removing an element or a relationship between elements. However, it is seemed that traditional domain modeling methods may not be silver bullets for modeling complex application, such as enterprise resource planning (ERP) services. In large-scale enterprise scenarios, huge quantities of form elements, complex and changing workflows and business processes always make domain modeling tedious, laborious, time-consuming, and error-prone.

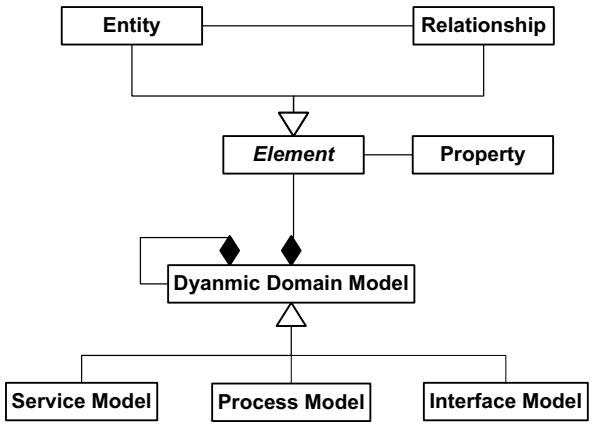

Figure 2. Abstract meta-model of dynamic domain model

Based on traditional domain modeling methods and CCRA, here we propose a Dynamic Domain Model (DDM) so as to create appropriate cloud services models in large-scale enterprise scenarios.

Figure 2 depicts the abstract meta-model of our proposed DDM in terms of the Unified Modeling Language (UML). In this model, the class *Element* plays a basic role, and it is the atomic unit of DDM; the class *Property* is used to describe the characteristics of an element. Both of the class *Entity* and *Relationship* inherit from the *Element* but focus on different aspects, respectively. An entity is used to describe an object within the scope of the domain, while a relationship is used to depict the associations between different entities. Relationships are various and specific due to their different application scenarios, which are beyond the scope of this paper and would be discussed in our following papers.

On the other hand, a DDM could be used in describing different artifacts in the CCRA. For describing cloud services in the Cloud Service Layer, the metamodel of the DDM is used to model cloud services. Under this context, entities are specified cloud components that implement a cloud service, properties are parameters used to quantify these cloud components, and relationships describe the composition relation among these cloud components. Taking Kbill, a third party electrical billing service which provides functionalities like PayPal, as an example, it is composed of several payment components from different banks. Therefore, in the service model for this electrical billing service, entities are used to specify these payment components, while properties are used to specify relevant metadata such as bank code, service identity, expiration date and values of these metadata.

In the Cloud Process Layer, the metamodel of the DDM is utilized to model cloud process in cloud computing environments. Under this context, a DDM for cloud processes is a specification of business processes. As important parts of the specification, entities are used to describe tasks in a cloud process. Each task would be bound to a specific service, and the binding relationships are specified by properties.

In the Cloud Interface Layer, a DDM could also be used to model cloud interfaces dynamically when cloud services interact with end users followed by cloud processes. Under this context, entities are used to specify buttons, textboxes, ratio buttons, dropdown list and other page controls. Properties of page controls are used to describe or quantify these page controls.

Additionally, due to the platform-independent, machine readable characteristics, the eXtensable Markup Language (XML) has been selected as a description language of our proposed DDM. Domain experts and IT engineers could make use of XML-compliant tools like XML Spy or system modeling software such as IBM Rational Rose to construct DDMs. Moreover, professional tools named Kingdee Business Operating System (BOS) designer have also been developed to support such dynamic domain modeling.

B. Model Driven Cloud Service Customization

In the implementation of MC for cloud services, we need appropriate customization methods for modeling cloud service, cloud process and cloud interface based on constructed domain models. Several patterns are utilized in customized domain modeling, and inheritance and composition are the typical ones.

Inheritance pattern means customizing a DDM by inheriting others. This pattern could be used in a context that a resulting model is constructed by covering all or most of elements derived from the base model and adding several customized elements. Furthermore, the elements from the base model could also be overridden. Figure 3 demonstrates a scenario of inheritance pattern. In the scenario, model A is the base model, which consists of two elements, namely element 1 and element 2. Model B is the resulting model, which has three elements, namely element 1, element 2 and element 3. For these three elements, only element 3 is an inherent element of the model B, and others are inherited from the model A. That is to say, the model A is the ancestor (or super-class) of the model B, while the latter is the descendant (or sub-class) of the former.

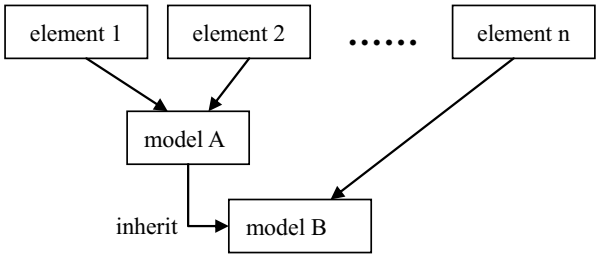

Figure 3 Inheritance pattern of mass customization

Composition pattern means customizing a DDM by means of composite elements from other DDMs. This pattern could be used in a context that the to-be-constructed model would like to contain elements from one or more DDMs, so that it can embrace these models by importing them. Figure 4 depicts the scenario of composition pattern. In this scenario, model B includes two elements, namely element 1 and element 2, and model A embraces these two elements by assembling the model B.

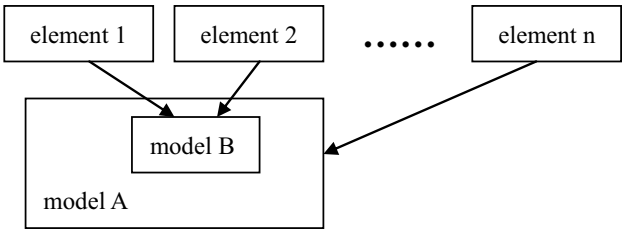

Figure 4 Composition pattern of mass customization

C. Mass Customization Development framework

In order to enable MC development, in this subsection a development framework for MC is proposed. Figure 5 shows the concept model of the development framework, which utilizes DDMs, model registry and Model Expositive engine (ME engine) as key components.

DDMs here are a series of industry-oriented domain assets constructed with the modeling approach and patterns we have introduced before. Building DDMs is often a staged process. In the early phase, common models are always the first to be established, and then industry models will be established based on the common models. Different from common and industry models, models used in enterprises and other organizations are much more specific, and they also include a large number of models. So, these models should be established at last by customization, and we call these models customization models.

All models including common models, industry models and customization models must be registered to the model

registry. The model registry is a centralized registry and repository which is used to publish and store these domain models. When the initialization of a process instance of cloud services development process is completed, cloud services developers could find a relevant domain model from the model registry, modify it to meet the real needs, and load it into the ME engine to realize an implementation of the expected cloud service. And what's more, modified domain models also could be registered to the model registry and repository in order to be reused in developing other cloud services.

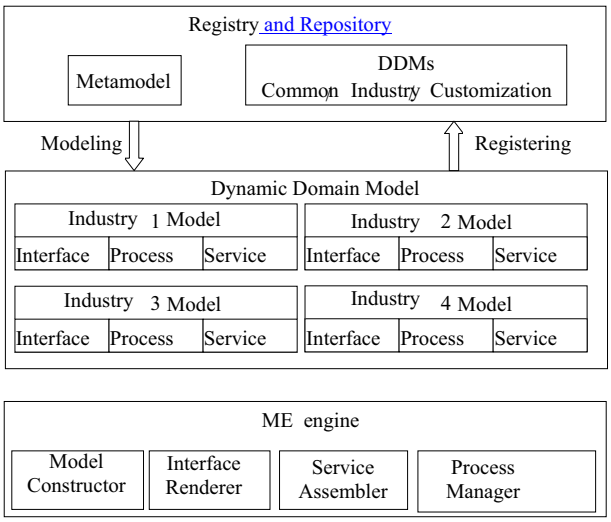

Figure 5 Mass Customization Development framework

As mentioned above, a ME engine is employed to translate customized models including service model, process model and interface model into implemented cloud services, e.g. SaaS and BPaaS. There are four key components in the ME engine, namely Model Constructor, Interface Renderer, Service Assembler, and Process manager. Model Constructor is used to construct complementary DDMs when these models are modeled by inheriting or assembling elements from other models; Service Assembler is used to assemble components as cloud services according to service models; Process Manager is used to manage and execute business processes on the cloud; Interface Renderer is used to render GUIs according to user channels (e.g., web browser, mobile devices or others).

V. IMPLEMENTATION

In order to verify and validate our proposed approach of model driven mass customization, we have constructed 28 standard domain models, covering manufacturing, warehouse, logistics, financing, human resource and other important domains for enterprises. These domain models contain nearly 624 different elements, and more than 12167 kinds of compositions among these elements and their properties. In addition, we have developed 126 cloud components and bound these components to the above-mentioned elements, which facilitates cloud services development and realization based on domain models.

Furthermore, a professional modeling tool named Kingdee BOS designer and a collaborative development platform named K/3 CDC that supports MC for cloud services have been implemented to provide an online development and runtime environment for ERP services designing, developing, testing and delivery.

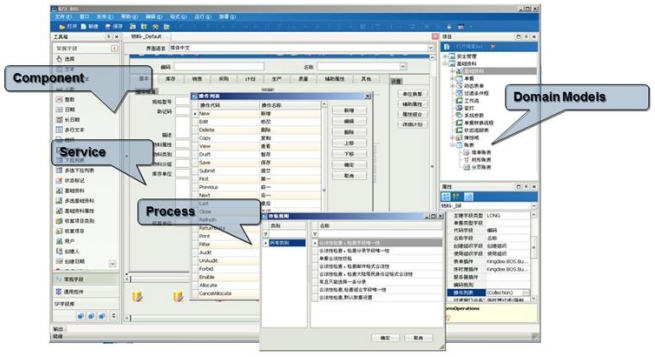

Figure 6. A practical case of dyanmic domain modeling

Figure 6 depicts a practical case of dynamic domain modeling. For this case, the Kingdee BOS designer is employed to describe an industry domain model from the Cloud Service Layer to the Cloud Interface Layer according to the CCRA. It is not an initial process of modeling an industry domain model, and the model will be established based on inheriting other common models available.

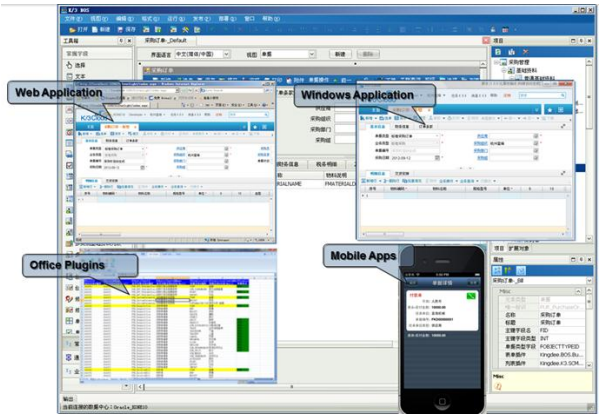

Figure 7. A practical case of model-driven cloud services development

Figure 7 depicts a model driven cloud services development case with the K/3 CDC. It could be easily found that a service developed by the K/3 CDC is channel-agnostic. The service could be accessed through different end devices, even including Microsoft office suite and mobile devices such as Apple iPad and iPhone.

Data analyzed from customer feedbacks and practical experience collected from more than 200 third-party service

vendors indicate that by adopting our MC-based development approach, customers can develop platforms and tools on the cloud at a lower cost and more effectively, and most functions of cloud services cloud be developed by 90% customization with simple configuration and 10% hard coding.

VI. CONCLUSION

Aiming at a standardized MC development of cloud services for large-scale enterprises, this paper presents a novel MC-based development approach for enterprise-level business cloud services based on TOG CCRA, and shares our practice about how this approach was applied to building the Kingdee K/3 CDC and some SaaS applications independent of end devices.

The contributions of this paper could be summarized as follows:

(1) It proposes the framework and primary components of a cloud computing architecture, which has been adopted by TOG as a standard reference architecture for enterprises to facilitate constructing their open cloud ecosystems;

(2) It presents a CCRA-based domain modeling method for engineering a common cloud service model, which could implement a cloud service customization with less development costs;

(3) It introduces the practice of developing an enterprise-level business cloud services platform (Kingdee K/3 CDC) in the light of our method, which could be a sound example of case study for other enterprises.

We will focus on domain models evolution management in the future, which includes designing effective algorithms and developing automated software tools as plug-ins of the Kingdee K/3 CDC.

ACKNOWLEDGEMENT

This work is supported by the central grant funded Cloud Computing demonstration project of China, R&D and industrialization of the SME management cloud (Project No. [2011]2448), hosted by Kingdee Software (China) Company Ltd., under the direct of National Development and Reform Committee of China; the National High Technology Research and Development Program of China (863 Program), under the grant No. 2012AA040915; the electronic information technology development foundation of Ministry of Information Industry(Grant No. GXBC [2011]506); the project of Guangdong Science and Technology under the grant No. 2012B070200014; the Shenzhen high-tech projects under the grant Nos. CXA201105060081A, CXZZ20120618164052978, and KC2012JSJS0021A; the National Science & Technology Pillar Program of China under grant No.2012BAH07B01; the construction fund of National Engineering Research Center for Supporting Software of Enterprise Internet Services.

REFERENCES

[1] C. W. Krueger, "Easing the Transition to Software Mass Customization," In Proceedings of the 4th International Workshop on Product Family Engineering, Germany, 2002, pp. 282-293.

[2] C. Krueger, "Eliminating the Adoption Barrier," IEEE Software, 19(4): 29-31, 2002.

[3] M. ter Harmsel, Mass customization as a solution for the Service Industry, School of Management and Governance, University of Twente, 2012.

[4] http://en.wikipedia.org/wiki/Cloud_computing

[5] F. Altaf, D. Schuff, "Taking a flexible approach to ASPs," Communications of the ACM, 53(2): 139-143, 2010.

[6] K. He, J. Wang, P. Liang, "Semantic Interoperability Aggregation in Service Requirements Refinement," Journal of Computer Science and Technology, 25(6): 1103-1117, 2010.

[7] W.-T. Tsai, X. Sun, J. Balasooriya, "Service-oriented cloud computing architecture," In Proceedings of the 7th International Conference on Information Technology: New Generations, USA, 2010, pp. 684-689.

[8] B. Rochwerger, D. Breitgand, E. Levy, *et al.*, "The Reservoir model and architecture for open federated cloud computing," IBM Journal of Research and Development, 53(4): Article No. 4, 2009.

[9] L.-J. Zhang, Q. Zhou, "CCOA: Cloud Computing Open Architecture," In Proceedings of the 2009 IEEE International Conference on Web Services, USA, 2009, pp. 607-616.

[10] J. Liu, L.-J. Zhang, B. Hu, K. He, "CCRA: Cloud Computing Reference Architecture," In Proceedings of the 9th IEEE International Conference on Services Computing USA, 2012, pp. 657-665.

[11] http://en.wikipedia.org/wiki/Component-based_software_engineering

[12] S. T. Ruehl, U. Andelfinger, "Applying software product lines to create customizable software-as-a-service applications," In Proceedings of the 15th International Conference on Software Product Lines, Vol. 2, Germany, 2011, p. 16.

[13] S. T. Ruehl, U. Andelfinger, A. Rausch, *et al.*, "Toward Realization of Deployment Variability for Software-as-a-Service Applications," In Proceedings of the 5th IEEE International Conference on Cloud Computing, USA, 2012, pp. 622-629.

[14] S. Hosono, Y. Shimomura, "Towards Establishing Mass Customization Methods for Cloud-Compliant Services," In Proceedings of the 4th CIRP International Conference on Industrial Product Service Systems, Japan, 2012, pp. 447-452.

[15] J. Che, Q. Zeng, S. Zhang, "Study on Cloud-Based Service Platform for Mass Customization," Advanced Materials Research, Vols. 479-481, 2012, pp. 98-101.

[16] J. Schroeter, P. Mucha, M. Muth, *et al.*, "Dynamic configuration management of cloud-based applications," In Proceedings of the 16th International Software Product Line Conference, Vol. 2, Brazil, 2012, pp. 171-178.

[17] S. Hosono, Y. Shimomura, "Application Lifecycle Kit for Mass Customization on PaaS Platforms," In Proceedings of the 8th IEEE World Congress on Services, USA, 2012, pp. 397-398.

[18] The Open Group, SOA Reference Architecture, Technical Standard, 2011.