Toward Customizable Multi-tenant SaaS Applications

by

Xin Sun

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved May 2016 by
the Graduate Supervisory Committee:

Wei-Tek Tsai, Chair
Guoliang Xue
Hasan Davulcu
Hessam Sarjoughian

ARIZONA STATE UNIVERSITY

August 2016

ABSTRACT

Nowadays, Computing is so pervasive that it has become indeed the 5th utility [1] (after water, electricity, gas, telephony) as Leonard Kleinrock [2] once envisioned. Evolved from utility computing, cloud computing has emerged as a computing infrastructure that enables rapid delivery of computing resources as a utility in a dynamically scalable, virtualized manner. However, the current industrial cloud computing implementations promote segregation among different cloud providers, which leads to user lockdown because of prohibitive migration cost. On the other hand, Service-Orented Computing (SOC) including service-oriented architecture (SOA) and Web Services (WS) promote standardization and openness with its enabling standards and communication protocols. This thesis proposes a Service-Oriented Cloud Computing Architecture by combining the best attributes of the two paradigms to promote an open, interoperable environment for cloud computing development. Mutil-tenancy SaaS applicantions built on top of SOCCA have more flexibility and are not locked down by a certain platform. Tenants residing on a multi-tenant application appear to be the sole owner of the application and not aware of the existence of others. A multi-tenant SaaS application accommodates each tenant's unique requirements by allowing tenant-level customization. A complex SaaS application that supports hundreds, even thousands of tenants could have hundreds of customization points with each of them providing multiple options, and this could result in a huge number of ways to customize the application. This dissertation also proposes innovative customization approaches, which studies similar tenants' customization choices and each individual users hehaviors, then provides guided semi-automated customization process for the future tenants. A semi-automated customization process could enable tenants to quickly implement the customization that best suits their business needs. The approach is then enhanced to enable self-adaptive customization for mobile SaaS applications, which will result experiences that are better tailored to users' babits and the capacity of their devices.

To my parents, Lijiu Sun and Lanying Jin

ACKNOWLEDGMENTS

I wish to express my great gratidue to all the people who helped and supported me tremendously during my Ph.D. study.

First and foremost, I am heartily thankful to my advisor, Prof. Wei-Tek Tsai. This dissertation would not have been possible without his continuous guidance. Not only did he teach me how to find high-impact problems, conduct effective research and write high-quality papers, he also guided me in life by teaching me effective communication, decision-making and relationship building. He has given me many advise on how to move beyond my comfort zone to reach my full potential. I would benefit from all these for the rest of my life.

I am thankful to my other dissertation committee members, Dr. Guoliang Xue, Dr. Hasan Davulcu, Dr Hessam Sarjoughian, for their constructive feedback and suggestions for my research and dissertation. Their feedback and suggestions made my dissertation more complete and accurate.

Furthermore, I want to thank my fellow research labmates, especially Dr.Yinong Chen, Qihong Shao, Qian Huang, Wu Li, Peide Zhong, Zhibin Cao, Bingnan Xiao, Xingyu Zhou and Guanqiu Qi. I cherish all the discussions I had with them on research problems and the memorable fun time with them such as dining out or karaoke nights. It was them who made my Ph.D. student life more productive and fun.

Finally, I would like to thank my mom and dad for their unconditional love, support and encouragement. Without their love and support, I could not finish my PhD study. I love you, mom and dad!

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

1 SERVICE ORIENTED CLOUD COMPUTING ARCHITECTURE

Clouds have emerged as a computing infrastructure that enables rapid delivery of computing resources as a utility in a dynamically scalable, virtualized manner. The advantages of cloud computing over traditional computing include: agility, lower entry cost, device independency, location independency, and scalability [3].

There are many cloud computing initiatives from IT giants such as Google, Amazon, Microsoft, IBM as well as startups such as Parascale [4], Elastra [5] and Appirio [6]. However, there exist many different interpretations of what cloud computing is. This chapter attempts to establish the connections between SOA and cloud computing by presenting related issues, and proposes a Service Oriented Cloud Computing Architecture (SOCCA).

This chapter is organized as the follows: Section 1.1 provides a brief survey on cloud computing hierarchy and presents a survey on existing cloud computing architectures and their issues; Section 1.2 proposes the SOCCA; Section 1.3 discusses multi-tenancy architecture; Section 1.4 application development on SOCCA; Section 1.5 shows an initial prototype and experiment; Section 1.6 concludes this chapter.

1.1 A Survey on Cloud Computing

1.1.1 A Hierarchical View of Cloud Computing

Most of the current clouds are built on top of modern data centers. It incorporates Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), and provides these services like utilities, so the end users are billed by how much they used. Figure 1 shows a tiered view for cloud computing.

Data Centers: This is the foundation of cloud computing which provides the hardware the clouds run on. Data centers are usually built in less populated areas with cheaper energy rate and lower probability of natural disasters. Modern data centers usually consist of thousands of inter-connected servers.

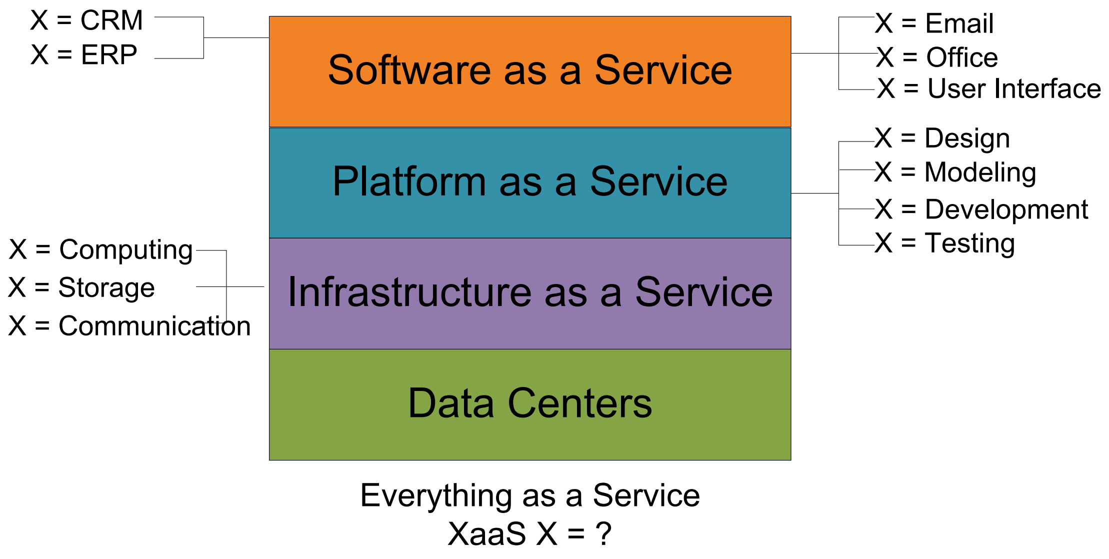


Figure 1 Hierarchical View of Cloud Computing

Infrastructure as a Service: Built on top of data centers tier, IaaS tier virtualizes computing power, storage and network connectivity of the data centers, and offers it as provisioned services to consumers. Users can scale up and down these computing resources on demand dynamically. Typically, multiple tenants coexist on the same infrastructure resources [3]. Examples of this tier include Amazon EC2, Microsoft Azure Platform.

Platform as a Service: PaaS, often referred as cloudware, provides a development platform with a set of services to assist application design, development, testing, deployment, monitoring, hosting on the cloud. It usually requires no software download or installation, and supports geographically distributed teams to work on projects collaboratively. Google App Engine, Microsoft Azure, Amazon Map Reduce/Simple Storage Service are among examples of this tier.

Software as a Service: In SaaS, Software is presented to the end users as services on demand, usually in a browser. It saves the users from the troubles of software deployment and maintenance. The software is often shared by multiple tenants, and automatically updated from the clouds, and no additional license needs to be purchased. Features can be requested on demand, and are rolled out more frequently. Because of its service characteristics, SaaS can often be easily integrated with other mashup applications. An example of SaaS is Google Maps, and its mashups across from the Internet. Other examples include Salesforce.com and Zoho productivity and collaboration suite.

The dividing lines for the four layers are not distinctive. Components and features of one layer can also be considered to be in another layer. For example, data storage service can be considered to be either in as IaaS or PaaS. Figure 1 suggests a hierarchical relationship among the different layers; however, it does not mean the upper layer has to be built on top its immediate lower layer. For example, a SaaS application can be built directly over IaaS, instead of PaaS.

In cloud computing environment, everything can be implemented and treated as a service. Figure 1 shows a few examples of what can be treated as a service in different tiers.

1.1.2 Existing Cloud Computing Architectures

Both academia and industry have been active on cloud computing research, and several cloud computing architectures have been proposed. In [7], IBM considers current single-providers cloud as limited resource, and the lack of interoperability among cloud providers prevents deployment across different clouds. A cloud computing architecture named Reservoir was proposed to create a federation from multiple cloud providers, which acts as a global fabric of resources that can guarantee the required SLA. In Reservoir architecture, the computational resources within a site are partitioned by a virtualization layer into virtual execution environments (VEEs). A service application is decomposed into a set of software components/services running on VEEs on the same or different VEEs within a site or across from different sites. However, Reservoir architecture does not allow a component/service to run on its duplicate on different VEEs; Moreover, computing resources are abstracted as hosting service which might not be necessarily true for all clouds. In [8], a software platform for .NET based cloud computing named Aneka was introduced. Aneka is a customizable and extensible service oriented runtime environment that enables developers to build .NET applications with the supports of APIs and multiple programming models. Aneka is a service-oriented, pure PaaS cloud solution. In [1], Rajkumar and his colleagues explained a market-oriented cloud architecture in detail used by Aneka, which regulates the supply and demand of cloud resources to achieve market equilibrium, adds economic incentives for both cloud consumers and providers, and promotes QoS-based resource allocation mechanisms that differentiates service request based on their utility. The key component of this architecture is SLA (Service Level Agreement) Resource Allocator, which is consisted of Service Request Examiner and Access Control, VM (Virtual Machines) monitor, Service Request Monitor, and Request Dispatcher. Based on the feedback from VM and Service Request monitors, the dispatcher routes the requests from users/brokers to the cloud resources that can fulfill their QoS requirements. In [9], Huang and her colleagues from IBM described a service-oriented cloud computing platform that enables web-delivery of application-based services with a set of common business and operational services. The platform supports multi-tenancy feature by utilizing single application instance model. The isolation among tenants is taken care by the underline design. Other services include subscription management, federated ID management, application firewall, etc.

1.1.3 Issues with Current Clouds

Current cloud computing have the following characteristics:

- ***Users are often tied with one cloud provider***. Even though up-front cost for a cloud computing deployment is reduced and long-term lease is eliminated, much effort and money is spent on developing the application for a specific cloud platform, which makes it difficult to migrate the same application onto a different cloud. Often, migration simply may mean redevelopment. For example, applications deployed on Amazon EC2 cannot be migrated easily due its particular storage framework [10].
- ***Computing components are tightly coupled***. This can be clearly explained using an analogy. Suppose one wants a new computer, this person has the choices of either buying a ready-to-use computer from a manufacturer (buying) or purchasing the components separately and building the computer in a DIY style (building). The advantages of building over buying include wider selection of components, flexibility to customize, and cheaper cost [11]. However, as the computing resources over the Internet, current cloud implementations do not allow this kind of flexibility. If a customer opts to use Amazon S3 storage service, he is then stuck with other cloud computing services Amazon provides, such as EC2, Elastic Map Reduce.

- ***Lack of SLA supports***. Currently, SLA is an obstacle that prevents wide adoption for cloud computing. Cloud computing infrastructure services such as EC2 are not yet able to sign the SLA needed by companies that want to use cloud computing for serious business deployment [12]. Moreover, business is dynamic. Static SLA is not able to adapt to the changes in business needs as cloud computing promises to.
- ***Lack of Multi-tenancy supports***. Multi-tenancy can support multiple client tenants simultaneously to achieve the goal of cost effectiveness. Currently, one has three types of multi-tenancy enablement approaches: virtualization, mediation and sharing [13]. To achieve the full potential of multi-tenancy, three issues remain to be solved [13]:
  - *Resource sharing*: To reduce the hardware, software and management cost of each tenant.
  - *Security isolation*: To prevent the potential invalid access, conflict and interference among tenants.
  - *Customization*: To support tenant-specific UI, access control, process, data, etc.
- ***Lack of Flexibility for User Interface***. UI is an important part of the application, and user experience can be a major evaluation factor for a business application. However, cloud/SaaS users are limited with UI choices because UI composition frameworks, such as the one proposed in [14], have not been integrated with cloud computing.

1.2 Service Oriented Cloud Computing Architecture (SOCCA)

1.2.1 Cloud Computing and SOA

SOA and cloud computing are related, specifically, SOA is an architectural pattern that guides business solutions to create, organize and reuse its computing components, while cloud computing is a set of enabling technology that services a bigger, more flexible platform for enterprise to build their SOA solutions. In other words, SOA and cloud computing will co-exist, complement, and support each other.

There have been several initiatives at attempting bridging SOA and cloud computing. Noticeably, the works in [8] [9] have more service-oriented features than the other mentioned in section 1.

1.2.2 Layered Architecture of SOCCA

SOCCA is a layered architecture that has the following layers:

Individual Cloud Provider Layer: This layer resembles the current cloud implementations. Each cloud provider builds its own data centers that power the cloud services it provides. Each cloud may have its own proprietary virtualization technology or utilize open source virtualization technology, such as Eucalyptus [15]. Similar to Market-Oriented Cloud Architecture proposed in [1], within each individual cloud, there is a request dispatcher working with Virtual Machine Monitor and Service/App Governance Service to allocate the requests to the available recourses. The distinction from current cloud implementations is that the cloud computing resources in SOCCA are componentized into independent services such as Storage Service, Computing Service and Communication Service, with open-standardized interfaces, so they can be combined with services from other cloud providers to build a cross-platform virtual computer on the clouds. In order to achieve maximum interoperability, uniform standards need to be implemented. For example, SQL is de facto standard for RDBMS data management, and many database vendors have their own implementations. A cloud version of SQL needs to be defined, so data manipulation logic of an application that works on one cloud can also work other clouds. A distributed computing framework standard to unify all different implementations of Map/Reduce is also in need for the same reason.

Cloud Ontology Mapping Layer: Cloud providers might not conform to the standards rigidly; they might also have implemented extra features that are not included in the standards. Cloud Ontology Mapping Layer exists to mask the differences among the different individual cloud providers and it can help the migration of cloud application from one cloud to another. Several important ontology sytem are needed:

1. ***Storage Ontology***: It defines the concepts and terms related to data manipulation on the clouds, such as data update, date insert, data delete, and data select, etc.
2. ***Computing Ontology***: It defines the concepts and terms related to distribute computing on the clouds, such as Map/Reduce Framework.
3. ***Communication Ontology***: It defines the concepts and terms related Communication Schema among the clouds, such as data encoding schema, message routing.

Cloud Broker Layer: Cloud brokers serve as the agents between individual cloud providers and SOA layer. Each major cloud service has an associated service broker type. Generally, cloud brokers need to fulfill the following tasks:

1. **Cloud Provider Information Publishing**: Individual cloud providers publish specifications and pricing info to the cloud brokers. Important provider information includes:

- Cloud Provider Basic Information: Company Name, Company Address, Company Website, Company Contact Info, etc.

- Resource Type and Specifications: Whether it is computer/storage/communication resource and its specification and limitation. For example, for the data storage service, the data transmission rate can be as high as 2Gb/s.
- Pricing Information: How the services charge. This varies the most among different cloud provider. For example, currently, Google does not charge for the first 500MB storage, and $0.15 per GB of data after, while Amazon charges $0.11 per GB-month for its EBS Volumes service. Even within a cloud provider, the pricing info might change as the market's dynamic changes.

2. **Ranking**: Like the service brokers in SOA, cloud brokers also rank the cloud resources published. Services can be ranked in several categories such as price, reliability, availability, and security, etc. Ranking can be achieved through user voting or historical service governance records.
3. **Dynamic SLA Negotiation**: Business is often dynamic, and the IT infrastructure has to be adaptive to accommodate the business needs, therefore to achieve the optimal ROI (Return of Investment). It's often the case that the IT resources a business demands can be predicted. Cloud service brokers can help cloud users and cloud providers negotiate on a SLA dynamically.
4. **On-Demand Provision Model**: Most services experience seasonal or other periodic demand variation as well as some unexpected demand bursts due to external events. The only way to provide "on-demand" services is to provision for them in advance. Accurate demand prediction and provision become critical for the successful of the cloud computing, which reduces the waste of utility purchase and can therefore save money using utility computing. We are investigating a demand prediction model and

model the evolution of multi-tenant as a discrete time stochastic process. We have investigated several macroeconomic factors in a real mortgage service platform [16] [17], and the initial results show that the underlying stochastic process may depend on a number of external factors such as macroeconomic variables, as well as service internal features. Some analysis results from real applications demonstrate the effectiveness of our models. Due to the space limitation, more details can be found in [16]. Specifically, the process needs to answer the following question: What is the forecast of tenant many days into the future? How to predict the workload distribution at different services? How to optimize the service provision process and minimize customers' dissatisfaction?

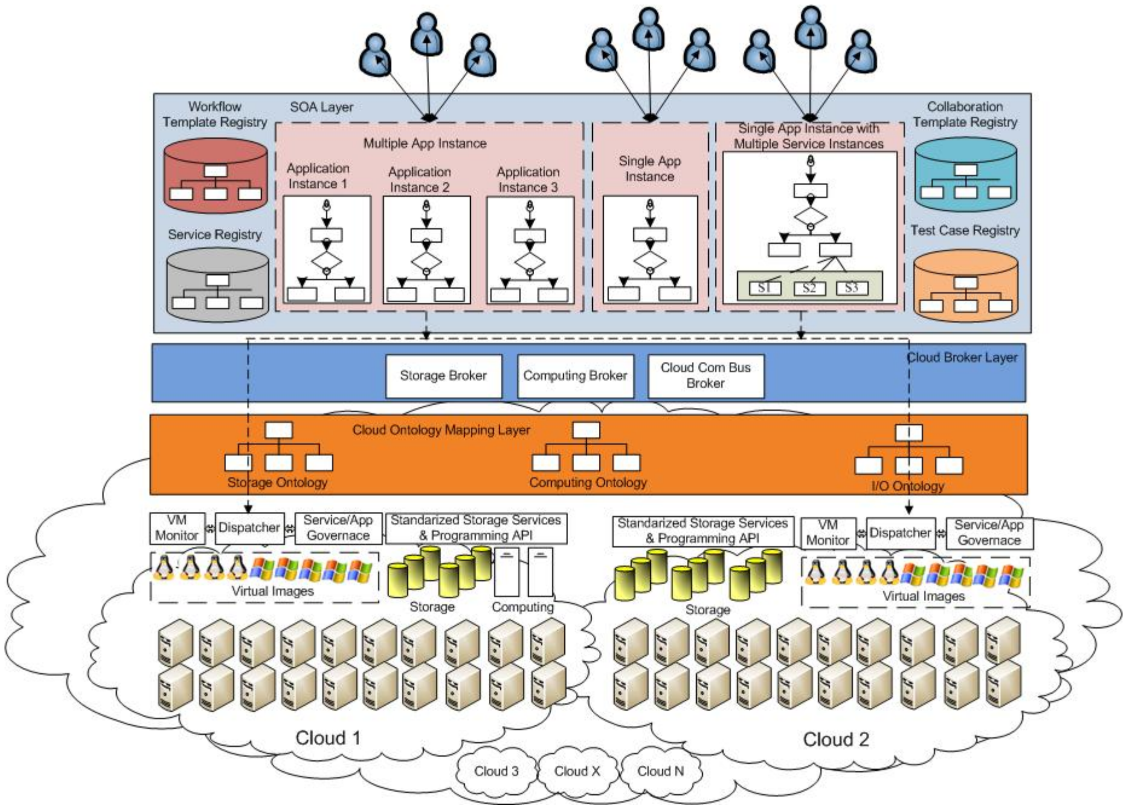


Figure 2 Service-Oriented Cloud Computing Architecture

SOA Layer: This layer fully takes the advantages of the existing research and infrastructure from traditional SOA. Many existing SOA frameworks, such as CCSOA [18], UCSOA [19], GSE [20] and UISOA [14] can be integrated into this layer. Figure 2 shows a possible SOA layer for SOCCA. Similar to CCSOA, not only services but also many other artifacts can be published and shared, such as workflow templates, collaboration templates and test cases. The registry for each type of artifacts is indexed and organized by its according ontology. The fundamental difference of the SOA layer of SOCCA from traditional SOA is that the service providers no longer host the published services anymore. Instead, they publish the services in deployable packages, which can be easily redeployed to different cloud hosting environments. Application developers can decide which clouds they want to these services to run based a set of criteria. The details will be discussed in section 1.4. Another major improvement is multi-tenancy support that allows more flexibility, which will be discussed in section 1.3. SOA layer of SOCCA allows more flexibility than traditional SOA; it further separates the roles of service providers and cloud providers, and the service logics and its running environments.

1.3 Multi-tenancy Architecture (MTA)

As shown in Figure 2, SOCCA allows 3 different main multi-tenancy patterns. In [9], the authors discussed the left two multi-tenancy patterns: Multiple Application Instance (MAI) and Single Application Instance (SAI). The authors pointed out, the former does not scale as well as the latter, but it provides better isolation among different tenants. Within SOCCA, a new multi-tenant pattern becomes possible: Single Application Instance and Multiple Service Instances (SAIMSI). The motivation behind this pattern is that the workloads among different application components are often not distributed evenly among application components, and the performance of the single application instance is limited by the application components having lower throughput. Moreover, to enhance scalability, we want to reduce unnecessary duplications as much as possible as opposed to Multiple Application Instances pattern. Figure 3 shows a simplified example. The example application is composed by A, B, S, three services with S being the computing intensive component. With S being the bottleneck to support multiple tenants, 3 instances of S are created to balance the workloads. Note that the 3 instances of the services can also reside on different clouds.

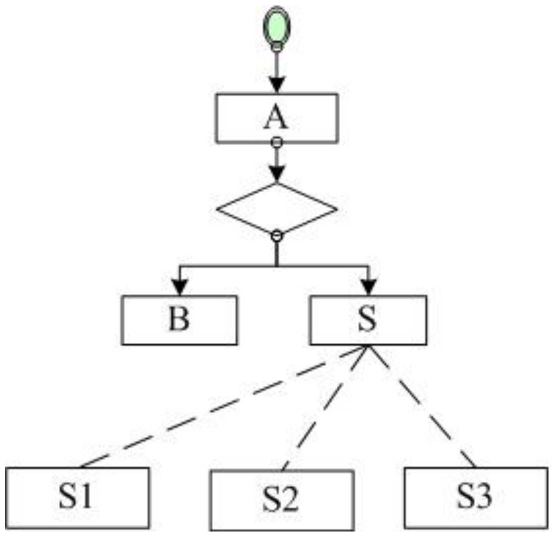


Figure 3 Single Application Instance Multiple Service Instances

Better scalability is not only benefit from the SAIMSI pattern, easy customizability is another gain. Suppose in the sample application, S is a payment service. Different tenants might have different payment method requirements, such as credit card, Paypal, or check. The application runtime environment (not described in this chapter) will direct users of each tenant to the correct service instance according to tenants' individual configuration. In the case that a future tenant has a payment requirement that cannot be met by the existing service instances, say money order, an according service instance can be easily plugged into the existing service instances group. The upcoming chapters on multi-tenancy from this dissertation will provide more details on this topic.

1.4 Application Development on SOCCA

1.4.1 Service Package

Service providers of traditional SOA develop the logic of a service and provide its running environment. In SOCCA, services are published as re-deployable packages, namely service package. A service package contains the following required/ optional information and files:

Compiled Code: If service providers only use the standard APIs and protocols, a single version of complied code is enough; if service providers optimize the performance of their services by utilizing some platform unique APIs and features, complied code for each platform is needed.

Source Code: This is optional. It is useful to help its user to understand the service better, also gives the freedom to its users to tweak the services to accommodate their specific requirement.

Configuration File: Services might use external basic services. For example, a computing intensive scientific service which also uses a lot of storage might deploy its computing logic on a cloud that provides high performance computing power, but use the cheaper storage service provided by another cloud. This requires a configuration file, which specifies the external service's locations, partner link, etc. This can also be achieved in a BPEL manner, however, since basic services such as storage services, have a widely adopted standards, and are frequently used, so it is more efficient to handle in a database connection configuration file style.

Resource Files: Any resource files that the service depends on, such as images, documents.

1.4.2 SOCCA Applications

Application development in SOCCA is similar to the development in CCSOA. Developers first search if there is a workflow template that matches the requirement. A workflow template is composed of service stubs/specifications, which specify the functionalities and interfaces of services. Later a service stub is bound with a service package. Depending on the QoS requirements and the budget for the application, cloud brokers will negotiate with cloud providers on SLA, and deploy the service packages on one or multiple clouds. An algorithm for request dispatching for a service across its deployments on different clouds needs to be applied. Figure 4 shows a typical application architecture on SOCCA.

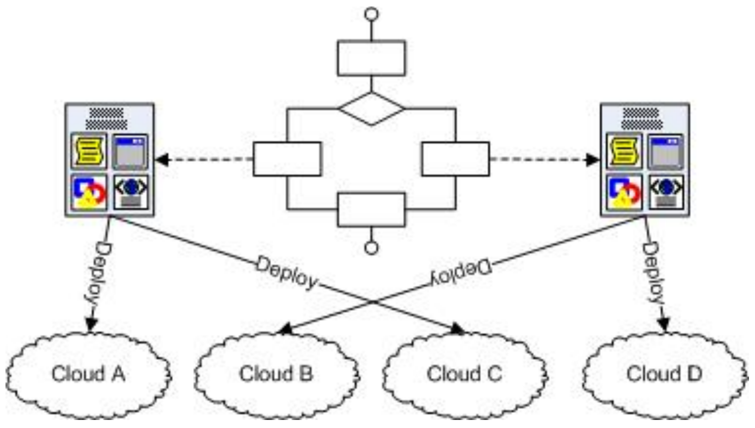
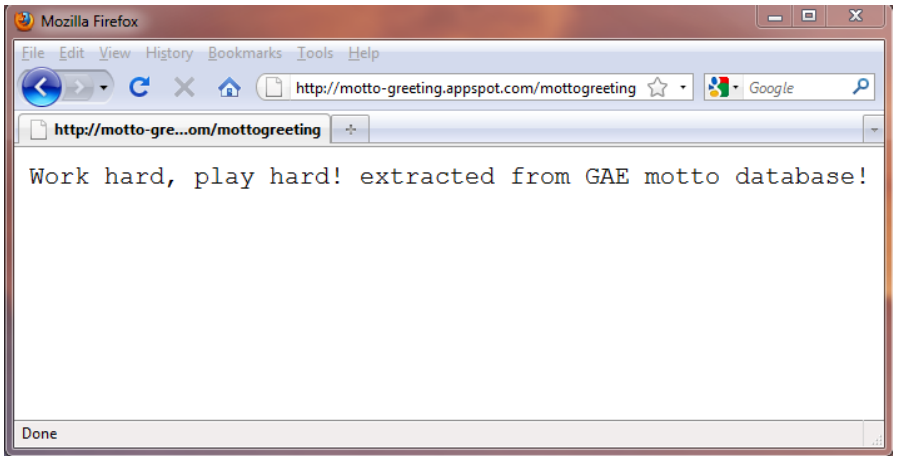


Figure 4 SOCCA Application Architecture

1.5 Initial Prototype and Experiment

This section shows an initial prototype and experiment to demonstrate the possibility of SOCCA. The demonstration web application has one easy requirement: When each user visits the web page, he/she will be greeted by a random message retrieved from a motto database.

1. We developed the application by using Google App Engine. Note that a number of mottos are retrieved and stored from and to Google cloud by using Datastore with JDO. Figure 5 shows a screenshot of motto application running on Google App Engine.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://motto-greeting.appspot.com/mottogreeting
Google
http://motto-gre...om/mottogreeting
Done

```
Work hard, play hard! extracted from GAE motto database!
```

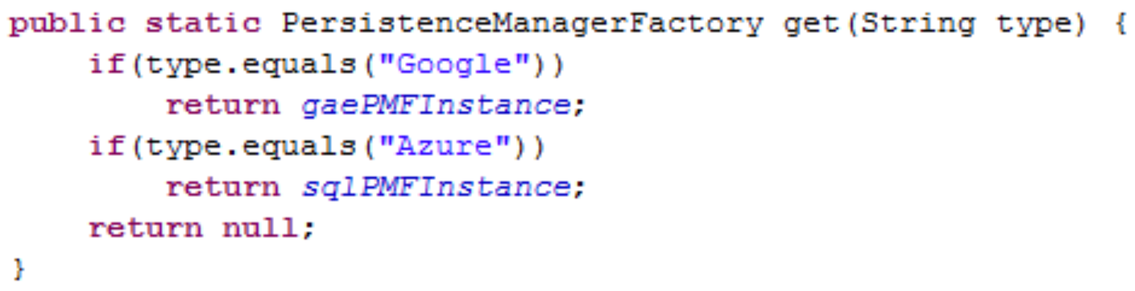
Figure 5 Motto Application with GAE Database

2. We created a web service that wraps the Azure SQL service that allows retrieving and storing mottos from the motto databases deployed on Azure.

3. We utilized the Web Service Connector (WSC) tool provided by [21] to generate Google App Engine compatible client code in java to access the web service developed by step 2. WSC is a code-generation tool that takes a WSDL file and generates an optimized java library that provides access to the web service.

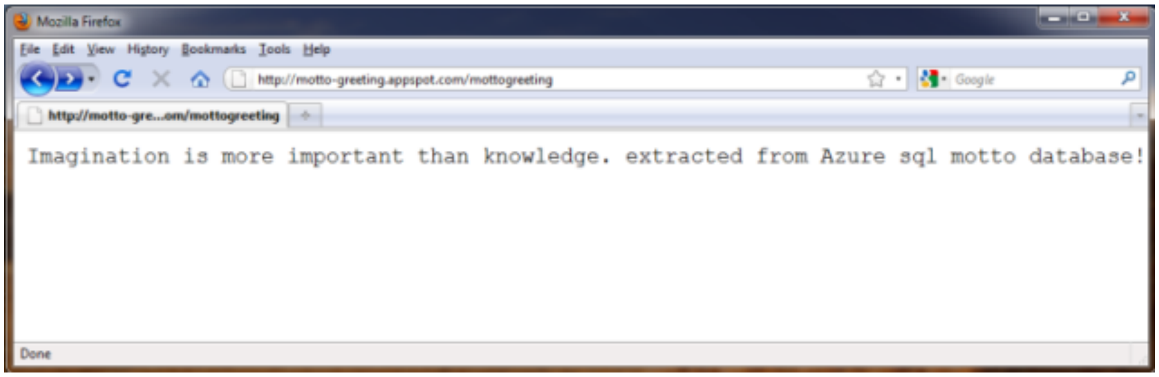
4. We created a class that implements "javax.jdo.PersistenceManagerFactory", which is the interface GAE Datastore uses to manipulate the data on the cloud.

5. We created a class called SQLPersistenceManagerFactory that implements "javax.jdo.PersistenceManagerFactory", which is the interface GAE Datastore uses to manipulate the data on the cloud. Figure 6, a code snippet, shows that depending on the config file, the application will be connected with either GAE Datastore database or Azure SQL database. Figure 7 shows that after changing the config file, the motto app is now connected with Azure SQL database.



```
public static PersistenceManagerFactory get(String type) {
    if(type.equals("Google"))
        return gaePMFInstance;
    if(type.equals("Azure"))
        return sqlPMFInstance;
    return null;
}
```

Figure 6 Code snippet for Database Service Configuration



Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://motto-greeting.appspot.com/mottogreeting
Google
http://motto-gre...om/mottogreeting
Done

```
Imagination is more important than knowledge. extracted from Azure sql motto database!
```

Figure 7 Motto App with Azure Database

The prototype demonstrates that a service package deployed on one cloud can be configured to collaborate with services from other clouds. However, it does not show that a service package can be redeployed on a different cloud and the instances for the same services can live on multiple clouds. This is due to that currently, different clouds support different language sets, and there is no powerful modeling language to support developments for multiple platforms. We are currently developing this feature using a modeling language PSML [22].

1.6 Conclusion

This chapter proposed a service-oriented cloud computing architecture SOCCA that allows an application to run on different clouds and interoperate with each other. The SOCCA is a 4-layer architecture that supports both SOA and cloud computing. SOCCA supports easy application migration from one cloud to another and service redeployment to different clouds by separating the roles of service logic provider and service hosting/cloud providers. It promotes an open platform on which open standards, ontology are embraced. The chapter also introduced related topics for future research, such as service customization, service demand prediction and service request dispatching algorithms. In chapter 2, 3, more focus will be given on how to perform customization for SaaS, mobile SaaS applications. In chapter 4, a two-tier dynamic resource allocation algorithm is discussed to solve the service request dispatching problem under SOCCA. In chapter 5, a customizable ontology-based service-oritented simuation framework for robotic studio is propsed based on SOCCA.

2 SAAS MULTI-TENANT APPLICATION CUSTOMIZATION

Cloud computing often uses the multi-tenancy architecture where tenants share system software. It is one of the key features of Software as a Service (SaaS) that enables higher profit margin by leveraging the economics of scale [23]. Tenants residing on a multi-tenant application appear to be the sole owner of the application and not aware of the existence of others. A multi-tenant SaaS application accommodates each tenant's unique requirements by allowing tenant-level customization. A complex SaaS application that supports hundreds, even thousands of tenants could have hundreds of customization points with each of them providing multiple options, and this could result in a huge number of ways to customize the application. This chapter proposes an innovative customization approach that studies similar tenants' customization choices and provides guided semi-automated customization process for the future tenants. A semi-automated customization process could enable tenants to quickly implement the customization that best suits their business needs.

2.1 Introduction

Cloud Computing has emerged as a new infrastructure that enables rapid delivery of computing resources as a utility in a dynamically scalable virtualized manner. Typically, SaaS applications allow multiple tenants to reside on single or multiple instances of the software at the same time, which helps brings down the cost. Though tenants co-exist on multi-tenant software, each of them might have its own branding, GUI, data, and workflow, etc, appearing as the sole owner of the software. A multi-tenant SaaS application is also referred as configurable SaaS, and achieved by customization [24]. A multi-tenant SaaS application usually provides a solution for common enterprise needs, which can be domain-independent, such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Human Resource Management (HRM), or domain specific, such as Inventory Management for retailers, Practice Management for medical practices. Though enterprises and businesses' requirements for software overlap greatly, each of them has some unique requirements that distinguish them from each other. Customization is needed to meet specific requirements for each tenant. Moreover, the pace of business process change is increasing ever faster with more globalized market [25]. This, in return, requires, SaaS applications to be customizable to adapt to businesses' ever changing needs.

However, customizing a complex SaaS application requires much manual work which can delay the time to market. Sometimes, tenants do not necessarily know which customization choices suit their needs the best either.

The chapter proposes an innovative customization approach that studies existing tenants' customization choices and provides guided semi-automated customization process for future tenants. The contributions of this chapter include:

- Applying hierarchical template design philosophy by modeling the customization process using orthogonal variability model
- Combining ontology and Inputs, Outputs, Preconditions and Effects (IOPE) for variants definition, organization or and discovery
- Specifically distinguishing tenants' inherent characteristics and application specific characteristics.
- A guided semi-automated customization process based on mining existing tenants' customization decisions to improve efficiency

- Consistency checking tenant's customization plan using soft and hard rules mined existing customization decisions

The chapter is organized as the following: Section 2.2 discusses the customizability of SaaS applications; Section 2.3 discusses the complexity of the workflow and service customization; Section 2.4 explains how to ontology to help discover and organize variants. Section 2.5 discusses a tenant based customization algorithm. Section 2.6 proposes a guided customization approach based on tenant mining; Section 2.7 demonstrates the customization process using a case study.

2.2 SaaS Customization

Customizability, used to describe the level of customization an application could have, typical involves the following aspects.

What: Typically, for SaaS applications, the following assets can be customized:

- **GUI**: GUI customization is the most elementary and visible customization. Typical GUI customization includes logo, theme, layout, fonts, interaction etc.
- **Workflow**: Workflow customization generally controls the business process logic. Tenants can compose their own workflow templates using existing type services, or choose the ones stored in the workflow templates repository.
- **Service**: Service customization usually involves two steps: service selection and service configuration.
    - Service Selection: In SaaS applications, service selection is the process to choose a concrete service implementation and bind with the service type used in workflow template.

    - Service Configuration: The tenant should configure the different properties a concrete service has to achieve the desired behavior and performance.

- **Data**: Data customization involves data storage, encryption, compression, schema enhancement, etc. Different multi-tenant data architectures might have different levels of isolation, sharing, and scalability. Three common data architectures are: separate databases, shared databases, separated schemas and shared databases shared schemas [26].
- **QoS**: QoS requirements are usually formalized using SLA between tenants and SaaS application providers. QoS customization is deeper level customization as it usually involves allocating corresponding amount of resource and choosing appropriate service and workflow customization, even the data architectures.

The customization assets actually follow a layered architecture with QoS affected by all the layers. The architecture can be illustrated by the following figure.

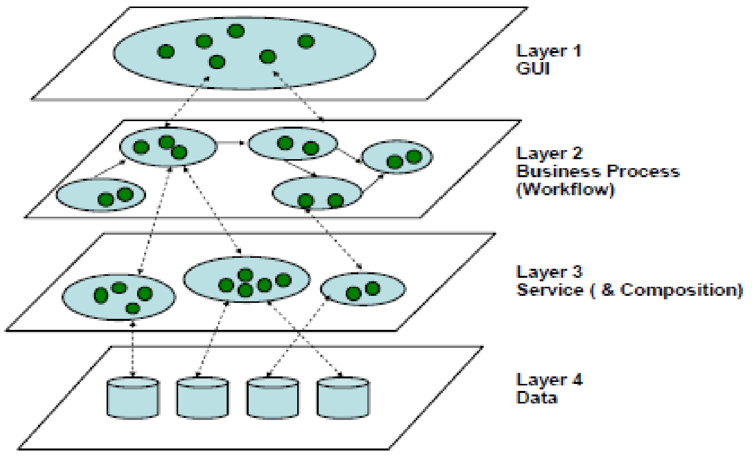


Figure 8 Layered Architecture

Who: The following parties can conduct customizations for SaaS applications:

- **SaaS application developers/designers**: The primary goal of application developers/designers is to provide a high customizability for the application. This could be achieved by the following practices:
    - Collecting the common requirements for a domain and considering possible variations.
    - Making the framework extensible.

The developers/designers could customize the application to a certain level to reduce the targeted tenants' customization work.

- **SaaS application tenants**: Considering an online retail SaaS application as an example. The application developers/designers can reduce each tenant's customization efforts by providing a number of pre-configured customized templates for merchants from different domains, such as electronics, clothing, grocery, etc. Each merchant (tenant) would continue the customization based on the chosen template.
- **SaaS application consultants**: For complex applications, such as enterprise ERP, HRM, CRM, the customization efforts could be so high that sometimes, the tenants would hire consultants specialized at customization to reduce the cost of training people to do it in-house, and to reduce the time shipped to market.
- **SaaS application users**: Users of each tenant could further customize the application according to their own personal preference. However, their customization should be rather superficial, and limited by the customizations done by the developers/designers, tenants.

How: Customization could be performed in the following manners:

- **Source Code**: For an extensible SaaS application, new source code can be added and integrated with the existing features by implementing a pre-defined interface or contract.
- **Composition**: Recomposing workflows could modify the business logics, catering different tenants' need.
- **Configuration**: Customization can be done by simply providing different configuration parameters in the config files this way.

How Easy: The level of easiness for customization could be simply categorized into the followings:

- **Manual**: Tenants have to manually make decisions at each customization point.
- **Automated**: All the customization choices are made automatically based on tenants' requirements inputs. The final customization results might not meet all tenants' requirements.
- **Guided**: For each customization point, automated customization will return a few top matching customization choices, and the tenant will manually review these choices and make a decision based on their judgment. This approach minimizes manual work greatly while eliminating unnecessary errors introduced by the automated customization.

2.2.1 Related Work

Software customization has been a research problem for a long time, however, the specific problems such as multi-tenancy, cost-awareness, present new challenges for SaaS customization. R. Mietzner etel demonstrated how to use BPEL to generate customization BPEL processes for tenants and deployment scripts to provision new tenants [27] [28]. Their approach helps SaaS developers to find a balance between catering to every single requirement of tenants and achieving economy of scale by reusing components among tenants. Zhang [29] proposed a novel SaaS customization framework using policy through design-time tooling and a runtime environment. A validation algorithm was also proposed for customization rules propagation. A grapevine model is used to provide customization recommendations to tenants by analyzing the similarities between tenants' application requirements and existing application templates. Instead of just identifying individual component, an application template has links to various components that can be used together. Thus when a full or partial match has been identified, a collection of components with their linkage information will be available for tenant developers to save time and effort. In other words, this is an approach where customization can be done on a collection of components with their linkage relationship, rather than at the component level.

Workday [30], a cloud-based enterprise application company, adopts a pure object-oriented development methodology and implements a metadata-based framework to enable non-code-based business logic changes. The metadata definitions are used to define the logic by customers. Selectica [31], a SaaS based sales and contract management software provider, uses a guided customization process to help sales people build custom deals while still enforcing the constraints set by the management. This process is similar to the AURA process where end users can participate in requirement acquisition directly. The essence of the AURA project is to ask users questions, and based on input received to ask more questions to complete a requirement model behind the scene. Selectica also used a model-based approach but a configurable one where each item may have multiple options to choose from with constraints built into the system. Thus, once all the items have been selected, a newly configured sale package is completed. Similar approach can be applied to SaaS customization. In [24], Tsai exploited a multi-layer structure of SaaS applications, analyzes their inherent relationships, as well as cross-layer relationship using ontology information. Tsai also briefly exploits a recommendation solution based on tenants' similarities. This chapter continues the work, and extends the recommendation solution.

Table 1 shows a brief comparison among these approaches.

Table 1 Customization Approaches Comparison

	R. Mietzner	Zhang	Grapevine	Workday	Selectica	Proposed approach
Model-based	yes(OVM)	can be added	yes	yes(OO)	yes	yes (OVM)
Hierarchical Workflow Design	yes	can be added	yes	can be added	can be added	yes
Guided Process	can be added	can be added	can be added	can be added	yes	yes
Consistency checking	can be added	yes	can be added	can be added	yes	yes
Recommenda tion Generation	can be added	can be added		can be added	can be added	yes
Tenant inherent characteristic s	can be added	can be added	can be added	can be added	can be added	yes
Ontology and IOPE variant filtering	can be added	can be added	can be added	can be added	can be added	yes
New template generation	yes (automated)	can be added	yes (automate d)	can be added	can be added	Yes

2.3 Complexity of Customization

According to last section, there were three approaches to perform customization: source code, workflow composition and configuration. In a mature CCSOA [18], GSE [32] like environment, there are seldom need to write new source code, or to create new services as the service repository are comprehensive enough to include almost all use scenarios. Therefore, customization is mainly consisted of workflow composition and service configuration.

2.3.1 Hierarchical Workflow Template Design

In SOA development, a workflow refers how two or more web services interact. Workflow designs often follow the template design pattern. Originally, the core idea of template design [33] is to use a template to define the program skeleton of an algorithm and one or more of the algorithm steps can be overridden subclasses to allow different behaviors. Hierarchical workflow template design (HWTD) is an application of template design for SOA software development. In HWTD, when application designers design workflows, for places where customizations are possible, they use "abstract methods" as placeholders. Application/Customization designers will implement these placeholders ("concrete methods") as sub-workflows. Sub-workflows can still have "abstract method". In SOA, these placeholders are actually, atomic or composite type services. Type services/service specifications define service interfaces, functionalities. Since most SaaS apps use SOA model and services are loosely coupled, there is no compiling, linking. Therefore, hierarchical workflow template design does not require OO features, such as inheritance and polymorphism.

2.3.2 Orthogonal Variability Model

Hierarchical workflow template design can be modeled by Orthogonal Variability Model [28]. In OVM, a variation point (VP) documents a variable item. A variant (V) documents the possible instances of a variable item. In workflow customization, a variation point is a component that can be replaced by one or multiple sub-workflow templates. A variant is sub-workflow template. A sub-workflow can still have variation points.

Definition:

Customizable Workflow: A workflow that has at least one VP. A single VP is also a customizable workflow.

Customized Workflow: A workflow that has no VP.

Figure 9 shows a customizable workflow with two variation points. *VP_1* has variations v1, v2, etc.

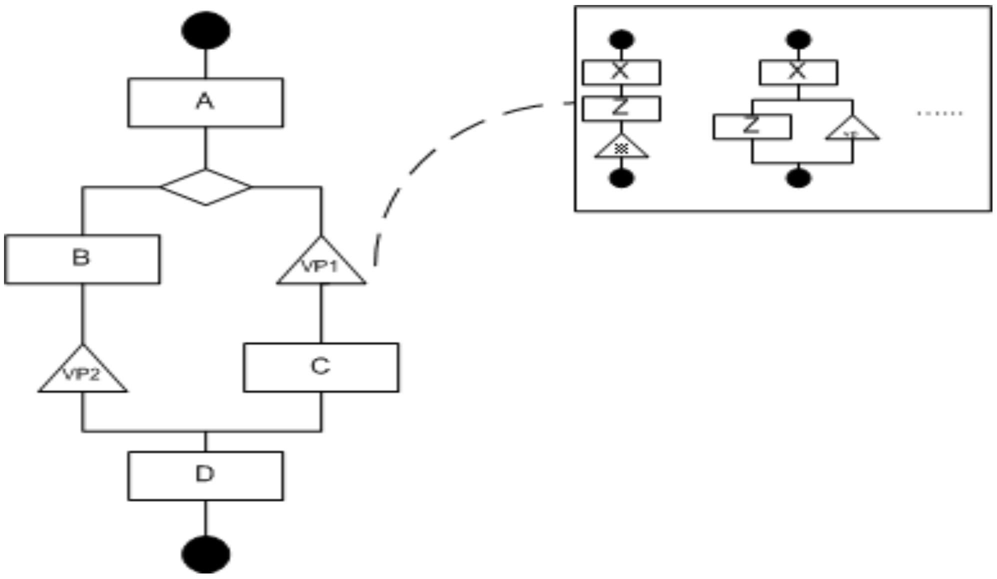


Figure 9 Customizable Workflow

Variation points can have a parent-child relationship defined as the following:

Parent/Child: Suppose a *variation point* (*vp_0*) and one of its variants *v_0*; *v_0* has a *variation point* (*vp_1*), *vp_1* is the child of *vp_0*, and *vp_0*is the parent of *vp_1*.

Level of Variation Points: If a variation point has no parent, then it is a root level variation point whose level is 0. The children of a level *i* variation point are level *i+1* variation points. In Figure 2, vp_1 and vp_2 are both root level variation points; vp_3 and vp_4 are level 1 variation points.

Leaf Variation Point: If a variation point's all variants have no variation point, then this VP is the leaf VP

Depth of Workflow: The level of deepest leaf VP is the depth of the customizable workflow.

2.3.3 Complexity of Customization

Suppose a customizable workflow's level is h, and on average each variation point has x variants, and each variant has y variation points. CPi represents how many customization possibilities for a level i variation point.

$$CP_i = x\cdot(CP_i+1)^{y}$$

If $h = 2$, $x = 3$, $y = 2$, $CP_0 = 2187$. This example shows that there could be thousands of variations even for a fairly simple customizable workflow. For a complex SaaS application, the number of variations could be exponential; therefore, a full manual approach is probably not the most optimal approach, and sometimes it is impossible

2.4 Defining, Organizing and Finding Variants Using Ontology

As discussed in section 2.2, the available assets for SaaS customization are: GUI, Workflow, Service, Data, and QoS. A customizable asset could have one or multiple variation points, and each variation point could have multiple variants. However, what qualifies an asset to be the variant of a certain variation point? How a variation point is defined? To solve these questions, an ontology-based specification is introduced to define variation points, and assist discovery, fuzzy matching.

2.4.1 Data Ontology:

Data ontology is defined first because it is the core of most SaaS applications. Whether it is the data presented in the UI, or data persisted in the storage, or even intermediate data passed through among different components, they all need a systematic way to be described. Data-related customizable assets include: data schema, data encryption, data storage, data load balancing.

2.4.2 Conceptual Data Modeling

For the interested domain, a conceptual data model can be constructed using an ontology system. A conceptual data model is a map of concepts and their relationships. It describes things of significance to the applications. Data modeling is similar to the OO design concept; however, it does not define actions/methods for each entity class. In Conceptual Data Modeling, the following relationships and concepts are defined:

Relationships:

is-a: A is-a B, written **is-a**(A, B), is a relationship where class A is a subclass of another class B. In other words, concept A is a specification of concept B, and concept B is a generalization of concept A [34].

has-a: A has-a B, written **has-a**(A, B), is a relationship where B belongs to A. In simple words, B is a member field of A.

Note that both **is-a** and **has-a** are reflexive and transitive, but not symmetric.

Based on the above two relationship, we define a third relationship.

is-has: is-has(A, B) if either **is-a**(A, B) or **has-a**(A, B). For example, if a service has an input B, providing A is sufficient for this service if **is-has**(A, B), because A has enough information for the service to finish its work.

Atomic data entity: It is the lowest level of details. Atomic data entities are usually numeric data or string literals that have no internal structures. For example, FirstName is an atomic data entity, and its value is string literals.

Composed data entity: A composed entity is a structured data that are built by using "has-a" relationship with atomic data or other composed data. For example, composed data entity Name could be composed by two atomic data entities, FirstName and LastName.

Conceptual data models can be described using OWL [35]. A sample data model is created for HRM is created using protégé, a free, open source OWL editor. In the HRM data model, each data entity is modeled as an OWL class; has-a and is-a relationships are modeled using object properties, and they are *transitive*, *asymmetric* and *irreflexive*. Atomic data entities are also constrained by datatype proterties, which specify the types of data that are allowed.

Figure 10 shows a part of the HRM data model. The brown dotted line indicates an is-a relationship, and yellow dotted lines indicate a has-a relationship.

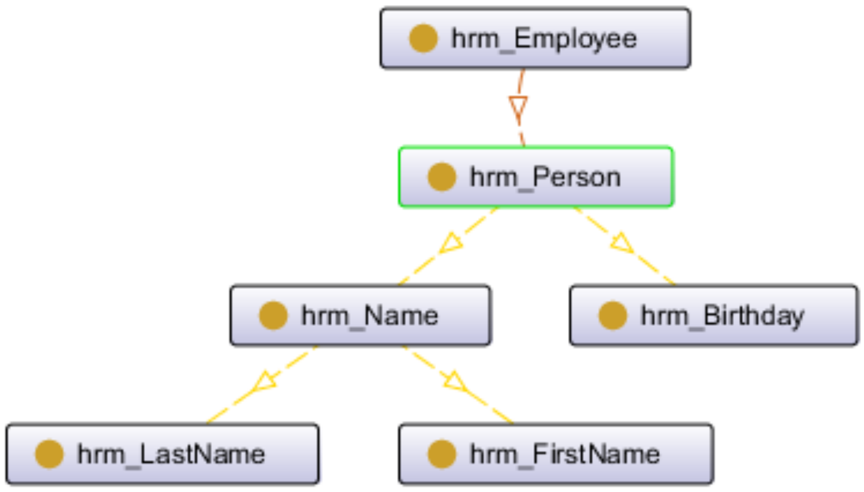


Figure 10 HRM Data Model

Data entities could be used to compose a more complex entity. For example, the Asset entity can be used to compose Customer as shown in the figure, or compose generic data schema Person.

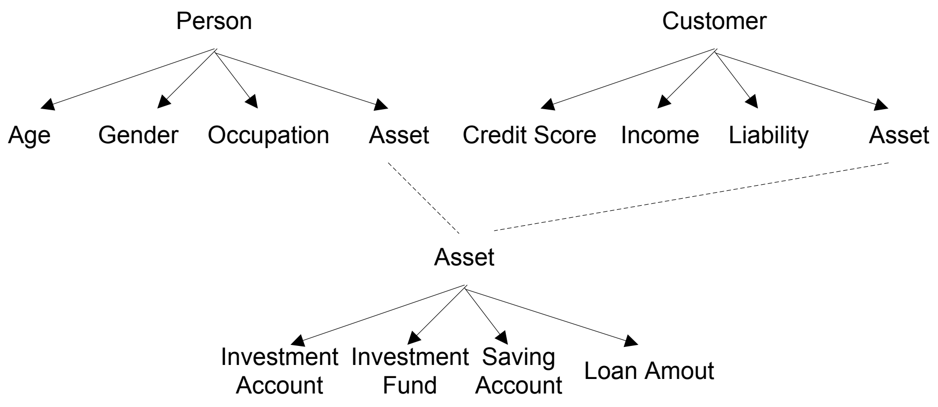


Figure 11 Data Schema Reuse and Composition

2.4.2.1 Data Storage

Storage can also be customizable. Depending on their requirement, tenants can choose different data pattern such as B-Tree, Binary Tree, or Distributed Hash Table. Depending on the sensitivity the data it stores, different encryption algorithms, such as DES, AES, or

SHA, can be applied. Different data serialization and data storage also exit. Table 2 shows what can be customized for data storage, and the available options.

Table 2 Data Storage Customization

Customizable Aspect	Customization Options
Data Pattern	B-Tree, Binary Tree, Distributed Hash Table
Encryption Algorithm	DES, AES, SHA.
Data Serialization	Protocol Buffer, Avro, Thrift
Data Infrastructure	GFS, HFS, DYNAMO,EC2, Azure

2.4.3 GUI Ontology:

2.4.3.1 GUI Component:

To expedite GUI development and customization, SaaS GUI development employs the core idea from SOA [36]: composition. GUI is servicetized into smaller, reusable, composable unit named GUI components, which can be used to compose into full, interactive GUI. GUI components can be classified into atomic GUI components and composed GUI components. Atomic GUI components could be basic user/password forms, a search bar, a navigation menu, a list view, etc. Composed GUI components are GUI composed by several atomic GUI to achieve a more complex function. For example, a fully functional flight search GUI components might include two date selectors, from and to city input forms, etc. To created rich end user experience, multiple technologies, such HTML, HTML5, Flash, Silverlight, components are supported.

Similar to [36], a GUI data specification can be described using the UI Data Profile. Unlike [36], Data profile is modeled in two categories:

1.Data Collection: the raw data collected from the end user.

2.Data Presentation: the data that the UI present to the end users.

2.4.3.2 GUI Interaction Template:

GUI interaction templates often correspond to certain workflows. However, a workflow can be achieved by more than one GUI interactions and a successful GUI interaction sequence is often the result of extensive user experience research and experiments. The difference sometimes could be as small whether to load web page in a new window or refresh the current window. The GUI interaction templates are published and stored in GUI interaction template repository so that the proven successful designs of UI flows can be reused by similar applications in the future.

2.4.4 Service and Workflow Ontology:

According to [36] [32], services are the basic building blocks in SOA. Service developers implement services according to service specification and publish them in the service repository. Application developers use services to construct workflows that meet the business requirements. Workflow templates can be stored in the workflow template repository for future reuse. Service and workflows can be ranked according to quality, price and other various factors. These concepts still remain the same for cloud computing and SaaS development. What distinguishes cloud computing from SOA is the level of customization it needs.

2.4.5 Inputs, Outputs, Preconditions and Effects

Services and workflows all have inputs, outputs, preconditions and effects. OWL-S [37], formerly DAML-S, is an OWL-based web service ontology that be used to describe the properties and capabilities of web services in unambiguous, computer-interpretable form. Specifically, OWL-S could be used to describe the IOPEs of services and workflows.

Combined with conceptual data model, it could be used to define variation points for workflow customization. Many web services do not specify preconditions, nor do they have physical effects on the system, therefore, to simplify the problem, a variation point is specified as an input/output pair.

Let **D** be the set of all the data entities of a conceptual data model for domain **A**. A workflow variation point $VP_{workflow}$ can be defined as following.

$VP_{workflow} = (\boldsymbol{I}, \boldsymbol{O})$, where $\boldsymbol{I} \subseteq \boldsymbol{D}, \boldsymbol{O} \subseteq \boldsymbol{D}$

Similarly, all the possible workflows and services can be defined as following:

$S = (\boldsymbol{I'}, \boldsymbol{O'})$, $\forall i' \in \boldsymbol{I'}, \exists i \in \boldsymbol{I}$, *where* **is-has**$(i, i')$ *and* $\forall o \in \boldsymbol{O}, \exists o' \in \boldsymbol{O'}$, *where* **is-has**$(o', o)$

Basically, the above notations mean that if a variation point has all the input data a service requires, and the service produces more than the variation point asks for output, the this service could be a variant for this variation point.

For example, a variation point is defined as GET_DEGREE = ({Student}, {Degree}). The intention of this variation point is for a given person, return his/her degree if he/she has one. There is a service defined as GET_BACHELOR= ({Person}, {Bachelor}). According to the above definition, GET_BACHELOR could be a variant for GET_DEGREE variation point because is-has(Student, Person) and is-has(Bachelor, Degree).

If the concept of variation points is introduced after a domain has established, meaning there are many existing, but unorganized services and workflows, the above definition can serve as a classifier to find all possible variants for a given variation point. Using the result as a starting point, semantic matching using ontology [38] or manual selection can be used to further reduce variants number for a variation point. It can also help to quickly locate which possible variation point a newly added service belongs to.

2.5 Tenant Mining Based Customization

To better illustrate the ideas and concepts discussed in the chapter, a multi-tenant Human Resource Management (HRM) system is employed as a running example. A HRM system is the strategic and coherent approach to the management of an organization's most valued assets, its employees [39]. A HRM system usually supports features such as employing people, developing their capacities, utilizing, maintaining and compensate their services. Our goal is to design a simplified version of multi-tenant HRM SaaS application, and help the tenants to quickly customize the application according to their unique needs.

2.5.1 Presumption

The customization of SaaS application for a tenant is determined by its requirements; the requirements could be classified into two categories: Tenant-Inherent Requirements and Application-Specific Requirements. Tenant-Inherent Requirements are bound by the tenants' inherent characteristics, and usually are the same across different applications that a tenant might have. For example, company X requires its HRM system to support IE6; it is likely that it requires CRM system to support IE6 too; because its browser requirement is based on the company's current hardware and software stack, which could not be easily changed. Application-Specific Requirements are associated with a specific application that a tenant deploys. For example, the functional requirements for a HRM system are certainly very different from a CRM system.

Though each tenant might have its own inherent characteristic and application specific requirements for a SaaS application, similar tenants might share similar customization needs. The proposed customization approach is built upon this presumption, using data mining techniques to find out what the deciding characteristics and requirements in the terms of customization. Additionally, a large SaaS application might have hundreds even thousands of tenants. Therefore, when a new tenant joins the SaaS application, customization recommendation can be given based on the customization choices that previous tenants had.

2.5.2 Tenant Inherent Characteristics

Tenant's Characteristics can be classified using the ontology shown in **Figure 12**. The characteristics discussed below are not exhaustive, but rather explanatory.

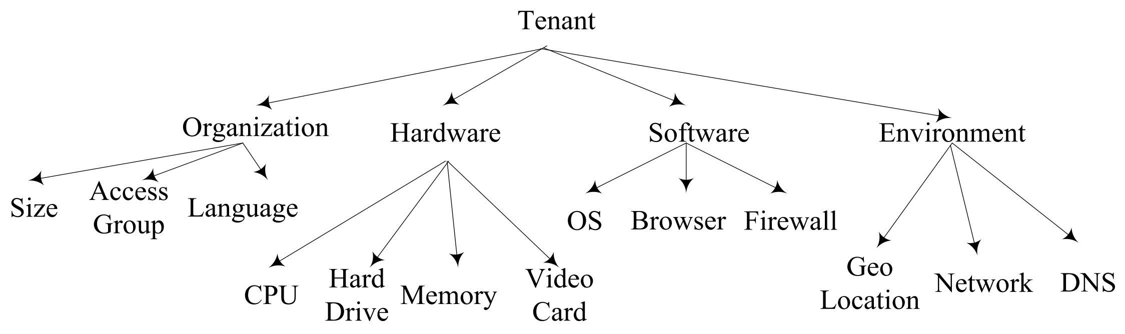


Figure 12 Tenant Characteristics Ontology

Organization Related:

Size: The total number of the users a tenant has or the estimated number of concurrent users. Generally, the size of tenant plays an important role for application's traffic and SLA.

Culture-Related Preference: Tenants from different culture background could have different software requirements. For example, different language options can result in different behaviors of software sometimes. For example, right-to-left written languages such as Arabic cause web page presentation error sometimes; Special characters in certain languages might require certain encoding.

Access Group: Each organization might have different structures. People with different roles might have different access to the system. A multi-tenant SaaS application should be able to handle the different access groups from various tenants.

Hardware Stack:

Tenant's hardware stack could affect both functionalities and SLA of a SaaS application. For example, a graphic intensive application might run faster on a computer that has better video card; an application might utilize the middle button of a mouse, while some mice might not have a middle button. Generally, it is impossible to standardize all hardware running within an organization; however, it is feasible to have all the working hardware to meet a lowest spec. Important hardware factors include CPU, Memory, Hard Diver and Video Card etc.

Software Stack:

Similarly, a tenant's software stack could also affect the functionalities and SLA of a SaaS application. Due to different integrations and implementations of web standards, some web applications might behave differently across from modern web browsers. For example, IE7 has minimal support for CSS3, a SaaS application that utilizes a lot of CSS3 features might appear broken running on IE7. JavaScript engine is another important distinctive feature for web browsers, as a lot of SaaS applications, such as Gmail, Facebook, use extensive JavaScript scripts to enrich client side user experience. A web browser with efficient JavaScript engine might improve a SaaS application greatly.

Moreover, in a corporate environment, certain security software might be installed on all the computers to enforce company-wide security policies. Certain actions, scripts that might violate the security policies might be disabled, blocked by the security software. For example, due to security concern, certain corporate environment might disable flash player. Therefore, a flash based implementation should be chosen for customization.

External Environment:

External environment factors could also affect how a SaaS application runs. For example, most cloud infrastructure providers have data centers in multiple locations. If a SaaS application in deployed in a data center in the US, tenants from the US and Europe might experience different response time, because a request originated from Europe might be routed more time and go through a slower inter-continent cable.

2.5.3 Application-Specific Requirements

Tenants also have application-specific requirements that also affect their customizations. For example, different organizations might have different hiring processes; therefore need to customize the HRM application accordingly. Generally, application-specific requirements are not as easy to capture as tenants' inherent characteristics. It usually requires working with the tenants closely to find out their business needs and translate to software requirements. There are cases that, tenants, such as small business owners, do not really know what to expect until SaaS application providers show them what is available. Sometimes, clients might express the same business needs in different wording, which results seemly very different software requirements. However, as application providers work with more and more tenants, these issues can be overcome. For example, advanced text analysis and ontology can map requirements in different wordings into standardized, unified requirements, which can be stored and reused. Tenant's application-specific requirement can then be described by answering standardized questions, such as "How many rounds of interviews will you have per candidate?", "Who makes the final hiring decisions?". Answers to these questions are either a values in a given range, or choices from a predefined set.

2.5.4 Guided Customization Algorithm

When a substantial amount of tenants joined the SaaS and completed their customizations, their choices can be used as training data to help future tenants make their customization decision. As shown in section 2.3, starting from level 0 *VP*, a tenant makes its decision on which variant to use from lower level *VP* to higher level *VP*, until all leaf *VP*s. The decisions are guided by the mining results from the existing tenants.

Suppose for a SaaS application customization, tenants' characteristics set is $C= \{c_i|i = 1, 2, 3\ldots, n, n \in N\}$

A variation point's variants set is $V= \{v_i| i = 1, 2, 3\ldots,n, n \in N\}$

Therefore, the problem of finding a SaaS application customization for a tenant can be formulated in the following:

For a variation point (vp), *V* is its variant set.

For a variant, *VP* is the set of the variation points it contains.

A customization decision (d) for a *vp* is defined as (*vp*, *v*), meaning, at variation point *vp*, variant *v* is chosen.

A customization decision for a workflow *D* is a set of *d*.

VP_i is the set of level *i* variation point.

The goal of the algorithm: Find a D for a tenant t given an h level customizable workflow whose root variation point is vp_{root}.

Standard data mining algorithms can be applied here to assist a new tenant to make a choice at each variation points. The algorithm can be described as the following.

```
Algorithm:
        $VP_0 = \{ vp_{\text{root}} \}$
        D = ∅
        for (i = 0; i <=h ;i ++)
        begin:
            $VP_{i+1} = \emptyset$
            for each $vp \in VP_i$, and its variant set $V$
                $v$ = run_tenant_mining_decisioning();
                D = D ∪ {(vp, v)}
                VPi+1 = VPi+1 ∪ VP
        End
```

2.5.4.1 Feature Selection

As discussed in section 2.4.2, the number of tenants' characteristics at interest could be huge; some of them could be irrelevant to the decision making, therefore generate noises and degrade the quality of discovered patterns for the data mining algorithm. Feature selection algorithms can be used to reduce the number of relevant characteristics and improve the training data quality. In general, feature selection works by calculating a score for each attribute, then select the attributes with the best scores [40]. Popular feature selection methods include interestingness score, Shannon's Entropy, Bayesian with K2 Prior, etc.

2.5.4.2 Customization Constraints & Suggestions

There could be situations where a decision at one *variation point* might have side-effects to decisions at other *variation point*s. For example, at one subflow, tenant chooses a variant that does not support SSL, while it chooses variants that support SSL for other subflows. However, the application is only as secure as the weakest part, so only one of the choices reflects the tenant's real intent. Similarly, a decision at one variation point might prompt a decision at another variation point. For example, a variant that supports both windows/Linux is chosen for one variation point, most likely, within the same workflow, the tenant will probably choose variants that support both Windows/Linux.

To formally capture the effects the decision made for one variation point might have to the decision making of other variation points; two types of customization rules are defined:

Hard Rules

- Can't be violated
- When violated, the customization decision set is voided, and needs to be modified.

Some of the basic hard rules are defined as below:

Constraints (hard rules):

- (vp, v) → (vp', v')
    - It means, decision (vp, v) entails decision (vp', v').
    - If (vp, v) exists, not only vp' must be in view, but v' should be its chosen variant.
- (vp, v) → − (vp', v')
    - It means, decision (vp, v) must not coexist with (vp', v') (mutually exclusive)
- (vp0, v0) ∧(vp1, v1)→ (vp2, v2)
    - If (vp0, v0) and (vp1, v1) exist, it infers (vp2, v2)
- (vp0, v0) ∨(vp1, v1)→ (vp2, v2)
    - (vp1, v1) → (vp2, v2) and (vp1, v1)→ (vp2, v2)

Soft Rules

- It is more of a guideline.
- It gives tenant suggestions, warnings.
- It does not fail a customization decision set.

Formally, soft rules can be defined as the following.

Suggestions (soft rules) :

- Given d, the probability that d' exists is p. P(d'|d) = p, it can also be written as $d \overset{p}{\rightarrow} d'$, when p = 1, $d \overset{p}{\rightarrow} d'$and → are equivalent.
- Only when p is greater than a threshold, the rule is stored.

2.5.4.3 Rules Deduction

Hard Rules Attributes:

- → is transitive
  d1 → d2 , d2 → d3, then d1 → d3
- → is self-reflexive
  d → d
- → is not symmetric
  d1 → d2, but it might not be true necessarily that d2 → d1

Soft Rules Attributes:

- $\overset{p}{\rightarrow}$ is transitive
  d1 $\overset{p}{\rightarrow}$ d2, and d2 $\overset{q}{\rightarrow}$ d3, then d1 $\overset{pq}{\rightarrow}$ d3
- $\overset{p}{\rightarrow}$ is not self-reflexive and symmetric.

Constraints and suggestions can be based on variation points and mixed with decisions

Rule Operation:

- For a vp, and its variant set V={vi| i = 1, 2, 3…,n, n ∈N}, if ∀ vi ∈V, (vp, vi) → d, the vp → d
- For a vp, and its variant set V={vi| i = 1, 2, 3…,n, n ∈N}, d $\overset{p_i}{\rightarrow}$ (vp, vi), and $\sum_{i=0}^{n} p_i = 1$, the d → VP

2.6 Guided Customization Framework

Based on a discussed definitions and algorithms, a guided customization framework is proposed.

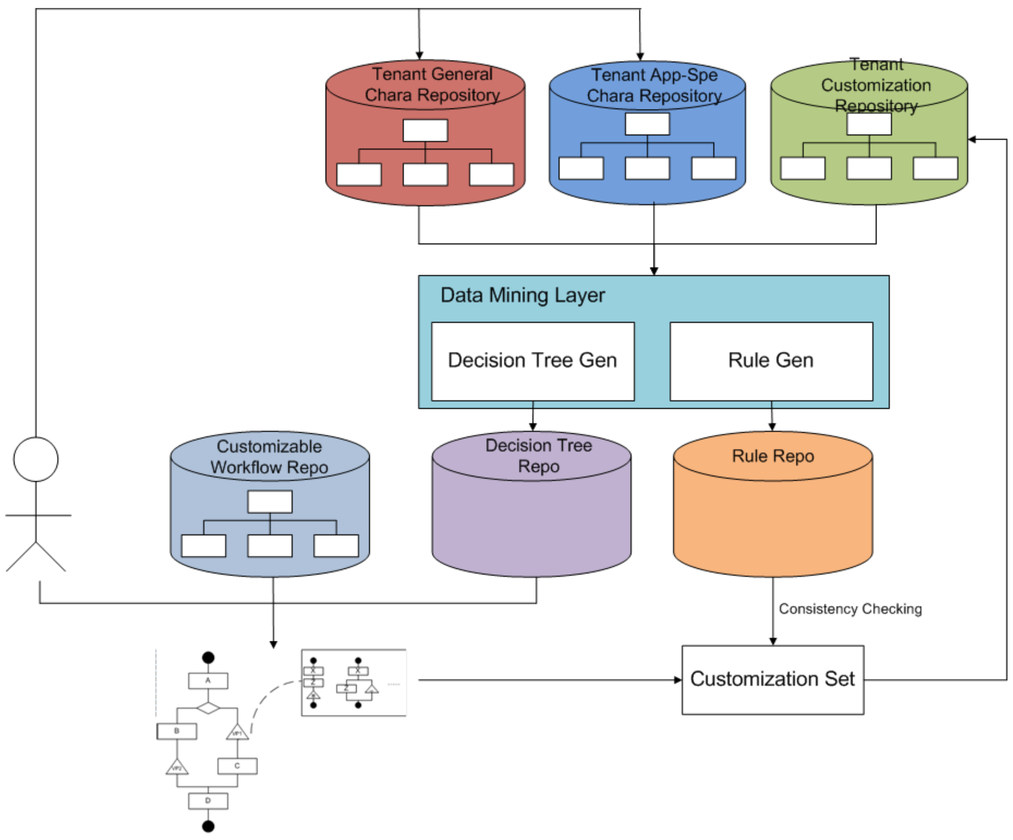

Tenant General Chara Repository
Tenant App-Spe Chara Repository
Tenant Customization Repository
Data Mining Layer
Decision Tree Gen
Rule Gen
Customizable Workflow Repo
Decision Tree Repo
Rule Repo
Consistency Checking
Customization Set

Figure 13 Guided Customization Framework

2.6.1 Guided Customization Framework

- Training Data Repositories: As discussed in section 2.5, the training data are consisted of three repositories:
  - Tenant Inherent Characteristics Repository: It stores each tenant's inherent characteristics, such as number of employees, preferred languages. The inherent characteristics can be categorized into Organization Related, Hardware Stack, Software Stack and External Environment.
  - Tenant Application-Specific Characteristics Repository: Application-specific characteristics are, in another word, application-specific requirements, consisted of functional requirements and QoS requirements.

  - Tenant Customization Repository: Existing tenants' choices at each variation point are their customization decision.
- Data Mining Layer:
  - Decision Recommendation Engine: Based on historical tenant's data and the characteristics of tenant at interest, customization recommendation can be generated by applying algorithm in section 2.5.4.
  - Rule Engine: Traditional Association Rule generation algorithms such as apriori algorithm, eclat algorithm, FP-growth algorithm, can be applied to generate the base rules. The rule set can be further expanded by applying the rule deduction formulas introduced in section 2.5.4.3.

2.6.2 Guided Customization Process

1. When a tenant joins the SaaS, it needs to provide some basic information about its inherent characteristics and its requirements for the application. This could be achieved by having the tenant answer a predesigned questionnaire, or having the SaaS provider send people over to tenant's origination: they can observe tenant's organization, inspect tenant's software and hardware stack and other inherent characteristics, and talk to tenant's stakeholder to gather their requirements for the software, and translated to standardized application specific characteristics. The collected information will be stored into tenant general characteristics repository and tenant app specification repository.

2. The customizable workflow repository contains all workflows that the SaaS supports. Each workflow would contain variation points, which could again contain variation points as explained in previous sections. Based on the historical data, decision trees will

be generated for each variation points. An automated process will help the tenant make the decision at each variation point and return a fully customized workflow.

3. By default, the customized workflows returned will confirm to all the hard rules in constraint rule set. The tenant would examine or test, or experiment with them, then makes necessary modifications. Consistency check will be performed on the modified workflow again against the constraint rule set. If there are hard violations, the workflow needs to be further modified to resolve the violations; if there are only soft violations, they will be brought to tenant's attention, and tenant can make the decision on whether to fix them or not.

4. The finalized workflow will be stored to tenant's customization repository, and decision tree and rule set will be updated using this data.

2.7 Case Study

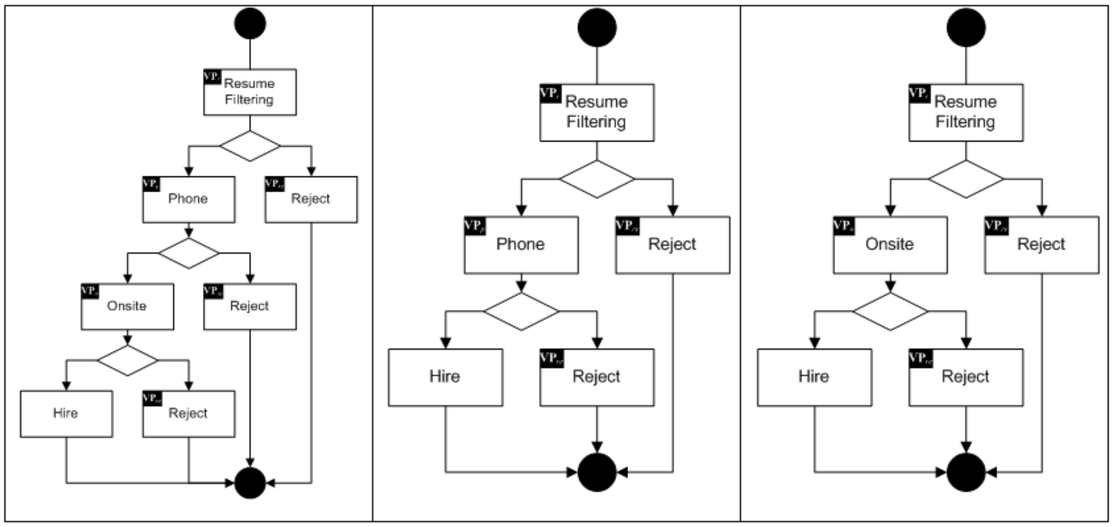
This section will show a simplified but complete example from HRM domain to demonstrate how guided customization using data mining techniques can improve the customization for SaaS application. Section 2.7.1 will give a repository of variation points and all possible variants for the hiring process for an HRM system. Section 2.7.2 shows the initial training data collected by the simulation process and selected examples of the decision trees and constraint sets using the data minng techniques. Section 2.7.4 shows how a guided customization is achieved.

2.7.1 Sample Orthogonal Variability Models

The section gives an example of OVM in section 2.3. The process to model here is the hiring process. Based on each company's specific policies, the hiring process can contain the following activities: resume filtering, phone screening, onsite interview, and regretting/offering processes.

The root variation point is VP_{root}, which has 3 variants shown in the figure.

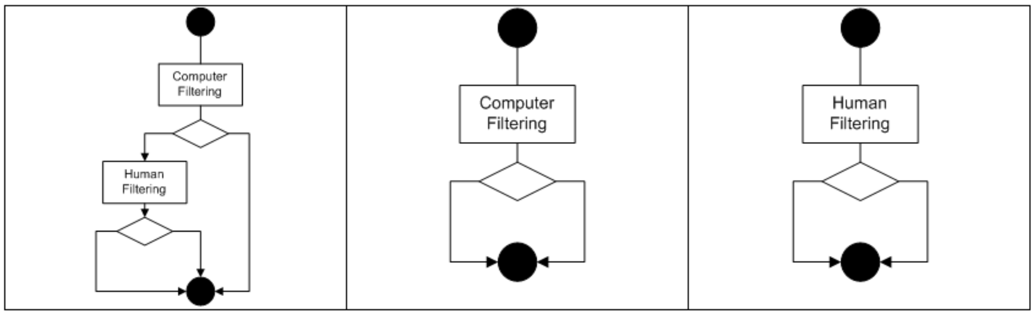
$VP_{root} =$



There are four level 2 variation points: $VP_{resume-filtering}$, VP_{phone}, VP_{onsite}, VP_{reject}.

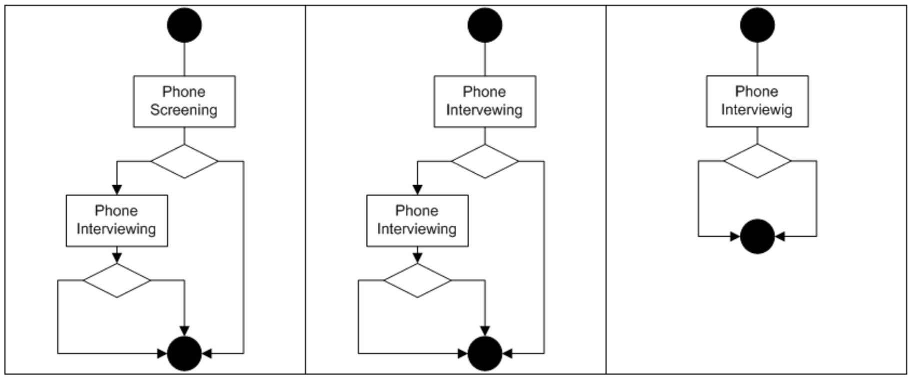
$VP_{resume-filtering}$ has three variants.

$VP_{resume} =$



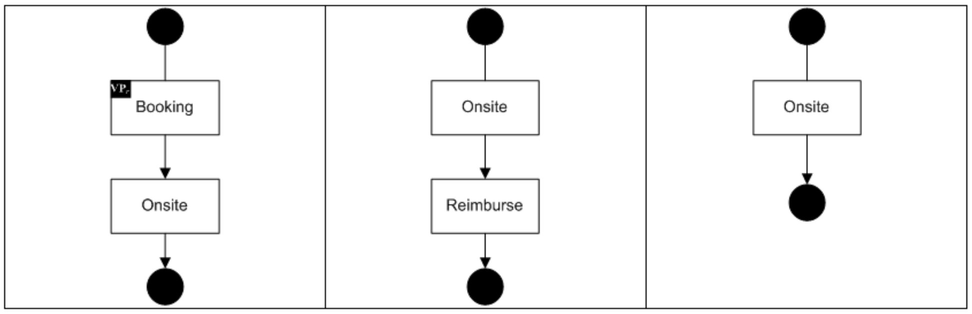
VP_{phone} has three variants: one has one phone interview; one has a phone-screening followed by the phone interviewing; one has two phone interviews.

$\mathbf{VP}_{Phone}$=



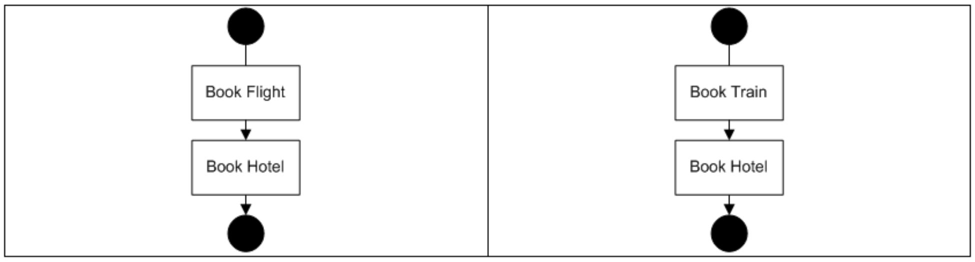
VP_{onsite} has three variants: Some companies will take care of the booking travel and accommodations for remote candidates then bring them onsite; Some might let the candidates do the booking, then reimburse them for the cost; some might just consider local candidates and there is no booking involved at all.

$\mathbf{VP}_{Onsite}$=



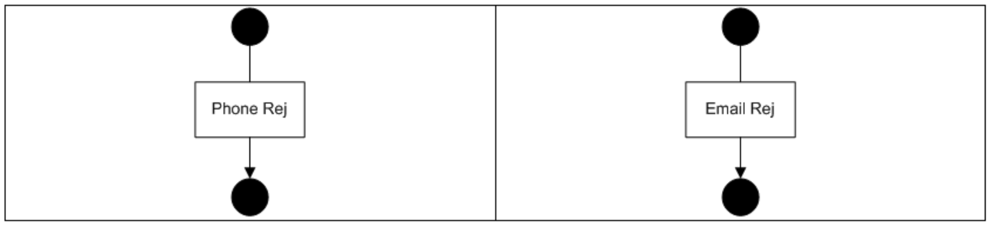
VP_{book} has two variants: One has flight booking followed by the hotel booking; the other has training booking followed by the hotel booking.

$\mathbf{VP}_{Book}$=



VP_{reject} has two variants: One has phone reject; the other has email reject.

$\mathbf{VP}_{Reject}=$



The complexity of the OVM is:

• Number of VPs: 6

• On Average, number of vagrants for each VP: 2.67

• Number of possible customizations: 372

2.7.2 Data Collection

Data collection is completed through simulation. The process can be described as the following:

1.Define a set of tenant inherent characteristics and application characteristics specific for the hiring process for HRM system.

2.Create a set of simulated tenants and assign them with predefined values for their tenant inherent characteristics and application specific characteristics.

3.Create a survey for each tenant on what the customization choices should be made, and distribute the surveys to experienced software engineers and program managers or other capable individuals.

4.For each particular tenant, the most popular customization will be used as the final customization choices.

The inherent characteristics for each tenant should be kept as a profile; once created, it can be shared by multiple applications. The inherent characteristics set should be complete and cover the four categories described in section 2.5.2. However, to simplify the process, only the following inherent characteristics from organization-related category are considered for the simulation:

Organization related = {country, domain, size, years of history, annual revenue}

Application specific characteristics and possible values are defined in Table 3. 10 simulated tenants shown in Table 5 were created for the survey. The surveyed audience was asked how they would customize the HRM system for each tenant with the specific inherent and application characteristics. The most popular results were chosen as the final customization for the tenants. The results are shown in Table 6.

Table 3 Application Specific Characteristics

Name	Description	Possible Values
localOnly	Only consider local candidates or not	Yes, No
phone screening cost	The cost of conducting one phone screening translated into dollars.	0-100$, 100$-200$, 200-500$, 500-1000$
phone interviewing cost	The cost of conducting one phone interview cost	0-100$, 100$-200$, 200-500$, 500-1000$
Onsite interview cost	The cost of conducting one onsite interview excluding candidate's travel cost	0-100$, 100$-200$, 200-500$, 500-1000$
Resume Volume	The expected incoming resume volume per opening	0-10,10-50,50-100, 100-500, 500-1000,1000-

Table 4 Simulated Tenants

Tenants	Country	Domain	Size	History	AR	LO	PSC	PIC	OIC	RV
T1	US	IT	10-50	3	100M	NO	0-100$	100-200$	500-1000$	0-10
T2	US	IT	50-100	10	500M	NO	0-100$	100-200$	500-1000S	100-500
T3	US	IT	5K-10K	20	10B	NO	100-200$	200-500$	> 1000$	> 1000
T4	CH	IT	100-200	5	10M	Yes	0-100$	0-100$	200-500$	500-1K
T5	CH	IT	5K-10K	5	100M	Yes	0-100$	0-100$	200-500$	> 1000
T6	CH	IT	5K-10K	10	500M	No	0-100$	100-200$	100-200$	> 1000
T7	US	Car Sale	10-50	10	5M	Yes	0-100$	200-500$	500-1000$	0-10
T8	US	Car Sale	50-100	20	15M	Yes	0-100$	200-500$	500-1000$	50-100
T9	CH	Car Sale	10-50	5	2M	Yes	0-100$	0-100$	100-200$	200-500
T10	CH	Car Sale	50-100	10	10M	Yes	0-100$	0-100$	100-200$	> 1000

Table 5 Existing Tenants (Training Data)

Tenants	VP_{root}	$VP_{resume\text{-}filtering}$	VP_{phone}	VP_{onsite}	VP_{reject}	$VP_{booking}$
T1	$V_{onsite\text{-}only}$	$V_{human\text{-}only}$	N/A	$V_{booking\text{-}before}$	$V_{phone\text{-}rej}$	$V_{flight\text{-}hotel}$
T2	$V_{phone\text{-}only}$	$V_{computer\text{-}only}$	$V_{2\text{-}phone\text{-}inter}$	N/A	$V_{email\text{-}rej}$	N/A
T3	$V_{phone+onsite}$	$V_{computer+human}$	$V_{1\text{-}phone\text{-}inter}$	V_b	$V_{phone\text{-}rej}$	$V_{flight\text{-}hotel}$
T4	$V_{phone+onsite}$	$V_{computer+human}$	$V_{2\text{-}phone\text{-}inter}$	$V_{no\text{-}booking}$	$V_{phone\text{-}rej}$	N/A
T5	$V_{phone+onsite}$	$V_{computer+human}$	$V_{2\text{-}phone\text{-}inter}$	$V_{no\text{-}booking}$	$V_{phone\text{-}rej}$	N/A
T6	$V_{phone+onsite}$	$V_{computer+human}$	$V_{phone\text{-}screen+inter}$	$V_{reimburse\text{-}after}$	$V_{phone\text{-}rej}$	$V_{train\text{-}hotel}$
T7	$V_{oniste\text{-}only}$	$V_{human\text{-}only}$	N/A	$V_{no\text{-}booking}$	$V_{email\text{-}rej}$	N/A
T8	$V_{phone+onsite}$	$V_{human\text{-}only}$	$V_{2\text{-}phone\text{-}inter}$	$V_{onsite\text{-}only}$	$V_{email\text{-}rej}$	N/A
T9	$V_{phone+onsite}$	$V_{computer\text{-}only}$	$V_{2\text{-}phone\text{-}inter}$	$V_{onsite\text{-}only}$	$V_{email\text{-}rej}$	N/A
T10	$V_{phone+onsite}$	$V_{computer\text{-}only}$	$V_{2\text{-}phone\text{-}inter}$	$V_{onsite\text{-}only}$	$V_{email\text{-}rej}$	N/A

Table 6 New Tenants

Tenants	Country	Domain	Size	History	AR	LO	PSC	PIC	OIC	RV
T11	CH	IT	10-50	5	5M	NO	0-100$	100-200$	500-1000$	200-500
T12	US	IT	100-500	10	15M	NO	0-100$	100-200$	500-1000$	200-500

A decision tree can then be constructed accordingly at each variation point. For example, $VP_{resume\text{-}filtering}$ has the following decision tree:

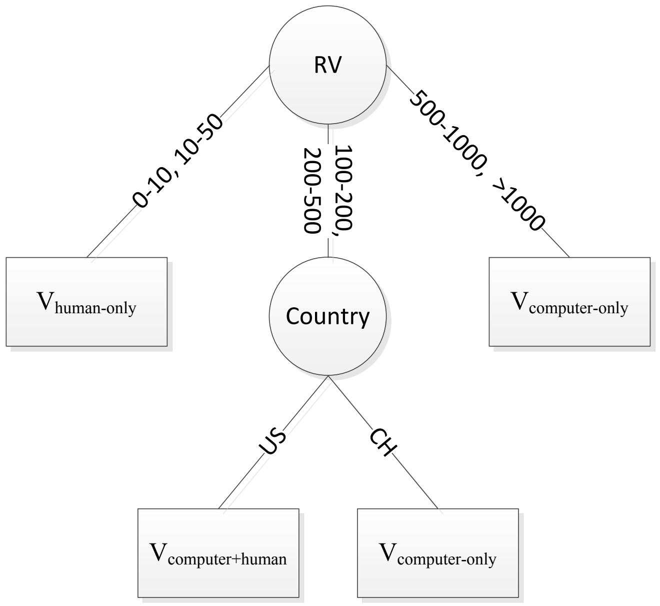


Figure 14 Decision Tree Example

Suppose new tenants T11, T12 with the characteristics described in Table 6 is trying to join the SaaS. Based on the decision tree, at variation point $VP_{resume\text{-}filtering}$, T11 should choose $V_{computer\text{-}only}$, while T12 should choose $V_{computer+human}$. Because a decision tree can be created for each variation points, one can create a customization for each new tenant.

Constraints and suggestions can also be mined from the training data set. For example, whenever $V_{booking\text{-}before}$ is choosen, $V_{flight\text{-}hotel}$ is also choosen. Therefore, a hard rule $(VP_{onsite}, V_{booking\text{-}before}) \rightarrow (VP_{booking}, V_{flight\text{-}hotel})$.

2.8 Conclusion

This chapter discussed the different aspects of tenant-level customization for SaaS applications, and then proposed to use OVM to model it. A tenant-based guided customization framework was proposed to reduce the typical manual work involved. This framework mines relationships between tenants customization decisions and tenants inherent characteristics, their application-specific requirements and uses this knowledge to automate the customization for the future tenants. A case study using HRM system was shown to demonstrate how guided framework works.

3 SELF-ADAPTIVE CUSTOMIZATION FOR MOBILE APPLICATIONS

The proliferation of mobile devices has transformed us into an app-driven society in recent years. Mobile apps have evolved from basic calculators to high-fidelity 3D games. People find themselves complete more tasks on their mobile devices than on their computers. Native mobile apps are preferred by the consumers due to their better performance, fluid UI transitions, etc, however, once a native mobile app is downloaded and installed on a device, it is required another update from the store, which often results in user loss. Mobile user base is large and diverse, and users have different behaviors. One version of an application might not be best suited for all users. In this chapter, we propose an architecture that can adapt to these variants and customize an application that fits each group of users.

3.1 Introduction

According to a study from Nielsen in July 2014, U.S Android and iPhone users age 18 and over spend over 30 hours each month using mobile apps, a 65 percent increase compared just two years ago [41]. However, mobile app developers face challenges that desktop and web applications developers didn't have to.

1. There are two prominent mobile platforms. While Android is leading iOS in terms of market share, iOS continues to be the more profitable platform, where often new ideas, new apps are first released. For a globally successful mobile application, ignoring either of the platforms will lose a significant amount of users automatically.
2. Mobile devices have different capabilities [42]. Mobile devices have different models, manufactured by different vendors, targeted at different price ranges,

support a various range of sensors. Using the most advanced sensors might result in the app providing a less satisfying experience for devices with less capability.

3. Mobile devices are mobile. The users are often on the go when they use the mobile devices. The network connectivity can go from being excellent to being spotty or none in one user session. The GPS signal strength can also change greatly when a user goes indoor from outdoor.
4. Mobile devices have limit battery [43]. Even with huge advancements in battery technology, power is still a limited resource. Mobile apps should use power conservatively, and react and adapt when the battery level drops to a certain level.
5. Mobile data is usually metered which means mobile users are very data-conscious [44]. Using network excessively might anguish users with very limited data quota, on the contrary, being too thrift when the users have abundant or unlimited data might result in suboptimal experience.
6. More and more mobile apps need to support multi-tenancy [45]. The same app might be deployed to different enterprises, organizations. It needs to be branded properly, customized for each tenant's specific requirements.

In addition, the number of mobile users grew exponentially in recent years. With the huge user base comes with great variants of user behaviors. Take the news reading app for example, some users might only open the app in the morning, while some users might choose to do that in the evening; Some users might only use the app briefly and sporadically through the day, while some users only use it once or twice a day but with long sessions.

Clearly, one size doesn't fit all. To provide the best experience to all the users while being aware of and considerate to the limits and constraints of their devices, and their individual habits, a mobile app needs to be more customizable and adaptive. In this chapter we will discuss a self-adaptive mobile apps customization, and the approaches could be used to test these apps. In 3.2, related work is presented. In section 3.3, a self-adaptive mobile customization architecture is proposed and illustrated; In section 3.4, the analytical process is discussed in details; In section 3.5, we will demonstrate a metric-driven A/B testing framework to iteratively customize mobile apps.

3.2 Related Work

In [46], Tsai and his colleagues established the SaaS-based BIB performance and benchmark architecture and proposed the SaaS-based BIB Database Model (SaaS-BIB-DM), the architecture layer (SaaS-BIB-AL), the data flow view (SaaS-BIB-DF) and the representative transaction model (SaaS-BIB-TM). The architecture can be applied directly to mobile BIB applications. To provide customization support, Yvette E. Gelogo and Haeng-Kon Kim proposed that developing mobile ERP application using Adaptive Object Model (AOM). An Adaptive Object Model (AOM) is a common architectural style for systems in which classes, attributes, relationships and behaviors of applications are represented as metadata, allowing them to be changed at runtime not only by programmers, but also by end users. In [47], Chen et al proposed a process customization framework that can customize and configure processes while guarantee the tenancy isolation. Multiple versions of processes might be generated based on one basic template process after the customization. The collection of customized processes is necessary for ISVs because customized processes reflect demanding information of tenants. In [48], Ren et al recognized the importance of preservation for tenancy history metadata. They proposed a method for adjusting template objects dynamically based on XML structured features for tenancy metadata that improves the convenience of on-demand customization and user experience, shortens the tenants' customization time and improves QoS. In the event of tenants requirement change, the template can update accordingly by analyzing tenancy history metadata from Graphic User Interface (GUI), workflow, service, and data layer. However, all the customization processes mentioned above did not take mobile applications unique characterstics into consideration. Their approaches also did not consider similar tenant share similar requirements and each individual's behaviors of each tenant. There is also no way to evaluate how a customization is successful or not.

3.3 Customizable Mobile Applications Architecture

To successfully support customization for mobile applications, the architecture of mobile client and server have been specifically design. This section proposes the customizable architectures for mobile clients and servers, and compares them to the traditional mobile client and server architectures.

3.3.1 Traiditional Mobile Application Client-Side Architecture

Figure 15 shows the traditional client side architecture for mobile applications, which has following layers and components:

Presentation Layer: There is where users directly intereact with the mobile applications. The information and data are rendered by typical mobile views, such as labels, lists, carousels, hyperlinks, buttons, etc. The UI are generally touch based, accepting user inputs through clicks, scrolls, swipes, etc. Developers can create custom views and view groups to build more advanced UI rendering and interactions.

Business Layer: Traditional UI based applications generally follow the Model-View-Controller (MVC) pattern [49]. The Business layer contains both the models and the controllers. The controllers, which contain the business logic generally in the format of workflows, transform, decorate, aggregate the data fetched from the data layer so that the presentation layer can consume them. Workflows are composed by components, which are reusable, shareable.

Data Layer: The data layer abstracts the common data operations, and exposes them to business layer for data fetching, data deletion and data manipulation. To improve the perceived speed to the end users and ensure the responsiveness of mobile applications, in-memory cache and disk cache can be used. The data layer is also responsible for unmarshaling the data received from network, and marshaling the data models used in the applications to the wire transfer format used in by the network.

Network: The network component implements a simple network protocol, usually http or https. The layer is fundamental as its communication channel between the mobile client and server. It downloads data from the server, and when sends data back to the server.

Storage: Storage layer is used to persist user data on the device. Generally, the user data is also saved on the server side for a network-based mobile application, and the server maintains the consistency as well. The client storage serves as a local storage to improve the perceived speed. Typical on-device storages are default phone storage, add-on mobile sd cards. The size of mobile storages ranges from a few gigabytes to hundreds of gigabytes nowadays.

User Settings: User Settings, built on top of the on-device storage, generally save the preferences of a user for using the app. Typical settings include push notification settings, network setting, tab layouts, language setting, etc. These settings can be perceived as simple customization done by the users.

Event Handlers: Touch-based mobile applications respond to events generated by the users, servers, and devices. Typical events include user touches, receiving a push notification, location changes through geo-fencing, speed/direction changes detected by highly advanced sensors, etc. These events can trigger an application workflow even the application is not running when the events happened.

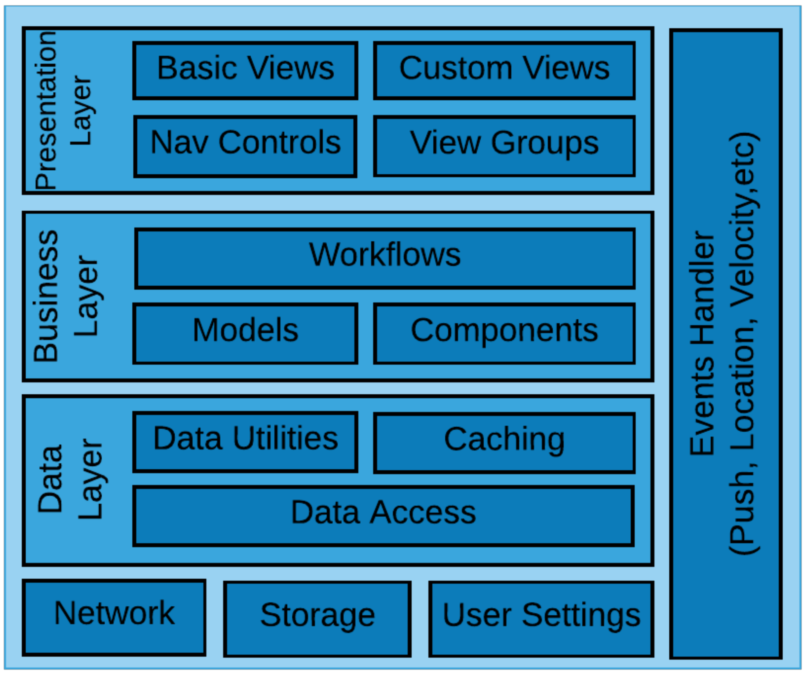


Figure 15 Traditional Client Side Architecture

As one can see, the existing mobile architecture supports building a functional, usable mobile application, however, it has the following drawbacks:

1. There is no way to understand how the users are using the app. Are users heavily engaged with apps? Do they use the app on a daily basis?

2. There is no way to understand the device condition when users are using the app. Whats the battery level? Did the user turn off the location service? How much available memory left?

3. Once the app is installed, and if users don’t turn on the auto-update feature provided by the mobile platforms, it is generally very difficult to change the behavior of a mobile application.

3.3.2 Customizable Mobile Application Client-Side Architecture

To solve these issues, new mobile client architecture is proposed to support customizations for mobile applications. Figure 16 shows the customizable client side architecture with the newly added component highlighted in yellow color.

Tracking: This component collects the data on how a user uses app by recording all users activities on the app, such as clicks, scrolling, swiping, etc, then batch these data and send them back to the server. It measures all performance potential bottlenecks, such as network round trip, image processing, storage/memory size, etc. Each tracking request is also sent with meta data, such as device manufacture, device model, OS version, app’s version, location, network type, etc. These data will be stored and analyzed on the server, which in the end will help provider the appropriate customization for a user.

Monitoring: The monitoring component collects all the device information, such as used memory, available memory, CPU utilization rate, battery level, location setting, push notification setting, etc. The information will be sent as meta data through tracking events sent by the tracking component.

Logging: In addition to sending all the tracking events to the server, these events, along with other general logging, such as debug logs, error logs should also be logged onto the device storage for troubleshooting purpose. In the case, where a user encounters an exception, a crash, or an error, these logs on the devices can be sent to developers to help them diagnose the problem.

Template Loader: Template loader is a component that queries the server if there are alternative implementations for a variation point. If there exists one, it will download it, then unpack it into the format that can be executed by the template executor.

Template Executor: For each variation point, it can intelligently select which implementation to load for a user, and then execute it based on the customization information sent back by the server.

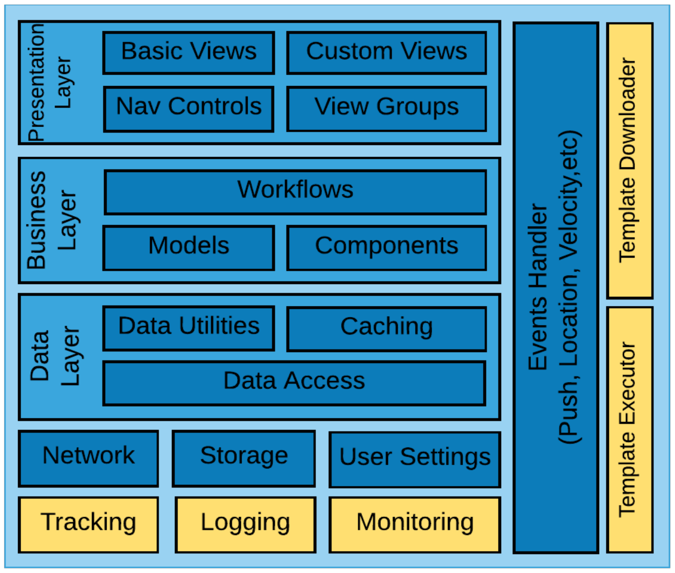


Figure 16 Customizable Client Side Architecture

3.3.3 Dynamic Code Execution

This process is possible for both iOS and Android.

iOS: JSPatch is an open source library on iOS that bridges Objective-C and JavaScript using the Objective-C runtime. Once included, the app can call any Objective-C class and method in JavaScript, which makes the APP obtaining the power of script language: add modules or replacing Objective-C code to fix bugs and add features dynamically.

Android: Grab'n Run is an open source library on Android that can securely load code dynamically into your Android application from APK containers or JAR libraries translated to be executable by both the Dalvik Virtual Machine (DVM) and the Android Runtime (ART). This gives an android application the freedom to potentially change everything after the app is installed. Figure 17 explains this process in a little more details.

1. The Android app, specifically the template loader will contact the remote server to ask if there exists a template for a certain variation point. If the server responds yes, the template loader will fetch the "template.jar" from a remote URL.
2. After the template.jar is downloaded, the template loader will store it on the phone storage.
3. The template loader will then prepare a direction for the optimized dex files
4. The template executor will dynamically load and execute the template once the variation point is encountered in the workflow.

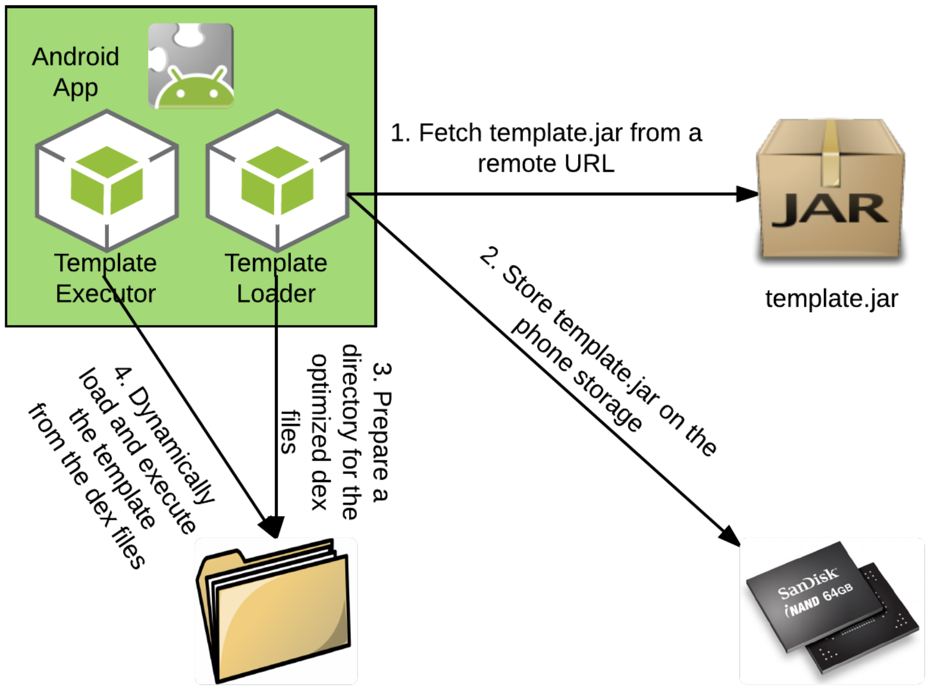

Android App
Template Executor
Template Loader
1. Fetch template.jar from a remote URL
template.jar
JAR
2. Store template.jar on the phone storage
3. Prepare a directory for the optimized dex files
4. Dynamically load and execute the template from the dex files
SanDisk iNAND 64GB

Figure 17 Dynamic Code Execution

3.3.4 Traditional Service Side Architecture

Figure 18 shows the traditional server side architecture for mobile applications, which has following layers and components:

Front End Service: Front end services typically serve the mobile clients with the data that need to be rendered to the end users. Front end services usually fetch raw data from multiple mid tier services, and aggregate and massage the data into a format that can be easily consumed by the mobile clients. Some front end service examples are:

1. Services that return profile data with basic profile information, enducation background, job experience, etc.
2. Services that return a stream of new feeds
3. Services that authenticate and authorize a user to a member based service application

Mid Tier Service: The mid-tier services usually act as a bridge between data storage and front end services. It usually includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The mid-tier layer should provide an API to the front-end services that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms.

Online Storage: The data storage that supports online, real time usage. It requires high throughput with low latency, with multi-threading support. Typical online storages include RMDB, such as MySQL, Oracle, or NoSql database, such as MongoDb, CouchDB.

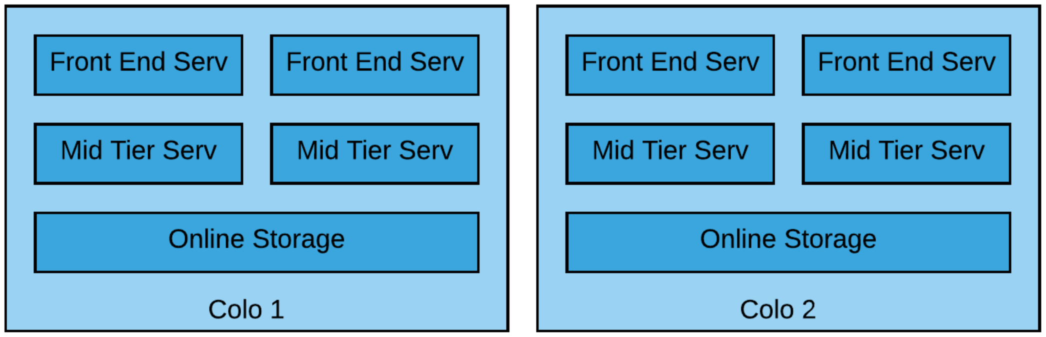


Figure 18 Traditional Server Side Architecture

Similarly to traditional client side architecture, traditional server side architecture has the following drawbacks:

1. There is no support to track user behavior, and device states.
2. There is no support to return different users with different dynamic template therefore provide personalized customization for the users.

3. There is no analytical support to understand the user behaviors based on the tracking data, and there is no support to understand how the performance of applications.

3.3.5 Customizable Service Side Architecture

To solve these issues, customizable server side architecture is proposed. Figure 19 shows customizable server side architecture with the newly added components in yellow color.

Authentication: This component is responsible to authenticate a user. If a user doesn't have an account, it will first onboard the user by taking him/her through the registration process.

Tracking: Tracking services receive the tracking event sent by the mobile clients, and persist them in the offline storage. Because of the huge volume of the tracking data, and lower requirement for transactional accuracy, the persistence of the tracking data is usually achieved in a fire-and-forget manner.

Analytical: A/B Testing is a more explicit approach to help the app developers to find out which approach/UED (User Experience Design) would achieve a set goal, for example, how to make the users to share more feed. To isolate the influence of other factors, A/B testing often targets at a user group with similar attributes, for example, users share less 10 jokes in a month. The goal is to find out which one of pre-design approach can achieve better result. These more targeted learning would affect the customization significantly.

Offline Storage: Offline storage needs to support a huge amount of data, typically over hudreds of thousands of petabytes. The data is stored in a distributed manner. There is no requirement to access the data in real time. The data stored on offline storage is primarily for analytical usage.

Machine Learning: The data collected by the tracking component can also be used for machine learning to improve user experience. For example, using machine learning algorithm can improve the relevancy of the content delivered to the users, thus enhances user engagement. It can also be used to generate new customized templates that can be loaded by the Template Loader from the client, and later used for dynamic code execution.

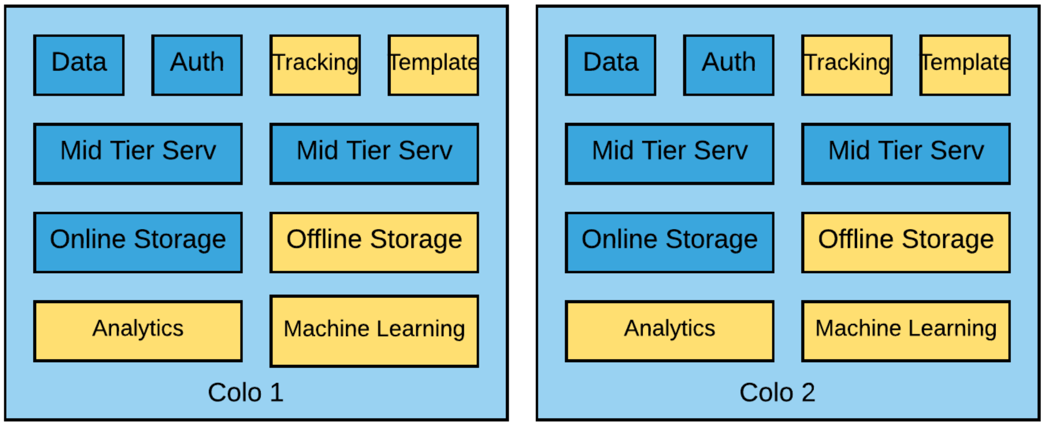


Figure 19 Customizable Server Side Architecture

3.4 Analytical Process

The data collected by the tracking component is huge, and have many dimensions. Moreover, to support analysis of user behaviors, the metrics are often needed to sliced or aggregated to support SQL-like queries. Figure 20 shows full analytical process from data collection to query serving.

1. Data Collection: Everytime a user performs an action of the interest of tracking, a user event is generated. The event is annotated with the auxiliary information for that

particular action. For example, if a user clicks on an ariticle link, the id of that article is included in the user event. In addition, basic device information is also included, such memory usage, CPU utilization, location, etc. To prevent from interfercing user's normal activities in the app, these events are batched on the client side, are only sent to the server when app is backgrounded or the user has been idle for than a certain threshold, for example, 30 seconds.

When the server receives the user events sent by the tracking component of the clients, it puts them in a message queue, such a Kakfa [50]. The messge queue acts as a broker, and broadcasts the received messages. Consumers subscribed to these messages will be notified once new messages are received. There are two primiary consumers of these user events, offline data processing and real-time data processing

2. Offline Data Processing

User tracking events are ETLed and stored in offline storage, such as Hadoop [51]. These data are then run through customized Hadoop job to restore needed dimensions to support the required queries. For example, to answer the questions, what type of news a particular user is most interested in, we need know all the news a user opens, and their types. The user event for opening a news link only contains the id for that particular news, but not the meta data, the hadoop job will annotate the event with meta data of the news. The data will be saved in Online Analytical Processing (OLAP), such as Pinot [52]. However, offline processing has some delays, ranging from a few hours to a couple of days; to get the near real time support, online data processing becomes necessary.

4. Online Data Processing

For the online data processing flow, user events are fed into a stream proessing component, such as Samza [53]. Samza has a callback-based "process message" API comparable to MapReduce, which can be used in a way similar to the offline data processing pipeline. For example, the new click event will be joined with the meta data of the news, and the joined event will be sent another message queue for online query.

4. Online Query

OLAP, such as Pinot, has two kinds of tables, offline and realtime. An offline table stores data that has been pushed from Hadoop, while a realtime sources data from Kafka. The offline and realtime tables are disjoints and may contain data of the same period. When the OLAP receives a query, it is smart to enough to fan out the query to both the online and offline tables, and join the results from both tables.

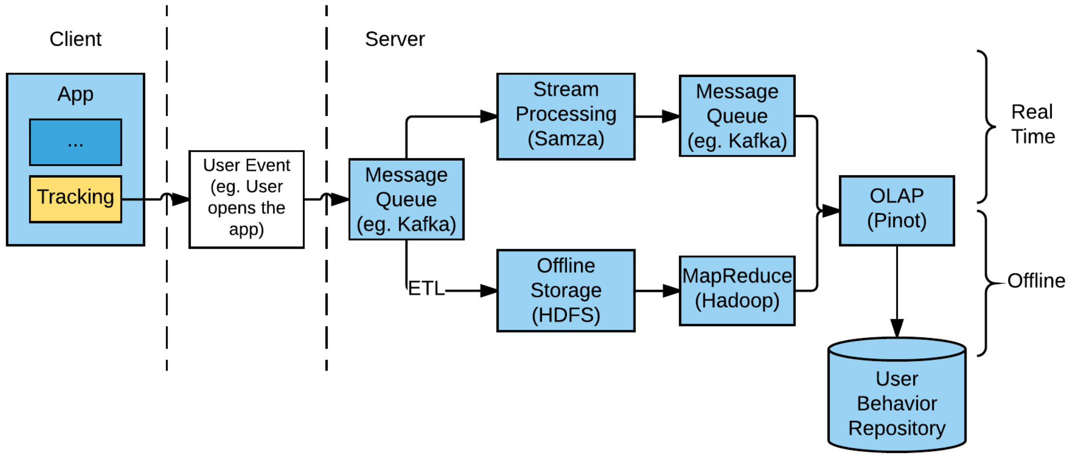


Figure 20 Analytical Process

3.5 Metrics-Driven A/B Testing Customization

As discussed in earlier section, different users have different behaviors, habits and devices, etc; therefore, having same app for everyone will result in sub-optimal results for many people. In this section, a metrics-driven A/B testing customization process is proposed.

3.5.1 News Reading App

To better explain this process, we focus on a common network based app as a running example. In general, a network based app usually fetches some data from server, and presents them to the users, and the users can perform some actions which in return causes the data to change/update from the server. To make it more concrete, we use a simple social news reading app as a running example. The app has the following features:

1. It allows a user to register an account and create a profile associated with it.
2. It allows a user to browse all news or news of a certain category, or news published by certain publishers.
3. A user can like, comment and share a news.
4. The app can send users push notifications when breaking news happen, or a summary of the important news of the day.

3.5.2 Metrics-Driven A/B Testing Customization Process

Metric-driven A/B tests start with the following

1. Identify top metrics

Having clear-defined, easy-to-understand metrics can set attainable goals, and measure improvements over iterations. For example, for a news reading app, one of the most important metrics is to measture how much time people spend on the app to read news. Any UI changes, be it as small as color change, font change, or as big as UI navigation change, if it can improve this metrics, it should be considered successful.

2. Identify variation points

As discussed in Chapter 2, variation points can be used to identify places that can be changed or customized in an app. As the client side architecture shows, UI, business layer, data layer, network, storage, etc can all be customized.

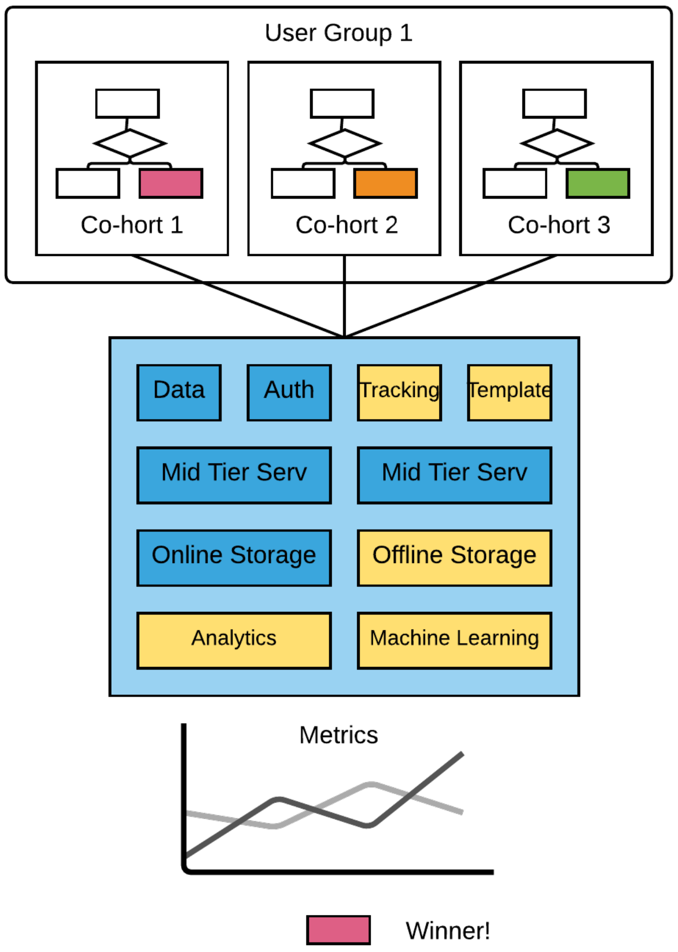


Figure 21 Metric Driven A/B Tests Customization

UI Customization: Some of the customizations can be exetremely easy, such as theme color, font, font size, tabs order. These easy customizations might not be implemented through dynamic code replacement and execution, but an easy config change. Some UI customizations need designers' assistance and enginees to implement, such as UI layout changes, navigation changes, design language changes.

Business Layer Customization: Workflows, models and components are the interests of customization in this layer. For example, when a new user first signs up, he/she has to go through an onboarding process. In this onboarding process, the user will be asked what type of news he/she is interested in, which publishers that he/she is interested to subscribe and what writers that he/she interested to follow. The order of this onboarding flow can b shuffed, and each component can be tweaked to provide better results.

Network Customization: A metered mobile user, data consumption is always on top of this mind. A network library can be customized to not download the images of an article until the user specificially asks it to. Or when a user on unlimited data plan or on wifi, the network can behave a little more aggressively by downloading more artcles in advance so when the user reads them, the response time would be exetremely low. Or just a different implementation of the protocol, because of the differnet chipsets on the different devices, the implementation on one type of devices might perform better than the rest, but it might not be true for another type of devices.

3. Run the Tests and Monitor the Metrics

As illustrated in Figure 21, for a particular variation point, we identify the user co-hort group with similar to identical traits. For example, if we want to test which network implementation is the best for users with slow network, all the users that are often on a slow network would become the test targets. If there are N variations points, we devide the user group into N co-hort, and run each of the variation on one of the co-hort. Through out the testing period, the tracking components continue to send tracking data back to server, run through the analytical process and generate the metrics that we monitor. In this case, time spent in the app is the metric of the interest.

After some time, the data will become statistically significant, then we can answer the question which variant has the best performance. We would then apply this variant to all the co-horts, thus conclude the experiment.

3.6 Conclusion

In this chapter we discussed a self-adaptive mobile apps customization framework. We achieved this by extending and improving both client side and server side architecture. We illustrated a dynamic code execution machenism that allows the behaviors of an app to be changed after it has been installed without reinstallation. An analytical process is discussed to transform the user tracking data into metrics that define success of each customization. Finally, a metric driven A/B testing cuatomization process is proposed to help adapt different users behaviors, habits and device capabilities.

4 TWO-TIER MULTI-TENANCY CCALING AND LOAD BALANCING

Cloud computing often uses the multi-tenancy architecture where tenants share system software. To support dynamically increasing demands from multi-tenants, the cloud service providers have to duplicate computing resources to cope with the fluctuation of requests from tenants. This is currently handled by virtualization and duplication at the application level in the existing cloud environment, such as Google App Engine. However, duplicating at the application level may result in significant resource waste as the entire application is duplicated. This chapter proposes a two-tier SaaS scaling and scheduling architecture that works at both service and application levels to save resources, and the key idea is to increase the resources to those bottleneck components only. Several duplication strategies are proposed, including lazy duplication and pro-active duplication to achieve better system performance. Additionally, a resource allocation algorithm is proposed in a clustered cloud environment. The experiment results showed that the proposed algorithms can achieve a better resource utilization rate.

4.1 Introduction

Cloud Computing has emerged as a new infrastructure that enables rapid delivery of computing resources as a utility in a dynamically scalable, virtualized manner. To take advantages of the scale of economics, many service and application providers start offering their software-as-a-service (SaaS) over the clouds. Typically, SaaS applications allow multiple tenants to reside on single or multiple instances of the software at the same time, which helps brings down the licensing fee. Virtualization is widely used in current Cloud computing systems [54] [55], which allows the ability to run multiple operating systems on a single physical system or one operating system on multiple physical systems as shown in

Figure 22.

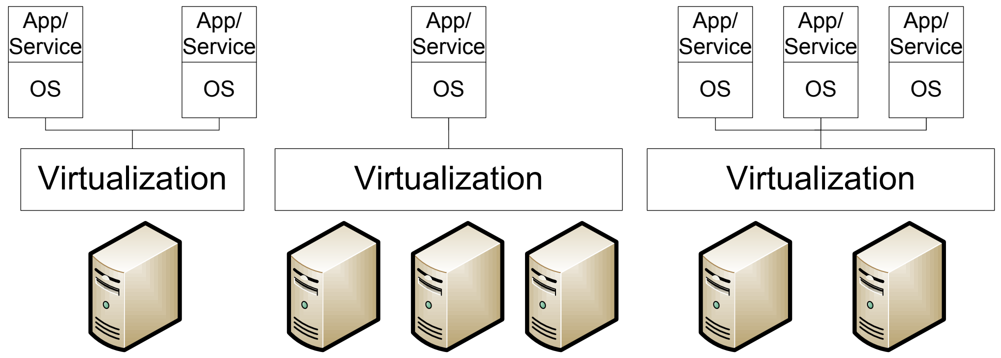


Figure 22 Virtualization in Cloud

To cope with dynamically increasing demands from multiple tenants, Cloud service providers need to allocate computing resources for the application dynamically. Existing solutions often involve application instances duplication. For example, for a Google App Engine (GAE) application, as the traffic to the application increases, more instances of the application are duplicated and deployed on more servers to handle the increased workload. However, most applications are componentized as software services, and computing load might not be distributed evenly among application components. Therefore, duplicating at the application level may result in computing resource waste, as it is usually only the bottleneck component that needs more computing resource.

Another challenge is that different server nodes (virtualized or not) can have diverse levels of computing power. Given a resource request, it is necessary to choose the most suitable server nodes to run application/service duplicates.

This chapter proposes a two-tier SaaS scaling and scheduling architecture that duplicates at both service and application levels along with a resource allocation algorithm that takes different computing power of server nodes into consideration. The main contributions of this chapter are as follows:

A two-tier SaaS scaling and scheduling architecture at both service and application levels;

A greedy resource allocation algorithm that selects suitable server nodes to run application/service duplicates.

Two duplication time strategies, lazy and pro-active are provided to be chosen according to application requirement and experiment evaluation that shows the effectiveness of the proposed model.

The chapter is organized as the following: Section 4.2 discusses server level and application level duplication and proposes a two-tier SaaS scaling and scheduling architecture; in section 4.3, a clustered-based resource allocation algorithm is proposed; section 4.4 discusses two different duplication time strategies and how to choose between them; section 4.5 shows some simulation results to show the effectiveness of the resource allocation algorithm; section 4.6 concludes the chapter.

4.2 Two-Tier SaaS Scaling and Scheduling Architecture

In this section, a case study is used to further illustrate why duplication at service level is necessary. Several definitions are first introduced for purpose of discussion.

Application Request (R) is a request sent by the end users to the application. For a service-oriented application, several intermediate requests to the composing services might be generated to process an application request. In Figure 2, intermediate requests, R_A, R_B, R_C might be generated for service A, B, C for a R_{APP}.

Component Throughput (Thr) is the maximal number of requests a component can process in a second without violating the SLA.

Overloaded/Underloaded Component: If more requests are sent to the component per second than the component throughput, the component is overloaded. If the average number of requests processed by the component per second is smaller than component throughput, the component is underloaded.

The overloaded components can affect the system's performance and result in SLA violation. The underloaded components might bring down resource utilization rate. For example, suppose one has a service-oriented application as shown in Figure 23. The application is composed by three atomic services: A, B, C. The number in the brackets indicates throughput of each component. The SLA for this application requires that a request to the application needs to be returned within 2 seconds. At a time snapshot, 15 requests are sent the application per second. Considering the following two cases of service requests of B and C:

<Case 1>: Twice as many intermediate requests are generated for B as C: $R_B = 2\ R_C$.

After A, 10 R_B are sent to B, 5 R_C are sent to C, which are the throughputs for B and C accordingly, therefore, the application can return these requests within 2 seconds, thus successfully process 15 requests. The application is at the optimal condition, since all of its components are working at the maximal capacity.

<Case 2>: if twice as many intermediate request are generated for C as B: $R_C = 2\ R_B$.

After A, 10 R_C are sent to C, 5 R_B are sent to B. All the requests to B can be processed in one second. Because C's throughput is 5, it takes 2 seconds to process these 10 intermediate requests, or it can only process 5 of them in 1 second. Therefore, only 10

R_{APP} are successfully returned in 2 seconds. In this situation, B is **underloaded**, and C is **overloaded**.

For case 2, with GAE, to return all the 15 application requests within 2 seconds, the system needs to create another instance of the application, which leads to resource waste. If duplication can be operated at the service level, we can just duplicate C since C is the bottleneck of the application. Moreover, if there are services underloaded, they can be moved to less powerful server nodes.

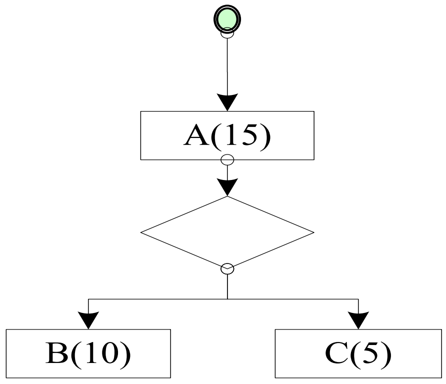


Figure 23 Sample Service components in a workflow

As one can see, application level duplication does not make the best use of resources, especially when some overloaded components become the bottleneck of the whole system.

Duplication Strategies

From coarse granularity to finer granularity, one can use diverse duplication strategies, including:

Duplication Applications: This is a similar strategy to the one used with GAE. However, since applications are service-oriented, not only the application instance is duplicated, but also the service instances belonged to the application instance, as illustrated in Figure 24.

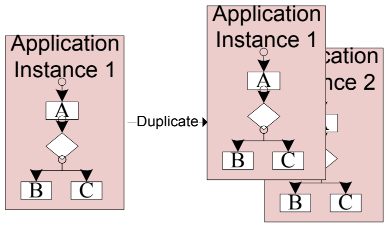


Figure 24 Application Duplication Strategy

Duplication Services: When the volume of requests to an application increases, and the application fails to meet the SLA specified by its users, it means the application is overloaded. However, according to the analysis in section 4.2, this might be caused by an overloaded service. Therefore, under this strategy, new service instances will be created and deployed to servers if all existing instances for a service are overloaded as illustrated in Figure 25. In the initial application instance, B and C, marked by darker color, are overloaded, so new service instances of B and C are duplicated.

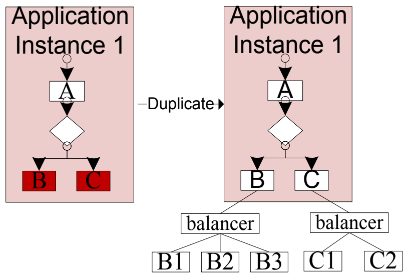


Figure 25 Duplication Service

Duplication at a mixture of Service/Application: One downside of the service duplication strategy is that the duplication always happens at the service level and if the request volume to an application keeps growing, too many service instances will be duplicated under one application instance, which will eventually overload the service balancers. If the duplication can happen at both application and service levels, shown in Figure 26, each load balancer might have fewer instances to manage, thus the balancing workload is further distributed on the application level. A load balancer is needed for the application instances too.

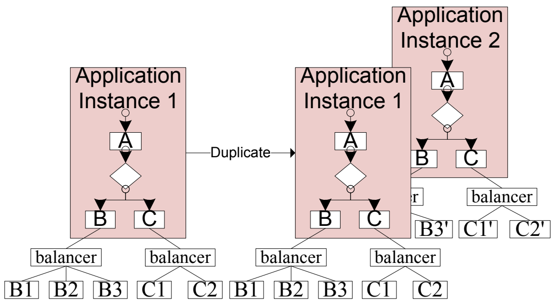


Figure 26 Duplication mixture of Application/Service

Two-Tier SaaS Scaling and Scheduling Architecture

Based on the above analysis, we propose a two-tier SaaS scaling and scheduling architecture described in **Figure 27**. Several components are as follows:

Application/Service Container: a runtime environment for an application/service which includes security, lifecycle management, monitoring and other supporting features.

Re-deployable Service Package: a package that contains the source code/complied code of a service and related resources required to deploy the service on to a service container. A new service instance can be rapidly created on another service container using the re-deployable package.

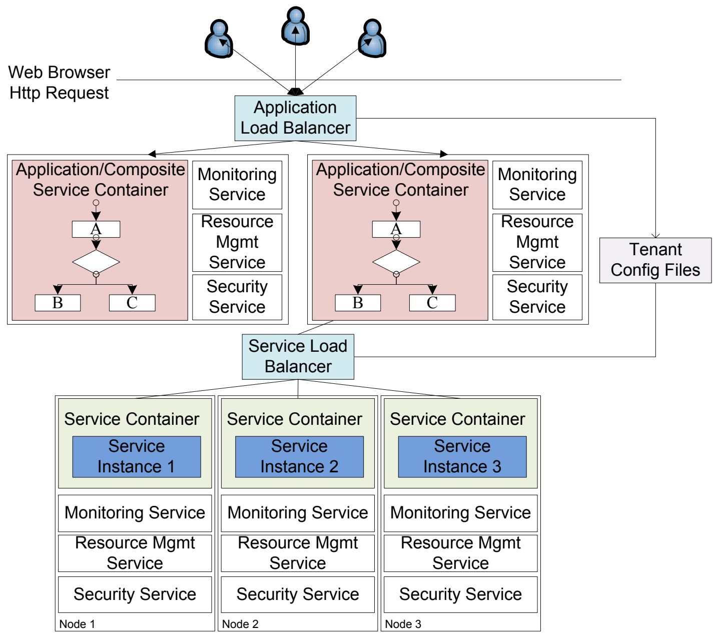


Figure 27 Two-Tier SaaS Scaling and Schedule Architecture

Service Replica/Instance: a concrete deployment of re-deployable service package on a service container, which is able to handle service requests. A service replica is also called service instance. A service replica might be customized to meet tenants' specific requirements according to the tenant configuration files.

Monitoring Service: It monitors the performance of a service/application instance closely. Based on historical data, it can report whether a service instance on its node is underloaded, overloaded or at its optimal condition.

Service Load Balancer: All the requests to a certain type of service are first routed to the load balancer for it. The load balancer manages all instances of a service belonged to an application instance. According to each service instance's throughput, status and configuration combined with the characteristics of requests, the load balancer routes requests to the different instances. If all the service instances become overloaded, the load balancer will create additional service instances on service containers running on newly provisioned server nodes using the re-deployable service package.

Tenant Configuration Files: In a multi-tenancy architecture environment, requests to an application may come from different tenants. Each tenant might customize the application/service instances according to their specific requirements. Tenant configuration files contain the customization information for each tenant that the load balancer can use when creating new application/service instances.

4.3 Cluster-Based Resource Allocation Algorithm

Different server nodes (virtualized or not) can have diverse levels of computing power. When allocating server nodes to run application/service duplicates, this difference needs to be considered. For instance, suppose the throughput of an instance, A1, of service A is 1000. When the number of requests sent to A1 per second increases to over 1000, the instance becomes overloaded. A2, a new instance of service A, will be created and deployed. If A2 is deployed to a similar server node like A1, the throughput for A2 will also be 1000, thus the throughput for service A is 2000 ideally. However, if we know that the frequency of the requests to service A will be no more than 1500/sec, there is no need to deploy A2 to a server node as powerful as the one that A1 is running. In the following section, we further extend our duplication framework by taking the computing power of each server into consideration.

Naturally, we expect that the server nodes in cloud computing clusters possess different degrees of computing power. It is easy to see that, one can cluster server nodes into different categories according their computing power. Moreover, virtualization techniques can slice or merge multiple machines computing power to allow finer-grained resource allocation. Generally, the computing power of a server node/computer is determined by many factors, such as CPU speed, bus speed, cache size, RAM size and type, etc. When considering virtualization, certain overheads should also be considered, such as virtual network connectivity, scheduling, etc. However, the performance overhead of virtualization has diminished significantly over the past few years [56]. How to model a server node's computing power is not the focus of this chapter. For the rest of the chapter, we assume that computation power of each server node can be modeled as a number. More details of modeling computers' power can be found in [57] [58].

Clustering Servers using Computation Power Model

Suppose on a cloud environment, there are N server nodes. For each server j, P_j is the computing power number for server node S_j. P_{min} is the lowest computing power number of all the nodes. We define our clusters as the following:

$$C_{2^i} = \{S_j \mid P_j \in [2^i \cdot P_{min}, 2^{i+1} \cdot P_{min}), i \in N\}$$

For example, if $S_j \in C_1, P_{min} \leq P_j \leq 2P_{min}$. $|C_{2^i}|$ is the total number of server nodes in cluster i.

The motivation of this clustering schema is to acknowledge the computing power differences among the server nodes, yet reduce the complexity by grouping the sever nodes with similar computing power together. This also serves a guideline for cloud providers to build server nodes whose computing power increases in the power of 2. However, obviously, we cannot keep doubling the computing power for a server node infinitely in reality. For example, in a cloud cluster deployment, the cluster with most powerful server nodes is C_{32}, which means the i can only go as high as 5. We call this cloud cluster deployment a level l deployment where l is the highest i value.

Cloud monitoring service maintains two pools for each cluster. $C_{2^i-allocated}$ contains all the server nodes that have been allocated for an application/service in cluster C_{2^i}, C_{2^i-free}, shortened as C_{2^i}, contains the server nodes free to allocate in cluster C_{2^i}.

Cluster-Based Duplication Algorithm

According to the clustering schema, the minimum allocatable resources are the server nodes belonging to cluster C_1. We define a server node in C_1 as a resource measurement unit, written as r_1. Accordingly, a server node in C_4 represents 4 resource units, written as r_4.

Computing power requirements can be translated to the number of resource units needed by examining SLA, analyzing programs' characteristics and comparing with historical data. In most of the cases, the request for computing recourse is proportional to the how many requests the applications/services serve per second. For the rest of the chapter, we use r_x to present a resource request that needs X resource unit. Accordingly, we have $r_x + r_y = r_{x+y}$.

Optimization Goal of Resource Allocation

Let t_{2^i} be the number of server nodes chosen from cluster C_{2^i}, a set $T = \{t_{2^i} | t_{2^i} \in N, i \in N\}$ represents one way to allocate the resource. $t_{total} = \sum_{i=0}^{l} t_{2^i}$ is the total number of allocated server nodes of T, $r_{total} = \sum_{i}^{1\to l} t_{2^i} \cdot r_{2^i}$ is sum resources of all the selected nodes of T. **T** is the set that contains all T. Given a resource request r_x, one first searches solutions with the least r_{total} without going below r_x; if there are multiple candidates, we choose the one that uses the fewest server nodes.

The goal is set based the following two considerations:

- Avoid resource waste. For example, a resource request r_{15}, and we have two possible resource allocations T_1 and T_2: T_1 uses r_{16} resource, and T_2 uses r_{18}, we prefer T_1 over T_2 as T_1 uses less resources.
- Avoid scheduling overheads. Generally, the more number of server nodes, the more scheduling overheads. For example, for a resource request r_{16}, we have two allocations T_1 and T_2 that both uses r_{16} resources. T_1 only uses one server node from cluster C_{16}; T_2 uses two server nodes from cluster C_8. We prefer T_1 over T_2 as T_1 uses fewer server nodes.

To formalize the computation goal of our allocation algorithm, we define $\mathbf{T}_{\text{candidate}}= \{T \mid T\in\mathbf{T}, r_{total} \leq r_x\}$ as all the possible allocations that satisfy the resource request. $r_{min} = \min \{r_{total} \mid T\in\mathbf{T}_{\text{candidate}}\}$ and $\mathbf{T}_{\text{min-R}} = \{T|T\in \mathbf{T}_{\text{candidate}}, r_{total}= r_{min}\}$

The optimal allocation can be defined as follows:

$$\boxed{\text{Given a resource request } R_x\text{, find a set } T_{\text{opt}}\in\mathbf{T}_{\text{min-R}}\text{ , } t_{opt\text{-}total}=\min\{t_{total} \mid \mathrm{T}\in \mathbf{T}_{\text{min-R}}\}}$$

Resource Allocation Algorithm

Given a resource request r_x, the algorithm needs to find an optimal allocation T_{opt}. The algorithm uses a greedy approach to achieve the optimization goal as shown in Algorithm 1. Basically, the algorithm tries to allocate server nodes with computing power no greater than r_x in a descending order, meaning it allocates more powerful server nodes first till the server nodes in C_1.

If the total resource of T is equal to r_x, then it is the optimal allocation. However, some clusters might not have enough server nodes available, so it is possible that the total resource of T is less than r_x. In this case, the algorithm retrieves back to the last level where adding one more server node would cause r_{total} to be greater than r_x. It releases all the server nodes from lower levels, and adds one server node from this level. At this time, the r_{total} is the r_{min}. The algorithm then releases all the server nodes, and uses r_{min} instead of r_x to another round of allocation from the beginning and returns the allocation it finds.

In table 1, each value represents the current available resources in cluster C_x (x = 1, 2, 4.., 32). The initial distribution of each cluster is shown in the "initial" column. When getting the request of $r_{31,}$ as step 1, the algorithm skips C_{32} *as* $r_{32}>r_{31.}$ It checks if there is any free server nodes in cluster C_{16} and allocate one if there is. After this step, the values are shown in column "step 1". Specially, we circle C_{16}, as one server node is allocated in this cluster. As step 2, allocating another server node from C_{16} will cause r_{total} to be greater than r_{31}, therefore the algorithm allocates a server node from C_8 instead. Similarly, we keep allocating server nodes till C_1. At this time, $r_{total} = r_{30} < r_{31}$, and the last level adding one more server node of which will cause r_{total} to be greater than r_x is level 1, C_2. The algorithm adds one server node from C_2 to the allocation.

Now, we have $r_{total} = r_{32} = r_{min}$. The algorithm then releases all the server nodes in the allocation, and uses r_{32} to return the algorithm. Therefore, we cannot find an allocation that just meets the resource request r_{31}, instead, we allocate 1 server node from C_{32}. Note there are other possible allocations that also have r_{32} resource, however, our algorithm only returns the one with the fewest server nodes.

Algorithm 1: Resource Allocation (RA)

```
Input: $l$ level cloud clusters, given resource $r_x$.
Output: An optimal allocation $T_{opt}$
Signature: T allocate(int $r_x$)
for(int $i$ = 0; $i \le l$; $i$++) //initialization
```

```
  $t_{2^i}$= 0;
int $j = l + 1$;
// j is the last level adding one more server node of which // will cause $r_{total} > r_x$
for(int $i = l$; $i \geq 0$;$i$--) {
    while ($|C_{2^i}|$!=0 && $r_{total} + r_{2^i} < r_x$) {
        $t_{2^i}$++; $|C_{2^i}|$--;
    }
    if ($|C_{2^i}|$!=0)  $j = i$;
}
if($r_{total} == r_x$) return T;
for(int $i = 0$; $i < j$; $i$ ++) {
  $t_{2^j}$--;  $|C_{2^j}|$++;
}
if ($j == l + 1$) return null;
else {
  $t_{2^j}$++;  $|C_{2^j}|$--; int $r_{min} = r_{total}$;
  release(T);//release server nodes in  current allocation
  return allocate($r_{min}$);
}
```

Example 1: Considering a level 5 cluster C = {C_1, C_2, C_4, C_8, C_{16}, C_{32}} in cloud deployment, a resource request r_{31}, use the algorithm to find the optimal allocation $T_{opt.}$

Table 7 Running Example for RA

	initial	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
C_1	0	0	0	0	0	⓪	0
C_2	3	3	3	3	②	1	3
C_4	10	10	10	⑨	9	9	10
C_8	2	2	①	1	1	1	2
C_{16}	2	①	1	1	1	1	2
C_{32}	2	2	2	2	2	2	①

Resource Reallocation

When the usage of an application/service increases/decreases, resource might be added/withdrawn. However, the additional resource might cause the allocation not to be optimal. For example, initially a resource request for an application/service is r_6, an optimal allocation is to choose one server node from C_4 and one server node from C_2. When the usage increases, a r_2 additional resource is needed. One possible solution is to add another server node from C_2; however, since the total resource needed now is r_8, we could deploy an application/service instance onto a server node from C_8, and retract the two nodes allocated previously.

The algorithm for resource reallocation can be easily described based on the resource allocation algorithm.

```
Algorithm 2: Resource Re-allocation (RR)
Input: $l$ level cloud cluster development, an existing allocation T whose $r_{total}=r_x$, an additional resource $r_y$.
Output: An optimal allocation $T_{opt}$
Signature: T re-allocate(T , int $r_y$)
$T'$ = allocate($r_{x+y}$).
Compare the new allocation $T'$ with the old allocation $T$. Create application/service
```

instances on server nodes that were not in *T*, and retract the server nodes in *T* that are not in *T'* now.

4.4 Lazy or Pro-Active?

All the discussions in 4.4 use a lazy way to handle service requests, which is "wait until needed". While in some cases, when one can predict the demand request in the near future, pre-allocate resources can improve the system performance greatly. For example, tax return services are burst from January to April 15 every year, in that case, one can duplicate the service earlier and pre-allocate the resources needed, which in turn can improve the tenant's performance. In this section, we further propose a pro-active replication solution based on prediction models.

To predict the future demand of services, multivariate exploratory techniques in data mining and machine learning can be applied to identify patterns for future service demand. Time Series Analysis is one commonly used method, which includes identifying the nature of the phenomenon represented by the sequence of observations, and forecasting future values of the time series variable. After identifying the pattern of observed time series data, one can interpret and integrate it with other data and extrapolate the identified pattern to predict future events.

There are two types of time series patterns, trend analysis and seasonality. Those two general classes of time series components may coexist in cloud service demand. For example, service request of a tenant can rapidly grow over years but they still follow consistent seasonal patterns (e.g., as much as 95% of yearly tax return each year are made from January to early April, whereas only 5% in other months). Before trend analysis, data cleaning is necessary to remove the error using smoothing, such as using mean/median. Then one can fit a distribution function to the observed data, for example, the fitting function could be a linear, logarithmic, exponential, or polynomial function. Hypothesis test can be used to evaluate the fitting function at last.

Given the two possible models, lazy and pro-active ones, how to choose models according to specific application requirement is very interesting. Intuitively, one can see that the service requests can be coarsely classified into two categories according to their QoS (Quality of Services): low penalty task and high penalty task. The first type means even the service requirement could not be satisfied, the penalty is low and acceptable, one can use lazy model and duplicate until must do so. While the latter has a high penalty when short of service supply, it is highly recommended to use pro-active model to plan and duplicate ahead and reduce the penalty. Also it is easy to see, some service has a periodical pattern, for example, tax return services are always burst during Jan to April every year, it is better to use pro-active model and duplicate service earlier.

In a word, customers can choose different model according to their service requirement, as well as their budgets, QoS, and other factors to make a decision. The model proposed in this chapter provides flexibility for model choices.

4.5 Simulation Results

We built a simulation framework that is able to simulate service composition, service invocation, cloud deployment, resource allocation, application/service duplication and load balancing. Due to the page constraints, we only show the simulation results of resource allocation algorithm. Details of the simulation framework and other simulation results will appear in the extended version of this chapter.

We compare our resource allocation algorithm with two straightforward algorithms.

• *Small First (SF) Algorithm*: It tries to allocate less powerful server nodes from lower level clusters first. Using the same example from section 4.3, it will return an allocation T = {t_1 = 0, t_2 = 3, t_4 = 7, t_8 = 0, t_{16} = 0, t_{32} = 0}, $r_{toatl} = r_{34}$.

• *Big First (BF) Algorithm*: It tries to allocate more power server nodes from higher level clusters. Using the same example from section 4.3, it will return an allocation T = {t_1 = 0, t_2 = 0, t_4 = 0, t_8 = 0, t_{16} = 0, t_{32} = 1}, $r_{toatl} = r_{32}$.

Simulation Setup: We tested 3 three allocation algorithms with the following 5 level cloud deployments.

$\|C_1\|$=10	$\|C_2\|$=20	$\|C_4\|$=30	$\|C_8\|$=50	$\|C_{16}\|$=80	$\|C_{32}\|$=100

We first randomly generated 200 resource requests between r_1 and r_{50}. We tracked 3 numbers that are the most interesting. The first is utilization rate (UR), which is defined as the ratio of total resource requested to total resource allocated. The UR is an indicator how effective computing resource is utilized. If the UR is too low it means that much more resource is allocated than needed, leading to resource waste. Figure 28 shows how UR changes as the request number increases to 100 for the three algorithms.

As shown in the figure, the UR for RA decreases slowly as the request number increases and the lowest rate is around 0.94. This is because RA tries to meet the resource requests without going over, and when it has to, it only goes over the minimum possible. The UR for SF starts off strong then decreases sharply and the lowest comes at 0.72, because SF allocates server nodes from lower level clusters first, and as these server nodes quickly run out, it has no choices but to keep giving out more powerful server nodes even for smaller resource requests. The trend for UR of BF is to increase, which can be explained by the fact that BF first allocates server nodes from higher level clusters that possess more computing power than requested, which results in low UR in the beginning. As these server nodes with higher computing power number run out, BF starts to return allocations with allocated resource closer to the requested resource.

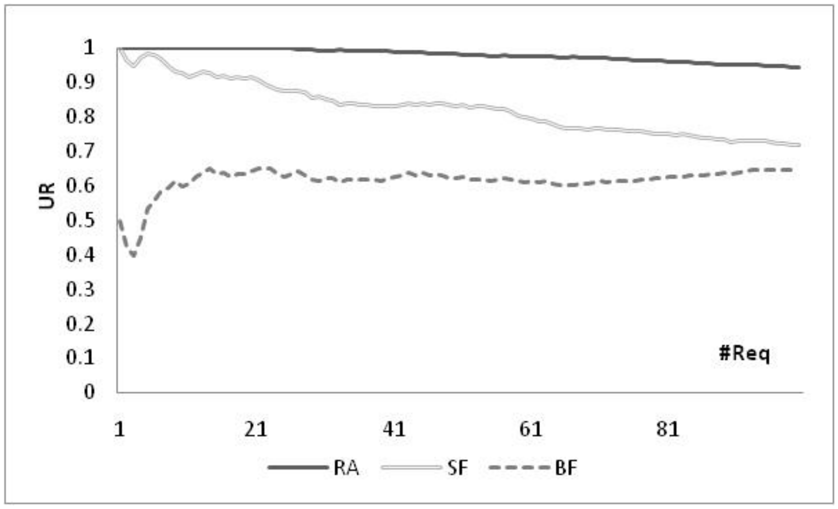


Figure 28 Utilization Rate Diagram

The total allocated server node number (SNN) affects the performance of load balancers. More server nodes mean more workload for load balancer. Figure 29 shows how SSN grows as the resource request number increases for the three algorithms.

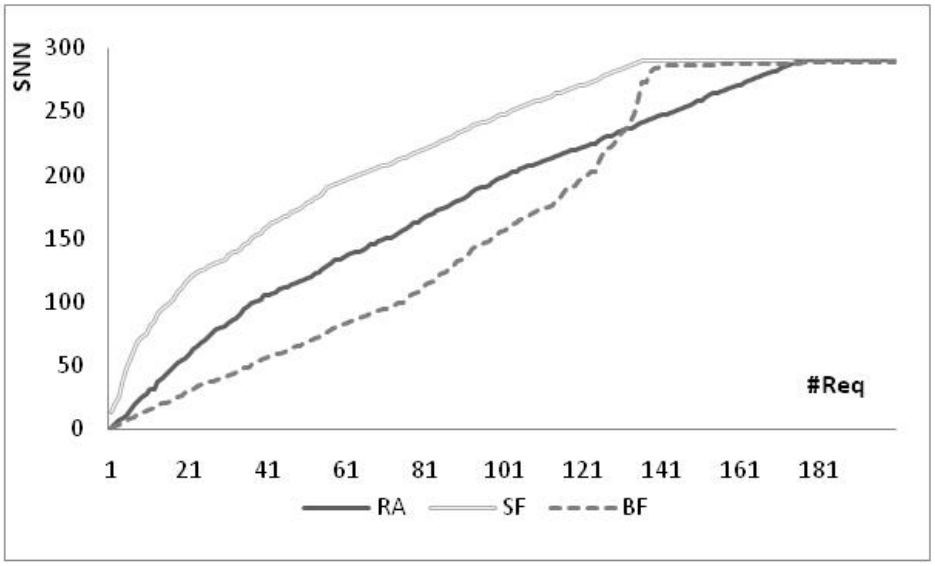


Figure 29 Server Node Number Diagram

At the beginning, SNN for SF grows the fastest, followed by RA, then BF. This is expected as SF first allocates less powerful server nodes thus uses more of them for the same resource requests than RA, BF. BF has the lowest SNN initially, however, its increasing rate grows higher, and at around 130th request, it surpasses the SNN of RA. This is because BF aggressively allocates more powerful server nodes first, which leads to smaller SNN. When server nodes from higher level cluster become unavailable, it is forced to use more server nodes from lower level cluster.

Each cloud deployment has limited resource; therefore, it can only serve up to a certain number of resource requests. Success Rate (SR) measures the percentage of the resource requests allocated successfully within the 200 resource requests generated. Figure 30 shows the SR for the three allocation algorithms. RA successfully allocates 175 of 200 resource requests, thus has the highest SR, while SF, BF only successfully allocates 135, 140 resource requests respectively. The numbers shown in the curly brackets are the differences of SF, BF compared to the SR of RA.

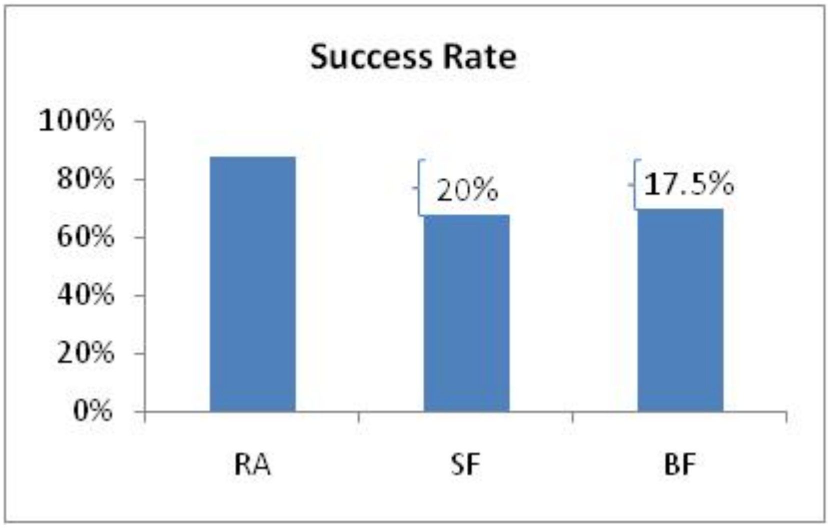


Figure 30 Success Rate

4.6 Related work

Service Duplication and Architecture. In [59] Recardo and etc. first identified the importance of replication in enhancing multi-tier information system's availability and scalability, then discussed several replication patterns in an architectural point of view. The authors only discuss replication in two major tiers: Application and Database, and they did not take the composable characteristic of SOA, and overlook the possibility of replicate parts of the applications. In [60], a redundancy protocol adapted from 3 phase commit protocol (3PC) for SOA is proposed to ensure that "at most one" service provider is executed when there are multiple service providers available. In [61], the authors proposed a Dynamic Service Replica Process (DSRP) system which has a mechanism to create and delete service automatically to achieve better load balancing and performance.

Several studies that focus on the various replication strategies, such as 'active' and 'passive' techniques, are presented by Maamar et al. [62], Guerraoui and Shipper [6].

These papers discuss replication communities in Web Services, survey replication techniques, and evaluate temporal and special redundancy techniques.

Architectures are described by Osrael et al. [63], who propose a generalized architecture for a service replication middleware, and Juszczyk et al. [64], who describe a modular replication architecture. Among the frameworks are those described by Salas et al. [65], Engelmann et al. [66], and Laranjeiro and Vieira [67]. They propose an active replication framework for Web Services, a virtual communication layer for transparent service replication, and a mechanism for specifying fault tolerant compositions of web services using diverse redundant services, respectively.

The focus of these studies is on describing and implementing various strategies for invoking and maintaining replicated services. However, most do not treat services as autonomous components, and so their applicability is limited for an SOA system.

Virtualization: is widely used in Cloud computing, the benefit of VM covers many perspectives, e.g. it reduces expenses in money, power, and space and improves agility to meet changing requirements. Due to sited specified, loosely coupled, non-portable restrictions, it is hard to specify, communicate and manage diverse requirements. Virtual Machine Contracts (VMCs) [55] is a platform, with three components such as Policy Language Processing, Rules Database, Enforcement Element Notification, and these three parts can automatic communicate to meet the requirement. SnowFlock [54] is provided to manage application state and cycle. In each window, arriving jobs are analyzed, producing a number of virtual machine according to the volume of incoming requests for each tenant. At the same time, each job is assigned to an available virtual machine class as it arrives, if one has been provisioned with sufficient resources to meet its requirements. In this chaper, we propose a novel two-tier duplication model with VM at lower level to support flexible scheduling.

4.7 Conclusion

In this chapter, a two-tier SaaS scaling and scheduling architecture was proposed. In this architecture, duplication can happen at both service and application levels to avoid resource wastes. In addition, discussion was provided on how to choose between two duplication timing models, lazy and pro-active. To further improve resource utilization rate a cluster-based resource allocation algorithm was proposed. Simulation results showed that the resource allocation algorithm can provide better resource utilization rate while using fewer server nodes.

5 ONTOLOGY-BASED SERVICE-ORIENTED SIMULATION FRAMEWORK WITH MICROSOFT ROBOTICS STUDIO

In Service-Oriented Architecture (SOA), the concepts that services can be discovered and application can be composed via service discovery bring great flexibility to application development. Microsoft Robotics Studio (MSRS) is a recent initiative in applying SOA to embedded systems and one of its key features is its 3-D simulation tool that allows applications to be simulated before deployment. This chapter proposes an ontology-based service-oriented simulation framework with MSRS by adding a set of ontology systems, i.e., service ontology, workflow ontology, entity ontology, and environment ontology. These ontology systems store relevant information useful to compose simulation applications, and items stored also cross reference to each other to facilitate reusability and rapid application composition, This chapter then provides a detailed case study on a popular robotic game Sumobot using MSRS to illustrate the key concepts and how they can support rapid simulation development.

5.1 Introduction

Service-Oriented Computing (SOC) using Service-Oriented Architecture (SOA) has received significant attention as major computer companies such as Oracle [68], Cisco [69], HP [70], IBM [71], Microsoft [72], SAP [73], as well as government agencies such as U.S. Department of Defense (DoD) [74]. SOA is characterized by loosely-coupled services, open standard interfaces, service publication, dynamic discovery of services, and dynamic composition using services discovered. SOA is being used not only for e-commerce applications, but also for embedded applications. For example, DoD is developing several large IT projects based on SOA, and some of them for real-time embedded applications. Microsoft Robotics Studio (MSRS) [75] is another example and this is a recent initiative by Microsoft to incorporate SOA into embedded system development.

SOA has its own lifecycle models and processes such as IBM Foundation Lifecycle [76], and they are different from traditional software development processes. As services may be discovered and selected at runtime, it is important to verify these newly discovered services before they are used in applications. One important verification approach in SOA application development is simulation. Specifically, SOA simulation is considered a key technique for business modeling and process management for IBM WebSphere applications [77]. MSRS also uses simulation to verify the robotics application.

As SOA computation, infrastructure, development processes, runtime environment and model-driven approach are different from traditional software, thus SOA simulation will also be different [78]. For example,

- With respect to SOA computation, SOA simulation needs to simulate services, workflows, policies, and collaboration.
- With respect to SOA infrastructure, SOA simulation needs to address SOAP [79], service publishing, discovery, composition, monitoring, deployment, and possibly even re-composition and reconfiguration. Furthermore, SOA hardware infrastructure can also be simulated including visualization, HasS (Hardware as Services) / IaaS (Infrastructure as a Service) [80] or PaaS (Platform as a Service) such as in the Intel SOI (Service-Oriented Infrastructure) project [81] [82].
- With respect to SOA development processes, SOA software development process is

different, and simulation can play a significant role as SOA requirement model, design and design architecture, services, workflows, policies [83] [84], dynamic collaborations [38], monitoring mechanisms can be simulated. For example, the IBM Foundation lifecycle is different from the traditional software development process where software development and execution are all considered in the same development model. Previously, software lifecycle models only consider software development phases and do not consider execution issues at all [85].

- Due to the dynamic nature of SOA, static analysis may not be sufficient and dynamic simulation will be needed. Specifically, simulation should be able to perform the following tasks:
  - Dynamic application composition simulation: As SOA software can be composed at runtime with newly discovered services, the simulation task should be able to simulate the re-composition process.
  - Runtime behavior and performance simulation: The simulation task should be able to perform the runtime behavior and performance simulation to make sure the newly recomposed application workflow will behave as expected.
  - Dynamic collaboration simulation: With SOA Dynamic Collaboration Protocol (DCP), collaboration can be dynamically established at runtime between two parties not knowing each other earlier [38]. Simulation of this DCP can ensure that all dynamic collaboration can be properly established for actual deployment.
- SOA development is model-driven; however, SOA models may be rather different from traditional software models. Specifically, SOA modeling language may need to model services, workflows, policies, SOA protocols, monitoring mechanisms, and

thus SOA simulation will also be model-driven. Each component in the simulation framework, including the simulation engines as well as the application components and tasks, are modeled as services or workflows using various modeling languages. Code may also be automatically generated from the model. One such modeling language is PSML (Process Specification and Modeling Language) [22] which can model processes as well as policies, and can be used to generate code.

- The SOA simulation infrastructure can also be structured in a service-oriented manner, i.e., each component in the simulationinfrastructure, including the simulation engines as well as the application components and tasks, can be modeledas services or workflows using various modeling languages.
- New SOA approaches such as cloud computing [3] pose new challenges to SOA simulation. In these environments, resources owned by different organizations are distributed geographically and interconnected via wide-area networksor Internet [86]. These introduce a series of problems such as resource management and application scheduling, resourceand policy heterogeneity, fault tolerance. Simulation can play a significant role in cloud computing by simulating itsbehavior and performance. The simulation can focus on aspects such as resource scheduling, performance, interactionsamong resource brokers and users over a network.
- Another interesting approach is Software as a Service (SaaS) [87]. SaaS allows software access via networks, and softwarecan be continuously updated so that a user will always use the latest version of the software over the network. Essentially,a user may be able to make specific options about applications, and the system will return appropriate versions to fit customers’needs. Simulation can play a role in SaaS.

Before the software is uploaded in the SaaS site, it can be simulated toevaluate its performance and scalability. Different workloads and user requests can be modeled to see how the system willperform in the SaaS environment.

This chapter is organized as followed: 5.2 will briefly overview the existing SOA simulation works; Section 5.3 introduces the MSRS architecture; Section 5.4 explains the collaborative SOA simulation framework with MSRS; Section 5.5 uses an example and simulation result to illustrate the effectiveness of the proposed framework. Section 5.6 concludes this chapter.

5.2 Previous SOA Simulation Research

Service-oriented simulation is supported by a number frameworks, including XMSF (Extensible Modeling and Simulation Framework) [88] [89] and the simulation grid system Cosim-Grid [90] and GridSim [86]. XMSF creates a modeling and simulation framework that utilizes a set of web-enabled technologies to facilitate modeling and simulation applications. XMSF involves Web/XML, Web services, and internet/networking to improve the interoperability. A significant contribution by XMSF is its web-based RTI (Runtime Infrastructure). Cosim-Grid is a service-oriented simulation grid based on HLA, PLM (Product Lifecycle Management) [91], and grid/web services. It applies OGSA (Open Grid Services Architecture) [92] to modeling and simulation to improve HLA in terms of dynamic sharing, autonomy, fault tolerance, collaboration, and security mechanisms. However, both XMSF and Cosim-Grid utilize web services as components in the component-based development without fully taking the full potential of SOA [93]. For example, SOA offers dynamic composition, re-composition, and reconfiguration, and these can be incorporated into the simulation framework.

Other related research includes INNOV8 [94], which is an IBM training simulator. It is a three-dimensional simulation video game that puts a business person in a virtual office with the task of constructing a more efficient company. Process improvement is a critical component to service-oriented architectures. INNOV8 is meant to resolve people's lack of skills in understanding the business process management and improving a company's internal business processes by offering a simulator to the end user. SIMPROCESS [95] is a process analysis tool commonly used to support business process improvement and operations research projects. It provides modeling, simulation and analysis capabilities. The models are used to define complex business process flows that are represented graphically as activity diagrams. SIMPROCESS can be used in SOA by providing simulation models as a callable service. However SIMPROCESS does not capture the dynamic features of SOC, thus the simulation is only at static level. ISTF (Interface Simulation and Testing Framework) [96] is an extensible SOA simulation tool offered by IONA. ISTF simulates the end-to-end distributed application scenarios and demonstrates how individual components will interface with each other in production. Its simulator simulates both the client and server side. It also has a set of shared test cases. The matching simulators provide a common, codified interpretation of interface behavior to both the server and client development groups. The use of shared test cases ensures a common interpretation of the data requirements for each interface. ISTF cannot simulate the workflow within the SOA environment and it does not have analyzing features and other dynamic features for the SOA.

DDSOS (Dynamic Distributed Service-Oriented Simulation) [20] [97] is an SOA simulation framework that provides simulation runtime services and supports. DDSOS has RTI like the one in HLA [98], however it extends RTI by introducing several new services so that it can provide on-demand simulation. Within this framework, simulation code can be dynamically generated and configured whenever demanded by the users. The DDSOS framework provides the following services and tools:

- Two layers of modeling support. At the high layer, it allows a user to specify components and their relationships. At the low layer, it allows a user to specify processes, services, and workflows. Both are supported by PSML [22] modeling language. Models specified can be analyzed using various analysis tools such as C&C (Completeness and Consistency) tool.
- An on-demand automated dynamic code generator (service) can generate executable code for simulation and for real applications directly from the model (specification) written in PSML.
- An on-demand automated dynamic code deployment service can support rapid and automated simulation/real code deployment.
- Simulation engines that carry out simulation tasks form a simulation federation. These engines can be geographically distributed on simulation services that are interconnected via network.
- An extended runtime infrastructure (RTI) to support the simulation services' operations.

As SOA development is mainly model-driven, thus SOA simulation will be model-driven. Each component in the simulation framework, including the simulation engines as well as application components and tasks, are modeled as services or workflows using various modeling languages such as BPEL [99] or PSML-S [22]:

- PSML-S is an SOA modeling language and it has the single-model-multiple-analysis (SMMA) capability [22]. PSML-S has three models. The Element Model specifies concepts and specific relations among them; The Structure Model defines connections among model elements; The Behavior Model defines the process of an action in response to an event. PSML uses a control flow to model system behaviors.
- Simulation can be applied to various activities in each phase throughout the lifecycle of SOA application development using the model driven approach. The activities that can be simulated include service publishing, service discovery, service composition, dynamic architecture, reconfiguration, dynamic collaboration, policy enforcement.
- In SOA application development, simulations may be done under different configurations such as: service simulation only, service and workflow simulation, hybrid simulation where real services participate and complete pure simulation

PSML is used as the modeling language, and the modeling process and techniques are as follows:

1) Model the identified services in an application: identified services can be mapped to PSML model elements and PSML structure elements;
2) Model the identified workflows or collaboration: workflows can be mapped to PSML model elements and PSML behavior model. Furthermore, SOA software architecture can also be modeled by PSML;
3) Perform analyses (such as completeness and consistency analysis) on the model;

4) Generate the simulation code based on the model: PSML uses three models to generate code: service model, system model, and environment model. The system model and the service model describe the platform-independent information, and the environment model provides the platform-specific information. Template-based approach and integration techniques are employed in the code generation process;
5) Deploy simulation code in a service-oriented simulation framework such as DDSOS to run the simulation;
6) Repeat this process until the system has shown satisfactory behaviors and performance.

SOA allows services to be discovered and composed at runtime, and thus SOA behavior can be determined at runtime only. However, in this approach, collaboration workflow is still determined at design time, except that services participating in the workflow will be determined at runtime. A more complex scenario is that even the collaboration workflow will be determined at runtime. In this case, two applications need to establish their collaboration workflows and identify participating services at runtime. The Dynamic Service-Oriented Collaboration Simulation (DSOCS) framework [100] addresses this issue. The DSOCS framework integrates the SOA dynamic collaboration and simulation concepts. The framework supports modeling and simulating systems with the distributed, interactive, and discrete–event driven focuses. An important element of DSOCS is dynamic collaboration configuration service which generates the simulation application configuration at runtime based on the application template and the service collaboration specifications.

Another important feature of SOA is policy-based computing where constraints can be specified as policies and enforced at runtime. While some SOA systems may have the same workflows and services, their runtime behaviors can be rather different if different sets of policies are used. For example, a tightly controlled policy may stop every message flow, inspect the data going through to ensure security policies, however another set of policies may do only random sampling in the background, and system performance will be drastically different. Thus the overall SOA system behavior is not only determined by the system specifications and implementations, but also the policies used. Policies can be specified by an independent third party rather than the system developers. However, policies themselves also need to be verified, and their impacts to the overall system need to be evaluated, and in many cases their impacts need be determined by simulation before deployment because both the system software and policies must be running concurrently to evaluate their interactions. Specifically, an SOA application can be simulated with different sets of policies to see how the system will behave under the control of different sets of policies. This also introduced new and interesting problems to software verification as previously computation behavior can be mostly determined once the system specifications and implementations are completed and known, but SOA software needs to be evaluated with policies that may be developed by others. Several simulation frameworks have been proposed to address this policy-based computing to control data access [83] [84], service collaboration [101], temporal relations [102], and to verify the consistency and completeness of policies [101].

5.3 Microsoft Robotics Studio

MSRS is a service-oriented development and simulation platform to create robotics applications, and its architecture is shown in Figure 31.

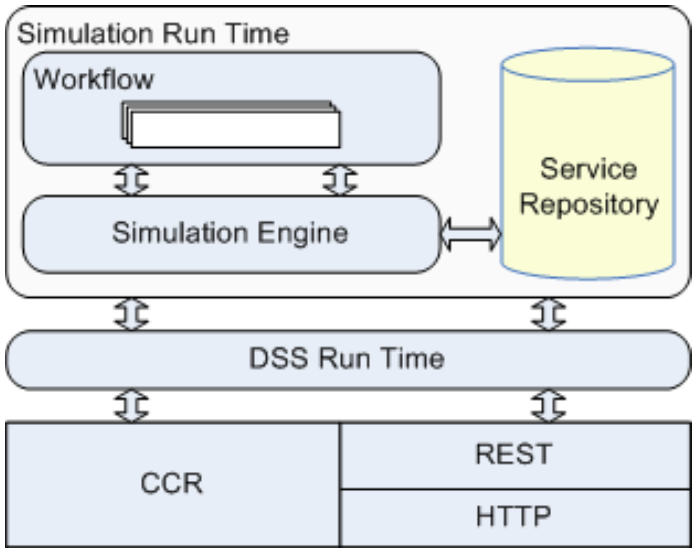


Figure 31 Rebotics Sutdio Simulation Architecture

As shown in Figure 31, DSS and CCR are the two most important components that provide the facilities to run robotic programs and their simulations. A brief explanation is given as follows:

Concurrency and Coordination Runtime (CCR) provides a concurrent message-oriented programming model for SOA application development, and it manages asynchronous operations, dealing with concurrency, exploiting parallel hardware and handling partial failure.

Decentralized System Services (DSS) provides interoperability between simulation engine and CCR, and it allows application composition using remote services. The DSS runtime is built on top of CCR and does not rely on any other components in MSRS. It provides a hosting environment for managing services and a set of infrastructure services that can be used for service creation, discovery, logging, debugging, monitoring, and security.

5.4 MSRS Simulation Runtime

For each simulation task, the user composes an application using the basic activities provided by MSRS, which are common programming constructs such as data, variable, if-then-else constructs, and other services that are related to collecting data from sensors and controlling the actuators of robots. When the users compose applications, they either compose with the actual services or generic services which can be bound to real services at runtime. Using generic services provides flexibility to the applications, because the users can run the same application in a simulated world on different simulated robots or in the real world on different robot hardware, with the generic services bound to different services using different configuration files. Figure 32 shows MSRS Simulation Runtime Workflow, and a sample configuration (manifest) file is shown in Figure 33. For example, a driving service is usually represented by the motor base entities or the wheel entities in the simulated world. When a simulation task starts, the manifest file is loaded first. The services listed in the manifest are started and configured accordingly, then bound to the generic services in the workflow. The simulation process is based on the logic described by the workflow. The entire simulation process can be monitored through a visualization service built in the simulation engine service.

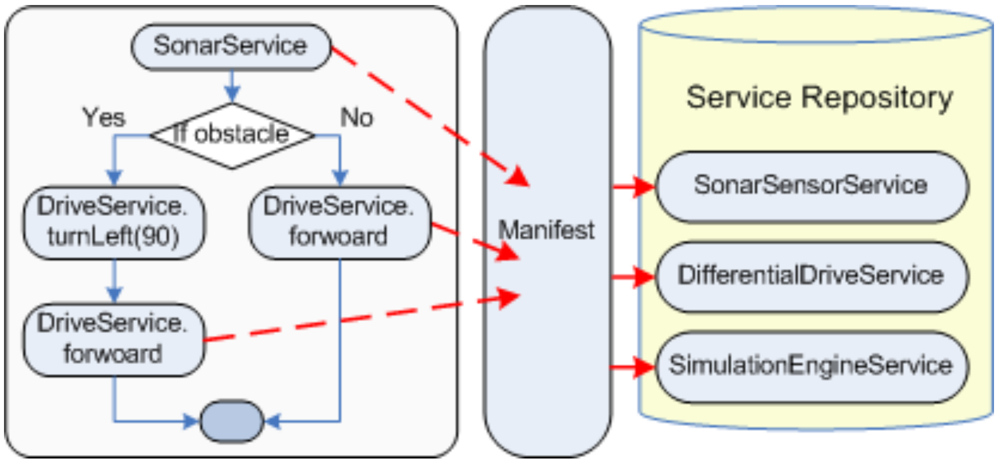
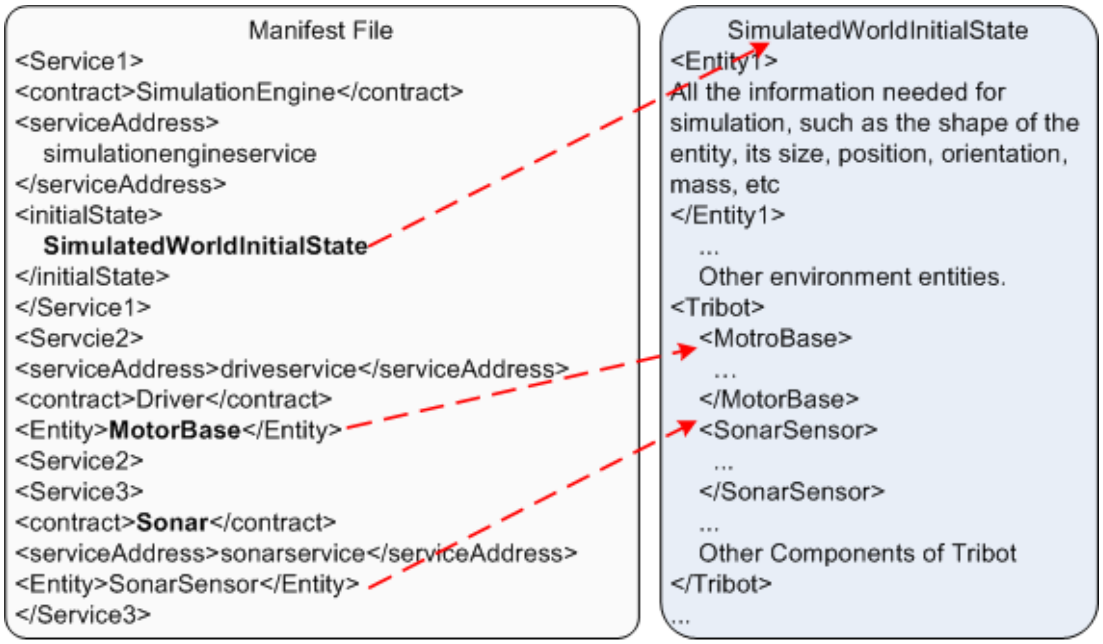


Figure 32 MSRS Simulation Runtime Workflow



```
Manifest File
<Service1>
<contract>SimulationEngine</contract>
<serviceAddress>
   simulationengineservice
</serviceAddress>
<initialState>
   SimulatedWorldInitialState
</initialState>
</Service1>
<Servcie2>
<serviceAddress>driveservice</serviceAddress>
<contract>Driver</contract>
<Entity>MotorBase</Entity>
<Service2>
<Service3>
<contract>Sonar</contract>
<serviceAddress>sonarservice</serviceAddress>
<Entity>SonarSensor</Entity>
</Service3>
```

```
SimulatedWorldInitialState
<Entity1>
All the information needed for
simulation, such as the shape of the
entity, its size, position, orientation,
mass, etc
</Entity1>
   ...
   Other environment entities.
<Tribot>
   <MotroBase>
     ...
   </MotorBase>
   <SonarSensor>
     ...
   </SonarSensor>
   ...
   Other Components of Tribot
</Tribot>
...
```

Figure 33 Manifest File And Simulation Environment File

This architecture provides flexibility such as composing with generic services gives the users the freedom of running it several times by using different configuration files.

5.5 Collaborative Simulation Framework with MSRS

Based on the existing MSRS architecture, this chapter proposes an ontology-based collaborative service-oriented simulation framework as shown in Figure 34. This framework is fully service-oriented as each component can be viewed as a service, and it has common SOC elements such as ontology, service and workflow discovery, and service collaboration.

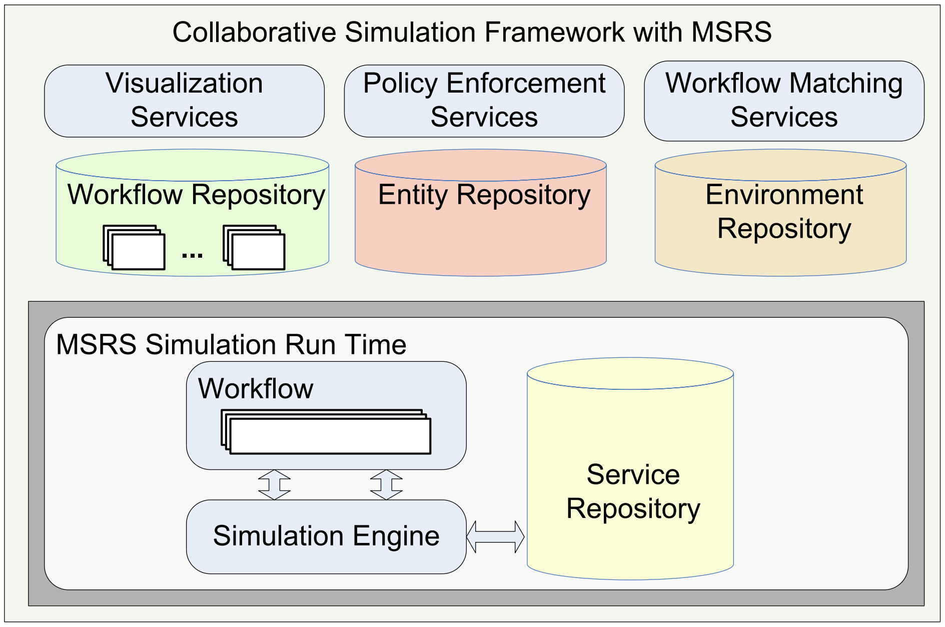


Figure 34 Collaborative Simulation Framework

Specifically, this framework has the following new features:

1) This framework has more service-oriented features. Simulation entities, environments can also be published as services shared across the internet, so a simulation task can be easily modified to run in different environments with different simulated entities.

2) This framework separates the simulation environment and simulation entities. With them separated, one does not need to create a new configuration file and a new initial state of the simulated world when running the simulation in a different environment with the same simulation entities. Moreover, new environments can be easily composed using simulation entities.

3) The framework has a set of ontology systems that facilitate service matching and discovering for application development. Several domain-specific ontology systems were

added to classify function services, workflow templates, simulation entities, simulation environments. The reasoning abilities provided by the ontology systems can facilitate the service matching process, and provide a certain level of flexibilities by returning the most compatible services when a perfect match cannot be found, and reduce the manual work for a user.

4) This framework is divided into platform dependent part and platform independent part. Users can use platform-independent modeling languages to specify their workflows and map them into specific platforms to perform simulation. In this way, users will be able to share platform-independent parts with others while performing simulation on specific platforms at the same time. This will be explained in detail in 5.4.2.

5) The framework provides cross-referencing and traceability among all simulation assets such as services, workflows, simulation entities and simulation environments utilizing ontology systems, so it is relatively easy to adapt to changes.

6) This framework is a collaborative framework. It can be empowered by the Web 2.0 principles [103], specifically "users are treated as co-developers". In other words, users are encouraged to publish their services, workflows, simulation environments and entities, or improved the ones published by others. As more people publish their work, a critical mass can be created collaboratively.

Table 8 is comparison between the extended framework and MSRS from two different perspectives.

Table 8 Overview of Extended Framework and MSRS

	MSRS	Extended framework
Level of service oriented	Currently, it supports mainly service publication and discovery.	In addition to service publication and discovery, it also allows other entities such as simulation entities and environments to be published and discovered.
Number of services	152 services come with the installation and extra services need to be added manually.	Service repository is centralized; More services will be available as service repository is maintained in a collaborative manner.
Ontology support	No explicit ontology support	Explicit ontology support
Collaboration support	Less collaboration support	More collaboration support
Adaptability for changes	Changes might result in unavailability of simulation tasks.	Cross-referencing and ontology support reduced the probability of unavailability caused by the change of assets.
Platform dependency	Platform-dependent	Both platform-independent and platform-dependent support

5.5.1 Simulation Framework

The proposed simulation framework has the following major services or facilities:

- **Simulation Engine:** This manages the simulation time and keeps track of all the entities' states in the entire simulation world. It gets and executes the application workflows from the workflow repository.
- **Visualization Services:** These retrieve the states from the simulation engine and display or visualize them. A user can directly view the simulation results and because they are independent services, different rendering services, such as 2D or 3D, can be used.
- **Entity Services:** These represent the simulated objects in the world. An entity service can either be a robot or an object in the environment. Each service contains all the

information needed by the simulation engine and visualization service, such as shape, mass, position, and orientation. Some properties can be omitted for certain simulation tasks. A user can use the interfaces provided by the service to inquire the state of an entity, or update it.

- **Environment Services:** These represent the simulated worlds. Each environment service is composed by entity services, and contains all the information needed by the simulation engine and visualization service.

- **Function services:** These implement simulated basic actuator and sensor services, such as differential drive, light sensor, and bumper, or complex services composed from the basic services. Each service implements a service specification, which specifies the tasks provided.

- **Service Repository:** This is a repository for users to publish and share their services. As more services are accumulated in the repository, the application composition will reuse more assets. For the same service specification, there can be multiple implementations, so the user can compare them and choose the one that best suits their simulation needs.

- **Workflow Repository:** This is a repository for users to share their simulation workflows. With the support of CCSOA [18], not only services, but also workflows can be published and shared. As more workflows are accumulated in the repository, users do not have to compose application from scratch. One can first search the workflow repository, find a workflow that they are interested and start from there. If no matching workflow is found, one can compose his own and publish it in the repository for future use.

- **Entity Repository:** This is a repository for users to publish and share simulated entities. Suppose a user did not find a simulated environment needed in the environment repository, he does not need to start from scratch. The user can search in the entity repository, and use the existing entities to compose the simulated world. In addition, one can also bind the services used in a workflow with the simulated objects (mostly robots for robotic simulation) found in the repository.
- **Environment Repository:** This is a repository for users to publish and share simulated environments and use them to compose new simulation tasks.

5.5.2 Collaborative Simulation

In the environment, the users can be classified into different groups according to their concentrations or skills:

- **Service Designers**: They are specialists of building services that control the robots or implement a particular functional component.
- **Workflow Designers**: They are specialists of providing various workflows that implement different strategies or algorithms.
- **Simulation Entity Designers**: They are specialists of designing visualized simulated robots and other objects. They also have to provide information of the simulated robots, such as how many wheels they have, the diameter of the wheels, or what additional functions it can support.
- **Simulation Environment Designers**: They are specialists of building simulated environments in which a simulation is executed.

Different specialists can work closely under this collaborative framework to accomplish a simulation task. For example, a workflow designer may publish his requests for desired services, and service designers will implement the services according to the service descriptions. To build an environment, environment designers may need some particular simulation entities from entity designers. The ontology systems provided in the framework will facilitate the collaboration among the different specialists, as requests are well classified. Figure 35 shows the collaborative simulation with four groups of simulation specialists.

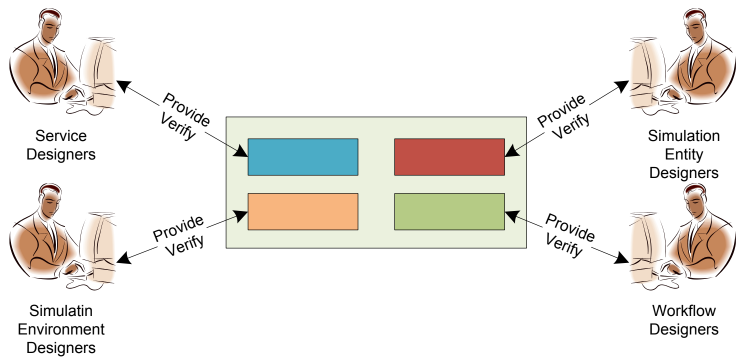


Figure 35 Collaborative Simulation

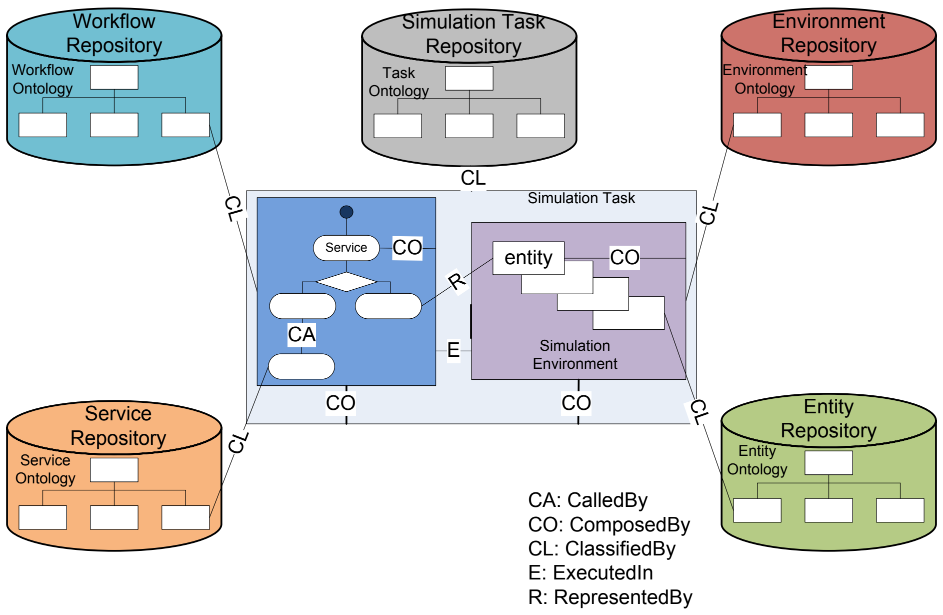


Figure 36 Relationships Among Assets

Table 9 How the Extended Framework Adapt to Changes

Asset	Change	Description
Services	Add	When a service is added, it is put in the repository, and **classified by** the service ontology. And workflow **composed by** this type of service, services **called by** this type of services, entities that this type of services was **represented by** will be notified that a new service is available.
	Remove	When a service is about to be removed, the framework will issue a warning that the assets that reference this service, for example, the workflow **composed by** this service; the entities that the service was **represented by**. If the user still wants to remove the service, the assets that reference this service will be notified that this service is no longer available.
	Modify	When a service is about to be modified, the framework will issue a warning that the assets that reference this service (see **add** and **remove** operation) may become unavailable. If the modification is major, such as interface change, the assets that reference this service will be notified that this service has been modified and according change should be made to maintain the availability.

Workflows	Add	When a workflow is added, it is put in the repository, and **classified by** the workflow ontology. And simulation tasks that were **composed by** workflows under this category will be notified that a new workflow becomes available.
	Remove	When a workflow is removed, the framework will issue a warning that the simulation tasks **composed by** the workflow may become unavailable. If the user still wants to remove the workflow, the simulation tasks that reference this workflow will be notified that this workflow is no longer available.
	Modify	Combination of **remove** and **add**.
Simulation Entities	Add	When a simulation entity is added, it is put into the entity repository, and **classified by** the entity ontology. And all the assets, such as services **represented by** this type of entities, simulation environments **composed by** this type of entities will be notified that a new entity is available.
	Remove	When an entity is about to be removed, the framework will issue a warning that the assets that reference this entity, such as services **represented by** this entity, environments **composed by** this entity) may become unavailable. If the user still wants to remove the entity, the assets that reference this service will be notified that this entity is no

		longer available, and entities under the same category will be suggested.
	Modify	Assets (see **add** and **remove**) that reference this entity will be notified that a modification has occurred to the entity, and they will have to check if the reference is still valid.
Simulation Environm ents	Add	When an environment is added, it is put in the repository, and **classified by** the environment ontology. And all the assets such as workflows **executed in** the environment, simulation tasks **composed by** the environment will be notified that a new environment is available.
	Remove	When an environment is about to be removed, the framework will issue a warning that the assets that reference this environment (e.g. workflow **executed in** this environment, simulation tasks **composed by** the environment) may become unavailable. If the user still wants to remove the environment the assets that reference this environment will be notified that this environment is no longer available.
	Modify	Combination of **remove** and **add**.

5.5.3 Change Management and Cross-Referencing among Simulation Assets

There are various relationships among the different assets. For example, a simulation task is **composed by** services, workflows, simulation entities and simulation environments; a workflow is **composed by** multiple services; a service is **called by** other services; a simulation environment is **composed by** simulation entities; all the assets can be **classified by** the according ontology systems; a service can be **represented by** a simulation entity; a workflow can be **executed in** a simulation environment, etc. The relationships can be illustrated in Figure 36.

The framework provides better adaptability to the changes of simulation tasks. A typical service-oriented robotic simulation task is composed by four major simulation assets: services, workflows, simulation entities, and simulation environments. Each type of simulation assets are published and stored in an according repository and classified based on the ontology system. By using standard SOA modeling, any asset can be easily changed and composed into the simulation task with the support of this framework. There are usually 3 types of changes: **add**, **remove**, and **modify**. Table 9 illustrates how this framework can lessen the impacts caused by the changes.

The framework will keep tracking what other assets a specific simulation asset is referenced by. Therefore, when there is a change for this asset, the referencing assets will be notified and make changes accordingly.

5.5.4 Platform-Independent Simulation

The framework can be further extended by separating the platform-dependent part and the platform-independent part as suggested by the OMG (Object Management Group at www.omg.org). In this way, the platform-independent part can be shared among all users, and various analyses can be performed on it. Once the mappings from the platform-independent part to platform-dependent part are available, a platform-independent system can be mapped to different platforms for execution. For example, a robotic application originally written for MSRS may be mapped to another platform, such as Intel robotic platform without changing the platform-independent part. The extended framework is shown in Figure 37.

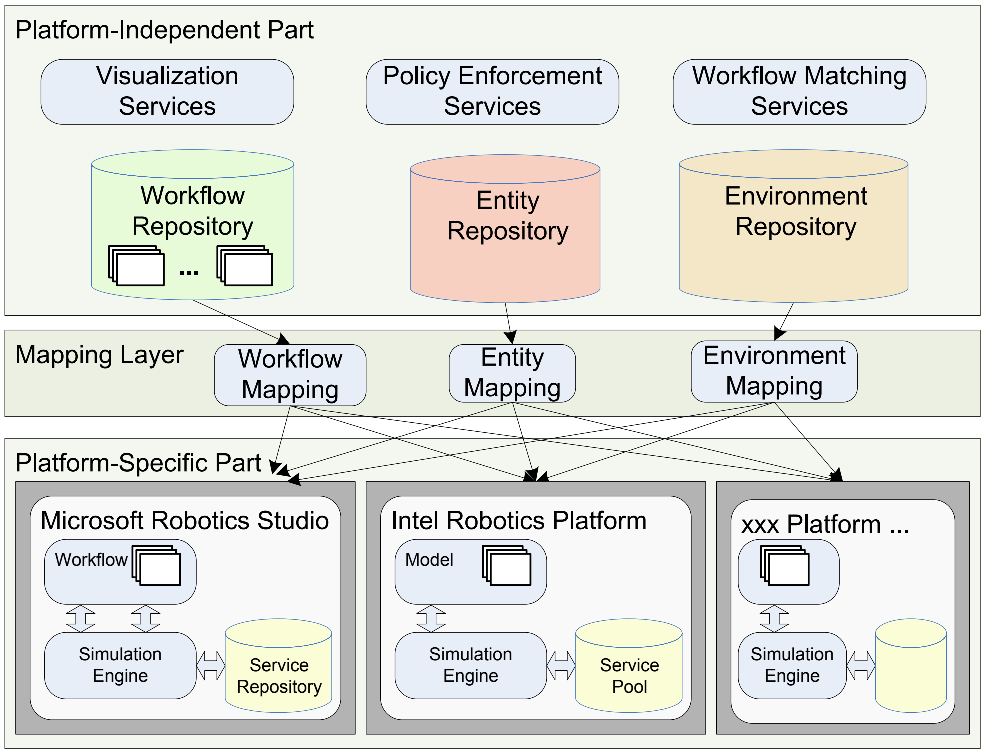


Figure 37 Extended Collaborative Simulation Framework

In this extended framework, one can use platform-independent modeling languages such as BPEL [99], or PSML-S [22] to build the workflow, entity and environment specifications and submit them to the public repository. There is a mapping layer to map these specifications to platform-specific specifications. The visualization service and collaboration matching service use standard interface to communicate with the platform-dependent part. This framework gives user the flexibility to choose different platforms for simulation while using uniformed modeling language to specify the system model. *Table 10* shows how PSML-S elements can be mapped to MSRS elements.

Table 10 PSML-S Element Mapping

Element Types	**Mapping to MSRS**
Actors	MSRS activities
Conditions	Condition statements
Data	Data
Actions	Actions in activities
Attributes	Attributes or constraints in services
Events	Notifications
Relations	Messages or constraints between entities

5.5.5 Service Ontology Modeling

Service Ontology (SO): This stores concepts and relations among services, and it also has classification trees with reasoning mechanism. In SO, both atomic and composite services can be presented. As shown in Figure 38, all the services are divided into three groups depending on their usages. In the example, when the sensor service is requested, the services registered under the sensor service will be returned. If no exact matching service is available, the system will return a closely matching service. For example, the system may return ultra-sound sensor service to replace the infrared sensor service by using the information in the SO. Furthermore, one can apply semantic distance to identify those related concepts [104].

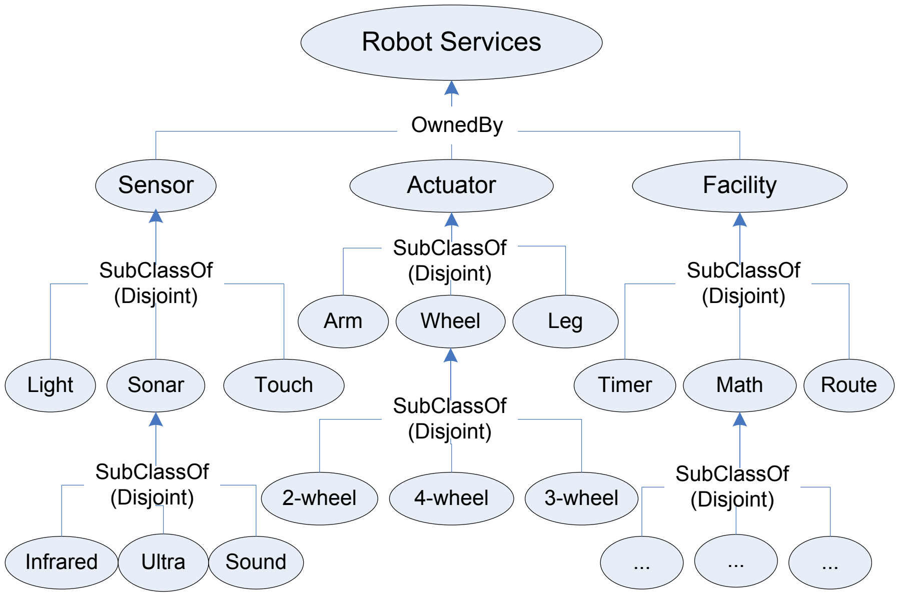


Figure 38 Rebot Services Ontology

Workflow Ontology (WO): This defines the concepts and the relations among workflows. For example, a goal keeper may have three different subclasses, defining three different kinds of behaviors, passive, active or patrolling as shown in Figure 39. Each different workflow may be used under different application contexts, and has different success rates. *Figure 39* shows a classification tree for workflow services.

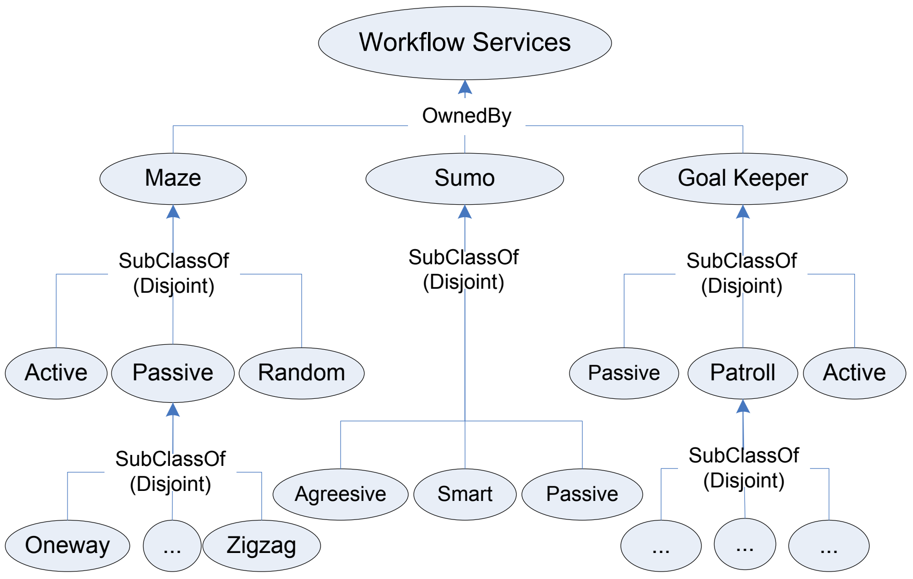


Figure 39 Workflow Ontology

Environment Ontology (EO): This specifies the concepts and relations of different simulation environments. For example, in Figure 40, there are several environment classifications, such as maze, geometry, and nature. Under each category, they are divided into subclasses according to different environment types, such as rocky environment, flat ground, and watery surface.

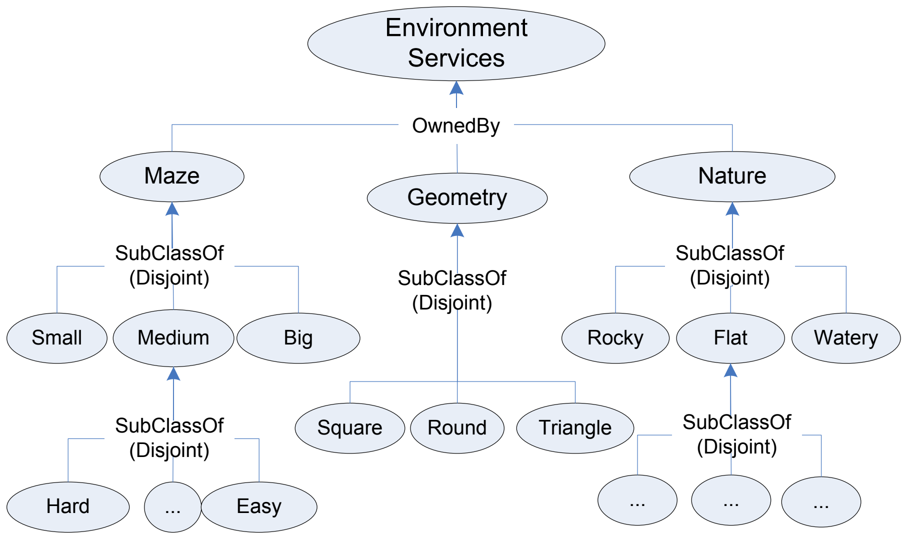


Figure 40 Environment Ontology

Entity Ontology (ETO): This specifies the concepts and relations of simulation entities. For example, in Figure 41, there are several major types of robots, such as car robot, and crawler robot. Under each category, they are divided into different providers, such as NXT, Lego, and IRobot. When an entity is registered in an entity registry, it will be placed under the corresponding concept. Once the application requires a specific entity service, the ontology can provide essential information to find the best candidate.

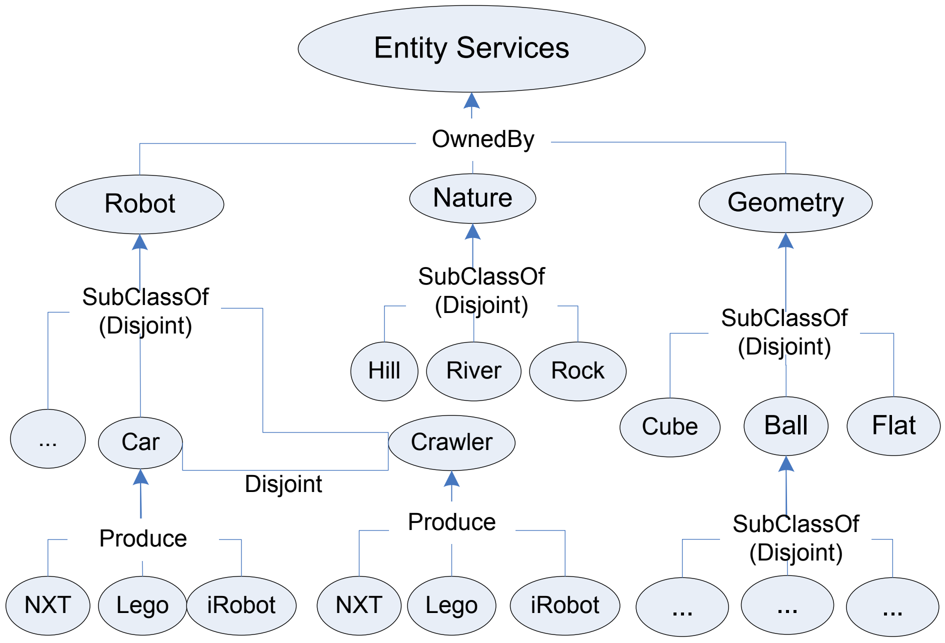


Figure 41 Entity Ontology

Part of the ontology axioms are listed as following in which A, B denote the concepts in the ontology:

1. $Disjoint(A,\ B) \leftrightarrow \neg\ Overlapping$

2. $(x \notin B \mid \forall x \in A) \leftrightarrow Disjoint(A,B)$

3. $Produce(A,\ B) \leftrightarrow ProducedBy(B,\ A)$

4. $OwnedBy(A, B) \Leftrightarrow OwnerOf(B, A)$

5. $Compose(A,B) \Leftrightarrow ComposedBy(B, A)$

According to the 2nd axiom, any car producer should not belong to crawler producer, however most robot companies can produce both models; the consistency analysis would find the conflict with the "Disjoint" relation between car producers and crawler producers. Thus the "Disjoint" relation between the car producer and crawler producer should be replaced by "Overlapping".

5.5.6 Simulation Application Composition

In SOA, not only services can be discovered, but also workflows can be discovered and even composed at runtime [38]. However, implementation of this kind of dynamic workflow discovery and composition requires a much sophisticated infrastructure. This chapter instead focuses on static workflow modeling and discovery.

Before running a robotic simulation task, one needs to know what workflows and which model of the robots, and what environment will be simulated. The user documents all these requirements into a simulation specification, and feeds it to the proposed simulation framework. The simulation specifications need to provide the following information for simulation.

- A set of workflows;
- A specific or a generic robot model;
- A specific or a generic class of environments the simulation will run in;
- A specific visualization service or a generic class of visualization services.

The simulation process can be described in the following steps:

1) Load workflow from Workflow Repository.

Given a simulation specification, the simulation framework first looks for a workflow by referencing the workflow ontology. The search is done by a matching service, which uses key word matching, ontology reasoning as its searching algorithm. One needs to choose the one that best matches the application requirements. If there is no such match found, one needs to use an independent modeling language, such as PSML, to model the workflow.

2) Matching platform-independent workflow to platform-specific language.

The workflow returned is written in platform-independent modeling language. To run the simulation on a specific platform, it needs to be mapped into a workflow written in a platform-dependent language. This is done by a mapping service discussed above.

3) Functional Service bounding using SO

Once the workflow is decided, more detailed configuration needs to be done. As discussed in the previous sections, a workflow is an application template that models the working process of an application. It is not composed by the actual implementations of services, but service stubs (service specifications). Therefore, to make the workflow working, each service stub needs to be bound with an actual service on platform. MSRS uses manifest to statically bind one generic service with a real service, so the binding is fixed. Unlike MSRS, the binding process under the new framework is dynamic and flexible. Each service stub used in the workflow has a specification (similar to the contract used in MSRS) required to be implemented by each of its implementations. All the specifications are classified by service ontology, and all the services are also classified according to the specification they implement. Suppose the service stub used in the workflow requires a service that implements the specification A, the matching service first checks if there is an implementation for specification A. If such implementations are not found, the matching service can use the ontology to check if the specification has any closely related ones, such as subclasses or siblings, and returns the closely related implementation that compatible with specification A. By doing this, the binding is no longer a literal key word matching, but a flexible reasoning process. This actually gives the workflow builders more flexibility because when the workflow builders compose the workflows, they may not know what services will be available when the workflow is actually executed. Therefore, they would rather just use a more abstract service specification and let the matching service decide which real services this specification will be bound to at the run time.

4) Entity service bounding using ETO

As to robotic simulation, each simulated service needs to be associated with a simulated entity that represents the service during the simulation process. This can also be done automatically under the new framework. For each service in the workflow, the matching service first searches for all the entities that can be used to represent the service. This could be done because the entities are categorized by the service specifications that they can be associated with. The matching service returns a set of entities for each service, and then it calculates the intersection of all the sets, returns it as the final result. Because each entity in the intersection can represent all the services used in the workflow, they all can be used in the simulation. As discussed earlier, each entity contains enough information to initialize the service it represents. Therefore, all the services are fully configured after this step.

5) Simulation environment bounding with EO

Similarly, by using the environment ontology, the matching service can return a list of environments that satisfy the users' requirement. Each environment has several extension points where users can choose to place the simulated robots.

6) Visualization service configuration

Similarly, by using the environment ontology, the matching service can return a list of environments that satisfy the users' requirement.

7) Simulation

After a visualization service is chosen, the simulation engine service starts to load the environment and simulated robots state, and run the simulation according to the workflow.

5.6 Case Study

This section illustrates the simulation framework using a case study.

5.6.1 Mission Description

SumoBot is a popular robotic competition. The main purpose of it is to have the robot stay inside of the arena while trying to push the opponent out. Suppose one wants to simulate a SumoBot competition. One needs to give a simulation configuration shown in Figure 42. One does not specify the workflow strategy at first (such as "Aggressive") as shown in the workflow ontology to show the flexibility of the framework.

```
<!--Simulation Configuration-->
<Strategy>SumoBot Competition</Strategy>
<Robots>
            <Robot1>SumoBot</Robot1>
            <Robot2>SumoBot</Robot2>
</Robots>
<Environment>SumoBot arena</Environment>
<Visualization>3D</Visualization>
```

Figure 42 Sample Simulation Specification

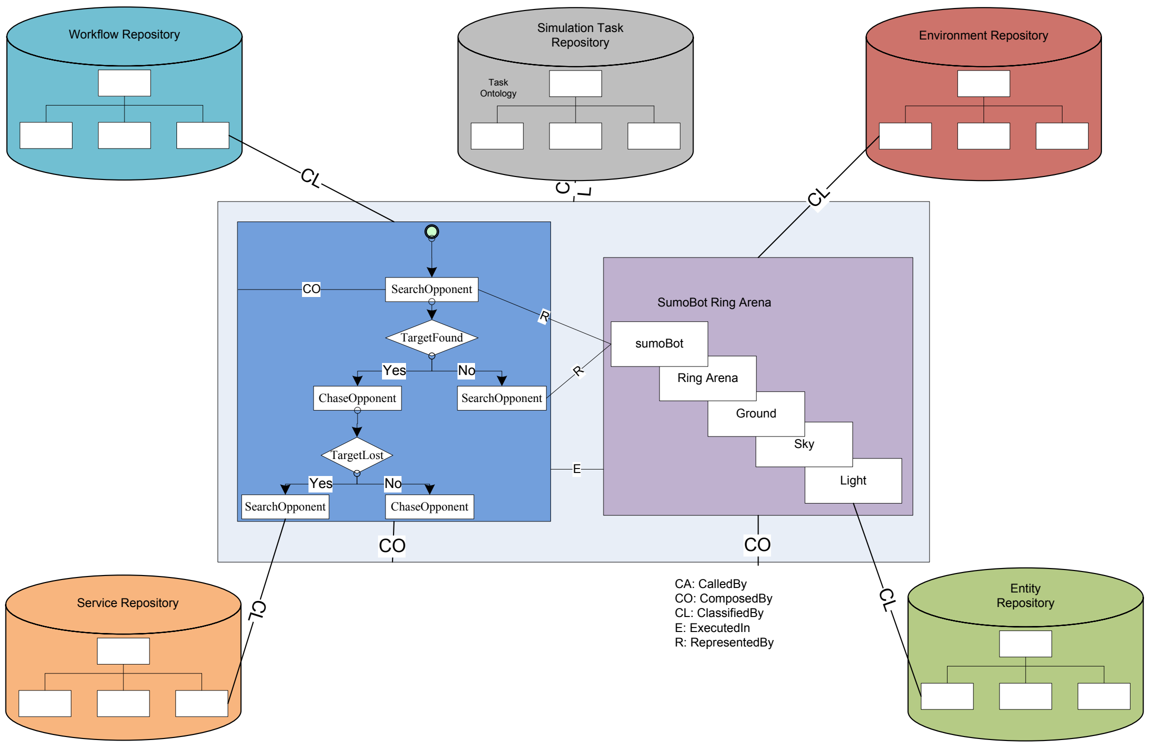


Figure 43 Relationships Among Assets

5.6.2 Relations among Simulation Assets

As shown in Figure 43, the simulation task has four major components: services, workflows, simulation entities and simulation environments.

Serivces:

- SearchOpponent Service Implementation.
- ChaseOpponent Service Implementation.

Workflow:

- Aggressive-SearchChase workflow

Entities:

- Simulated IRobot Roomba SumoBot
- SumoBot ring-shape arena
- Sky
- Ground
- Light

Environment:

- Simulated SumoBot ring-shape arena environment

Relations among the simulation assets:

In this sample, the workflow is **composed by** two service implementations of SearchOpponent, ChaseOpponent. The simulation environment is **composed by** Simulated IRobot Roomba SumoBot, SumoBot ring-shape arena, sky, ground, light. The workflow is **executed in** the simulation environment. The two services are **represented by** simulated IRobot Roomba SumoBot. The workflow is **classified by** the workflow ontology under the Workflow-Sumo-Aggressive-SearchChase category; the environment is **classified by** the environment ontology under the Environment-Sumo-Ring-Small category. Service implementation of SearchOpponent is **classified by** the service ontology under Service-Complex Service-Sumo-SearchOpponent category; Service implementation of ChaseOpponent is **classified by** the service ontology under Service-Complex Service-Sumo-ChaseOpponent. The simulated IRobot Roomba SumoBot is **classified by** the entity ontology under Entity-Robot-Sumo-IRobot; the ring-shape SumoBot arena is **classified by** the entity ontology under Entity-Geometry-Sumo-Arena-Ring; Sky, ground and light are **classified by** the entity ontology under Entity-Nature.

With these relations stored and monitored, a cross reference is established. Empowered by this scheme, one can predict the effects of the possible changes on all the simulations assets. For example, at a later time, one wants to delete the simulated SumoBot ring-shape arena environment. All the references to this environment including the one shown in this example will be returned to the user, and a warning will be issued to the user to let him/her know a deletion of this environment will result in unavailability of the assets that refer to it.

5.6.3 Application Modeling

This simulation uses the repositories presented in section 5.4.1 including Workflow Repository, Service Repository, Entity Repository and Environment Repository. The services stored in these repositories are classified using the ontology systems presented in section 5.4.3.

5.6.4 Workflow Repository

Figure 44 shows how workflows are classified in the workflow ontology. The workflows that are not related to SumoBot competitions are omitted.

Descriptions for each workflow are given in Table 11. Different workflows will produce different robotic behaviors.

Table 11 Workflows for SumoBot Competition

Workflow	Description
Aggressive-ChaseAlways	The robot will just speed up, and hope to hit the opponent by luck.
Aggressive-SearchChase	The robot will actively search for its opponent; Once it locates the opponent, it will chase after it, trying to push it off the arena
Smart	The robot will switch back and forth between the Aggressive and Passive modes before it locates its opponent.
Passive-WaitChase	The robot will passively wait until being attacked by its opponent, then fight back.
Passive-WaitLure	The robot will passively wait until being attacked by its opponent, then run away from its opponent, trying to lure it to the edge, and then suddenly turn at the edge and start attacking.

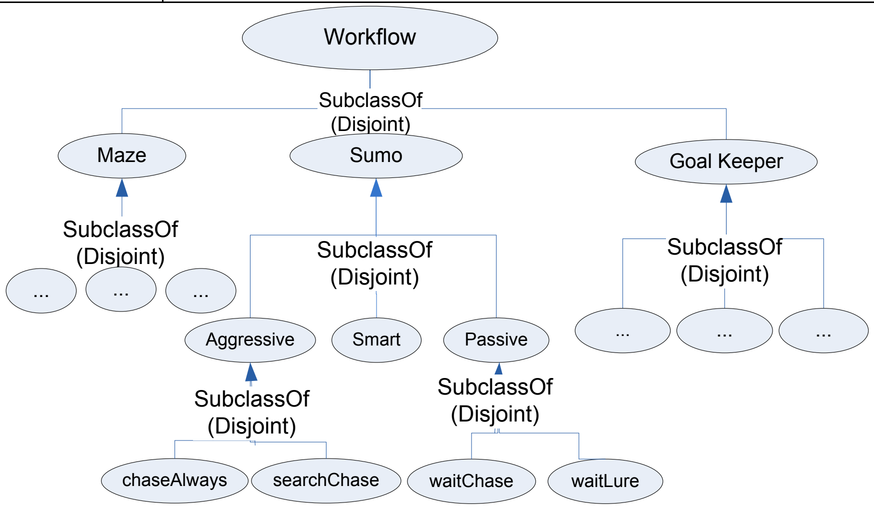


Figure 44 Sample Workflow Ontology

5.6.5 Service Repository

Figure 45 shows how services are classified in the service ontology. The services that are not related to SumoBot competitions are omitted.

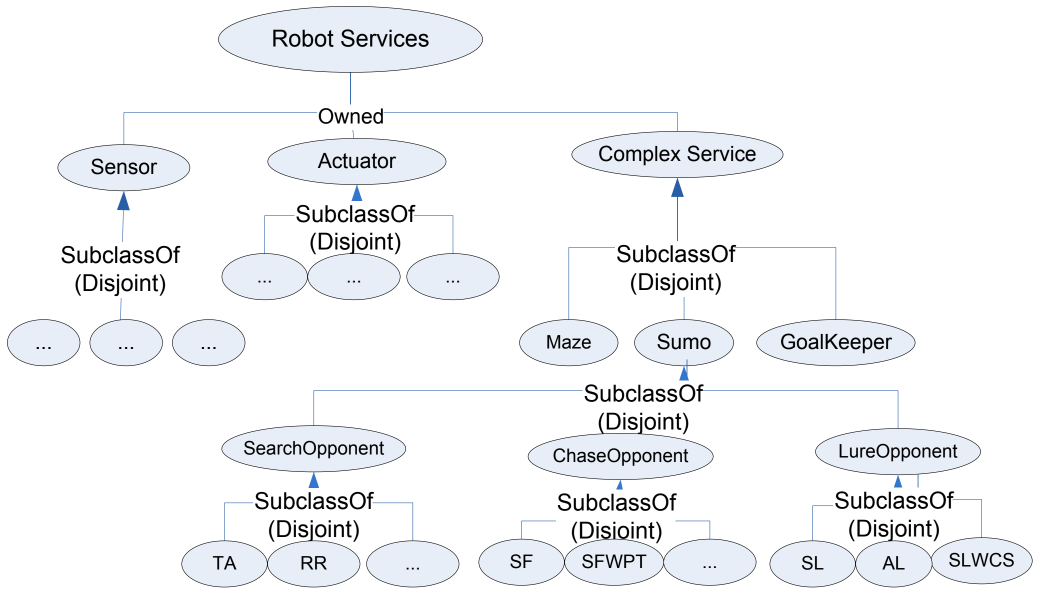


Figure 45 Sample Service Ontology

SearchOpponent: This is a group of services that implement the different search strategies listed in Table 12. In a SumoBot competition, the robot that can find its opponent first will have a big advantage. In many times the losing robots were pushed out of field from side or back before seeing the enemy.

Table 12 Service Implementation for Searching Opponents

Search Strategy	Description
Turning Around (TA)	Have the robot sitting still at a fixed place and turning around and around forever, expecting to see the opponent in this way.
Randomly Roaming (RR)	Have the robot roaming over the combat arena randomly. Such as go forward for a random distance, and turn a random angle and continue this loop.
Randomly Roaming and Turning Around (RRTA)	Have the robot roaming over the combat field randomly and turning around and around now and then.
Border Patrol (BP)	Patrol along the border of the field and look inside.
Square Patrol (SQP)	Starting from a point by the border, patrol on a square path.
Zigzag Patrol (ZP)	Start from anywhere, patrol on a zigzag route.
Spiral Patrol (SPP)	Starting from the center of the arena, patrol on a spiral route.

ChaseOpponent: This is a group of services that implement the different chasing strategies. They calculate the distance between the robot and its opponent, and change the speed accordingly. Such as when the opponent is detected but still in a distance, the robot needs to keep speed normal while employing fast turning algorithm, when it begins approaching the opponent and distance is short enough, it may speed up, using the speed momentum to push its opponent out of the field.

Chasing Algorithm	Description
Simple Follow (SF)	Follow the target passively. Start searching when target lost.
Simple Follow with Position Tracking (SFWPT)	Follow the target passively. Turn for a fixed angle to the direction when it last saw the opponent.
Simple Follow with Active Turning (SFWAT)	Active turning by calculate proper turning angle.
Active Turning and Full Speed Approaching (ATFSA)	Active turning, chasing the opponent at full speed once on target.
Active Turning and Active Approaching (ATAA)	Approaching the opponent on normal speed, increase speed when distance is close enough.

lists the services in the repository under this category.

Table 13 Service Implementations for Chasing Opponents

Chasing Algorithm	Description
Simple Follow (SF)	Follow the target passively. Start searching when target lost.
Simple Follow with Position Tracking (SFWPT)	Follow the target passively. Turn for a fixed angle to the direction when it last saw the opponent.
Simple Follow with Active Turning (SFWAT)	Active turning by calculate proper turning angle.
Active Turning and Full Speed Approaching (ATFSA)	Active turning, chasing the opponent at full speed once on target.
Active Turning and Active Approaching (ATAA)	Approaching the opponent on normal speed, increase speed when distance is close enough.

LureOpponent: This is a group of services that implement the different luring strategies. The main purpose of the services is to lure its opponent to the edge of the arena, and trick it to fall off the arena. It might involve calculating the distance between the robot and its opponent, and changing the speed accordingly, so that it will keep its opponent close enough that it will keep following and far enough that it will not be able to attack. Table 14 lists the services in the repository under this category.

Table 14 Service Implementation for Luring Opponents

Luring Algorithm	Description
Simple Lure (SL)	Just simply run away, suddenly turn left when getting close to the edge.
Active Lure (AL)	Calculate the distance between the robot and its opponent, and change the speed accordingly.
Simple Lure with Changing Speed (SLWCS)	Change the speed when it is running away

5.6.6 Entity Repository

Figure 46 shows how simulated entities are classified under the entity ontology system. The entities that are not related to SumoBot competitions are omitted.

The entities listed in Figure 46 are the simulated entities that represent the corresponding real entities.

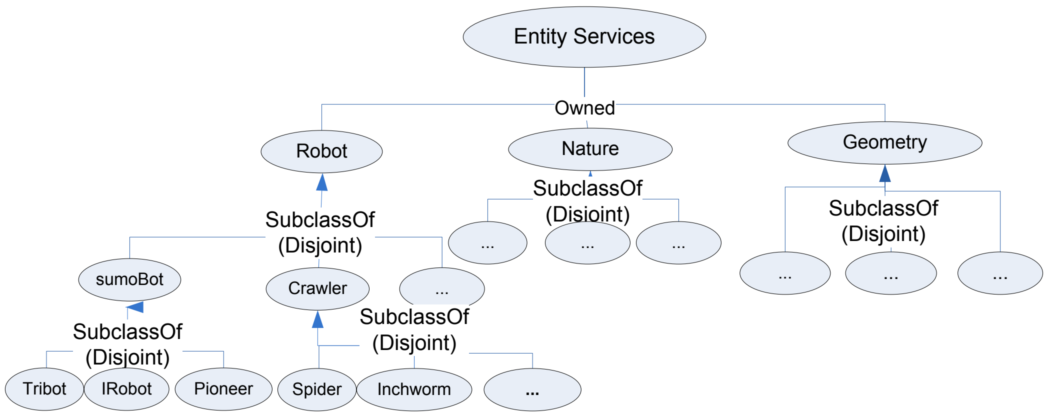


Figure 46 Sample Entity Ontology

Table 15 Simulated Entities

Entity	Description
Car-LegoNxt Tribot	It simulates the appearance of the LegoNxt robot, which is equipped with 3 wheels, a touch sensor, an ultra-sonic senor and a light sensor.
Car-Pioneer PDX3	It simulates the appearance of the Pioneer PDX3 robot, which is equipped with 3 wheels, a laser ranger finder, a front bumper and a rear bumper.

Car-IRobot Roomba	It simulates the appearance of the IRobot, which is equipped with 2 wheels, a series of light sensors and infrared sensors.
Crawler-LegoNxt Spider	It simulates the appearance of the LegoNxt Spider robot, which is equipped with 8 legs, a light sensor, a touch sensor and an ultra-sonic sensor
Crawler-Inchworm	It simulates the appearance of the Inchworm crawler robot, which is equipped with 4 legs, a light sensor, and an infrared sensor.

5.6.7 Environment Repository

Figure 47 shows how simulated environments are classified under the environment ontology system. The environments that are not related to SumoBot competitions are omitted.

The environments listed in Figure 47 are SumoBot arenas with different sizes and shapes.

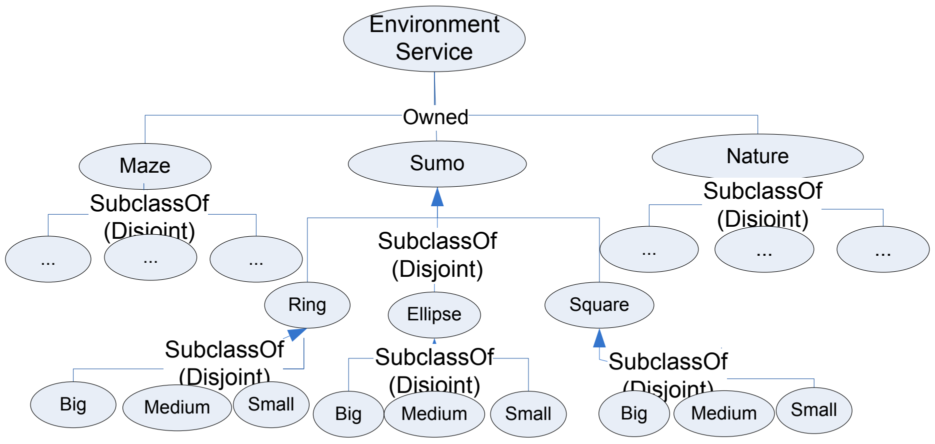


Figure 47 Sample Environment Ontology

Table 16 Simulation Environment for SumoBot Competition

Environment	Description
Ring-Big	This simulates a big size ring-shape SumoBot arena.
Ring-Medium	This simulates a medium size ring-shape SumoBot arena.
Ring-Small	This simulates a small size ring-shape SumoBot arena.

Ellipse-Big	This simulates a big size ellipse-shape SumoBot arena.
Ellipse-Medium	This simulates a medium size ellipse-shape SumoBot arena.
Ellipse-Small	This simulates a small size ellipse-shape SumoBot arena.
Square-Big	This simulates a big size square-shape SumoBot arena.
Square -Medium	This simulates a medium size square-shape SumoBot arena.
Square -Small	This simulates a small size square-shape SumoBot arena.

5.7 Simulation Execution

The simulation task can be executed in the following steps.

1) Load workflow from the workflow repository

When the simulation framework receives the simulation task request in the form of task specification, the matching service first searches in the workflow repository to find all the SumoBot strategy workflows based on the workflow ontology. If there is no workflow found, this process will stop and ask the user to compose a workflow and publish it in the workflow repository. In this case, the user has to model the workflow he wants to simulate using a platform-independent modeling language, such as PSML. Suppose the user wants to simulate the SumoBot algorithm in which the robot actively searches the opponent, and once it locates the opponent, it will chase after the opponent, trying to push it off the area. Table 17, 18, 19, 20 show how this modeling can be done.

The desired workflow will use a set of services, such as the service to search the opponent and the service to chase after the opponent. Table 17 illustrates the mapping from the SearchOpponent service to PSML model elements.

Table 17 Map Search Opponent Service into PSML Model Elements

SOA Systems	PSML Elements	PSML Element Description
SumoBot System	Actor	This is the SumoBot system
Search Opponent Service	Actor	SearchOpponentService
Functions		
search	Action	Search the opponent
Parameters		
Speed	Data	The spiral speed of the wheel
LeftWheelInfo	Actor	The left wheel information
RightWheelInfo	Actor	The right wheel information
LaserRangeFinderInfo	Actor	The laser range finder information
Properties		
ServiceNamespace	Attribute	Namespaces
ServiceURL	Attribute	URL Attributes

Table 18 Map Search Opponent Service into PSML Structure Element

PSML Elements	Relations	PSML Elements
Search Opponent Service	Own	**All elements, except itself**
Actions		**Data or Actors**
Chase	Use	LeftWheelInfo, RightWheelInfo, LaserRangeFinderInfo
Actors		**Data**
LeftWheelInfo	Own	Speed, Diameter
RightWheelInfo	Own	Speed, Diameter
LaserRangeFinderInfo	Own	Degree, Distance

Table 19 Map SOA Software Architecture by PSML

Architecture	PSML Model Elements	Example
Service Connection Architecture		
Inquiring Services	Actors	Search Service
Response Services	Actors	Chase Service
Service Connections	ServiceConnection Element	Search Service Connect Chase Service
Service Connection Attributes	Attributes	ConnectedPrecondition
Interface Connection Architecture		
Inquiring Activities	Events	StartSearch
Response Activities	Actions	StartChase
Interface Connections	InterfaceConnection Element	StartDistrict Connect StartChase
Interface Connection Attributes	Attributes	TriggeredPrecondition
Parameters	Data	Signal

Table 20 Model the Architecture of the Sample Application

PSML Elements	Relations	PSML Elements
Inquiring Service (Search Service)	Own	Event (StartSearch)
Response Service (Chase Service)	Own	Action (StartChase)
Owner System (SumoBot System)	Own	Inquiring Service (Search Service), Response Service (Chase Service), Service Connection (Search service Connect Chase service), Interface Connection (StartSearch connect StartChase)
Inquiring Activity (StartSearch)	Use	Parameters (Signal)
Response Activity (StartChase)	Use	Parameters (Signal)
Service Connection (Search service connect chase service)	Own	Service Connection Attributes (ConnectedPrecondition)
Interface Connection (StartSearch connect StartChase)	Own	Interface Connection Attributes (TriggeredPrecondition)

The identified model elements in the example have relations defined in Table 17.

SOA systems are a set of services communicated through messages and collaborated to fulfill specific missions. The software architecture of a system consists of software components, their external properties, and their relationships. For the example, in the example system, the Search service will search the component. When the opponent is located, the Chase service will be triggered. The architecture of this example can be modeled by following elements as shown in **Table 19**.

In this example, the connected precondition can be the “Opponent is detected”, and the triggered precondition is “Opponent is within a threshold”. Therefore, the event elements attach the signal as parameter, and the action elements take it as the input parameter. Among these elements, they have following relations defined in the structure model, as shown in **Table 18**.

The process of modeling of a workflow in PSML is mainly just in a drag and drop manner. The PSML models including the associated structural and behavior model are stored in XML format and can be visualized.

As the repository grows, many existing workflows will be available for reuse. Assuming the discovering process is successful, several maze traversing workflows are returned. For example, one workflow may be the Passive-WaitChase and the other is Aggressive-SearchChase. The user needs to decide which workflow to use. Figure 48 shows how the Aggressive-SearchChase algorithm workflow is visualized in PSML.

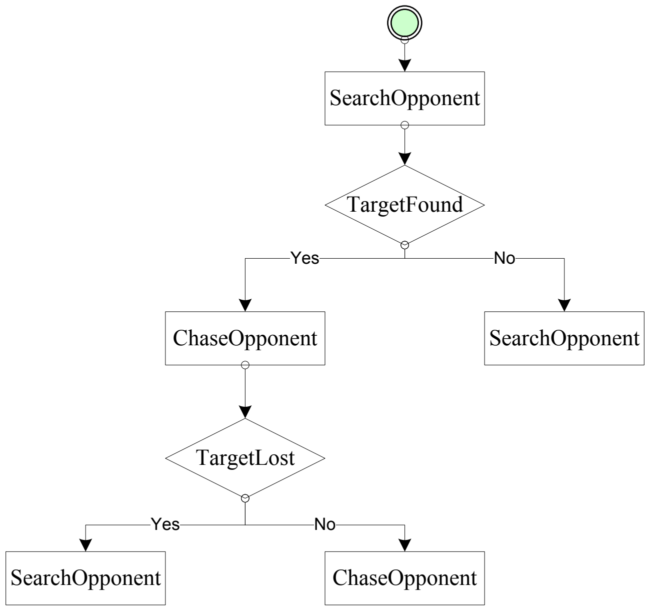


Figure 48 PSML Model for Aggressive SumoBot Chase Algorithm

2) Matching the platform-independent workflow to platform-specific language.

The Aggressive-SearchChase workflow needs to be mapped to MSRS. Specifically, the mapping layer of the framework will generate a program coded in VPL as shown in **Figure 49**

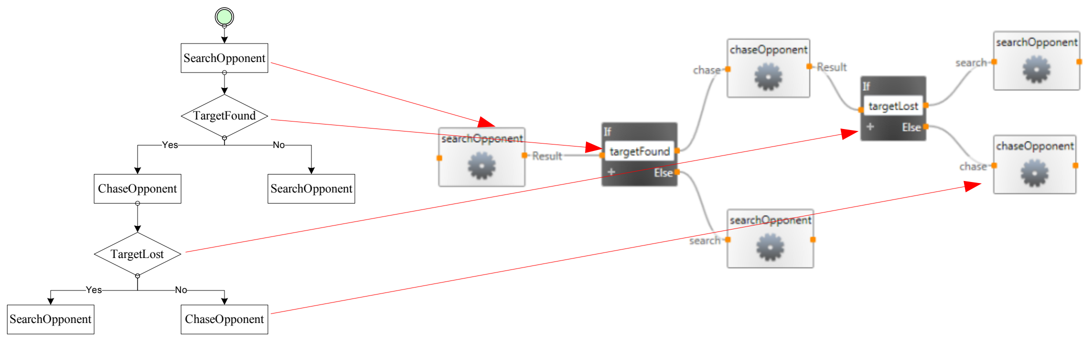


Figure 49 Workflow Mapping Example

3) Discovering services using service ontology

After the mapping, it finds out that the workflow contains two service specifications, one is SearchOpponet service, and the other is ChaseOpponent service. The matching service will search in the service repository using the service ontology. There are multiple implementations for the service specifications. The user is prompted to choose one to run the simulation. Users can choose different implementations in different runs of the simulation, and compare which implementation gives the most desired result.

4) Entity service bounding using ETO

After the two service implementations are found, the system finds out that the two service implementations need to be run on simulated entities that support two-wheel service and sonar sensor service. The matching service searches for the possible entities that can represent these two service stubs. For the two-wheel drive service stub, the results returned are {LegoNxt Tribot, IRobot Roomba, Pioneer PDX3}; for the sonar sensor service stub, the results returned are {IRobot Roomba, Pioneer PDX3}. The intersection of the two sets is {IRobot Roomba, Pioneer PDX3} and both robots in the intersection are of “SumoBot” type entities, so they are used. The user will be prompted to choose one. Assuming the user chooses IRobot, the two services are then configured according to the IRobot Roomba, such the wheels' size, and the distance between the two wheels, and the measurement limit of the sonar.

Because a SumoBot competition involves two simulated robots, process 1 to 4 will be executed again for the second robot.

5) Select an environment using environment ontology

After the two services are fully configured, the matching service searches the environments that meet the user's requirement, e.g., a small arena, in the environment repository. Several small arenas with different shapes are returned. The users will be prompted to choose one from them. Assuming the user chooses the ring-shape arena.

6) Visualization service configuration

Similarly, a 3D visualization service can be chosen.

7) Simulation execution

After all the configurations have been set up, the simulation engine service loads the SumoBot arena and the two simulated robots, and then starts running simulation. The two simulated robots will behave according to the service implementations and workflow chosen for them. The simulation result is shown in Figure 50. One can compare the performance difference among the different service implementations by running the simulation multiple times using different service implementation for the two robots.

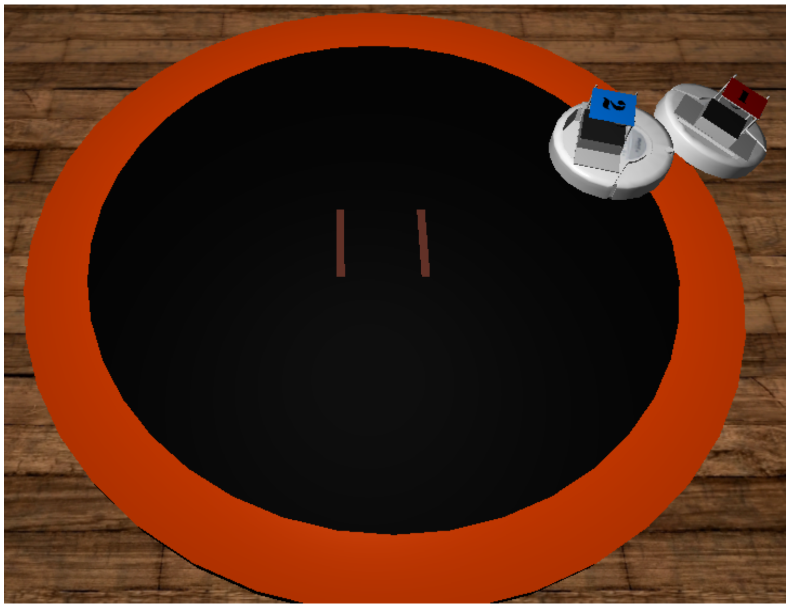


Figure 50 Simulation Snapshot: Player Lost

To illustrate the benefits of the proposed fully service-oriented simulation environment, the same simulation task is run again. However, this time a different arena is chosen as shown in Figure 51. Notice that the user does not need to create a new configuration file or a new initial state file from the simulated world to run this new simulation. This scheme also opens the possibility of running simulation tasks in batch. The framework will run the simulation using different simulated entities in different environments. With a monitoring service added, one does not need to monitor the simulated robots' behavior all the time. The simulation framework will run the simulations sequentially, and the results can be recorded by the monitoring service.

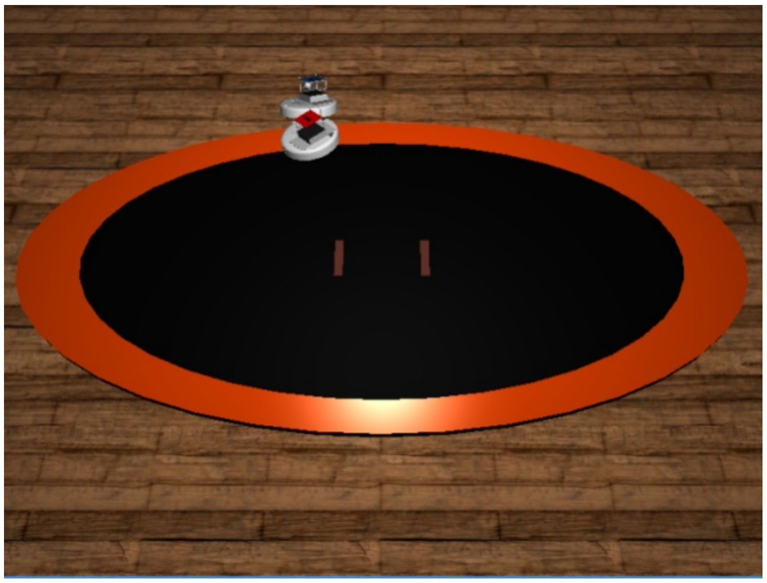


Figure 51 Simulation Snapshot: Ellipse Arena Simulation Results

5.8 Simulation Results

The workflows for different strategies configured with different service implementations were run using the simulation techniques described in the previous sections. The simulation results are given in the follows.

- Searching services

Two simulated robots, robot A and robot B, are placed in a ring-shape arena. Robot A is configured with searching service implementation, and Robot B will just randomly wander around the arena. The time used for robot A to locate robot B and the distance between them when robot A sees robot B were recorded. The simulation for each service implementation was run 10 times to get the average for the time and distance. The result is shown in Table 21.

Table 21 Simulation Results for Searching Service

Search Strategy	**Time(sec)**	**Distance (The diameter of the simulated IRobot is 1 distance unit; the radius of the area is 20)**
Turning Around (TA)	3	15
Randomly Roaming (RR)	6	12
Randomly Roaming and Turning Around (RRTA)	8	14
Border Patrol (BP)	6	8
Square Patrol (SQP)	4	6
Zigzag Patrol (ZP)	3	11
Spiral Patrol (SPP)	3	7

Because one wants to find the opponent as soon as possible, and the distance should not be too far as when it is accelerating, the opponent may locate you as well, or too short because it will not get enough speed. Therefore, the best searching service is zigzag patrol strategy.

- Chasing services

Two simulated robots, robot A and robot B, are placed in a ring-shape arena. Robot A is configured with chasing service implementation, and Robot B will just randomly wander around the arena. The simulation for each service implementation was run 10 times. The times that Robot A lost the target are recorded. The result is shown in Table 22.

Table 22 Simulation Results for Chasing Results

Chasing Algorithm
Simple Follow (SF)
Simple Follow with Position Tracking (SFWPT)
Simple Follow with Active Turning (SFWAT)
Active Turning and Full Speed Approaching (ATFSA)
Active Turning and Active Approaching (ATAA)

As the result shows Simple Fellow with Active Turning (SFWAT) and Active Turning and Active Approaching (ATAA) have the fewest times of losing the target. Simple follow has the worst performance.

- Luring service:

Two simulated robots, robot A and robot B, are placed in a ring-shape arena. Robot A is configured with Simple Following chasing service implementation, and Robot B is configured with various luring service implementations. The simulation for each service implementation was run 10 times. Times that Robot B won and lost were recorded. The result is shown in Table 23.

Table 23 Simulation Results for Luring Services

Luring Algorithm
Simple Lure (SL)
Active Lure (AL)
Simple Lure with Changing Speed (SLWCS)

As the result clearly shows, the Active Lure strategy has the most winning times, and fewest times of losing the target and fewest time of losing the game. Therefore it is the best luring strategy.

- Workflows

Here we only compare the performance of two selected workflows: Aggressive-SearchChase (ASC) and Passive-WaitLure (PWL). Two simulated robots, robot A and robot B, are placed in a ring-shape arena. Robot A is using ASC, and Robot B is using PWL. ASC is configured with the most effective searching and chasing services from the previous results, and PWL is configured with the most effective luring service. The simulation was run 10 times. The times of each robot won the game were recorded and shown in Table 24. Figure 52 shows the winning rate.

Table 24 Simulation Results for Workflows

Workflow	Winning Times
Aggressive-SearchChase (ASC)	7
Passive-WaitLure (PWL)	3

As the result shows, ASC won more times than PWL, so it is a more effective SumoBot strategy.

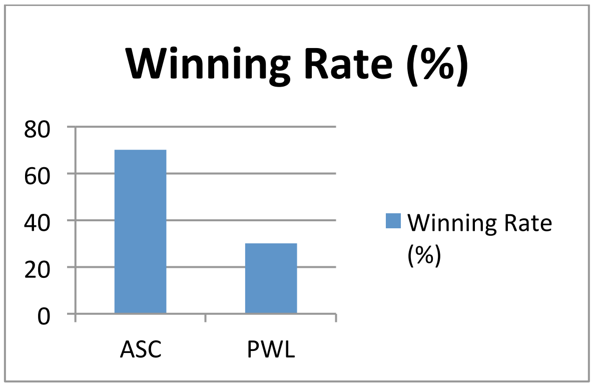


Figure 52 Winning Rate

5.9 Conclusion

SOA simulation is an important verification approach and it can be applied to every phrase of the application lifecycle and different domains including embedded system development. Simulation is particularly useful because SOA may involve dynamic discovery and composition, and simulation may be one of few techniques available to address these issues. This chapter proposes an ontology-based collaborative SOA simulation framework with MSRS for service-oriented embedded system development.

The framework extends the MSRS simulation runtime and provides flexibility to the simulation process. Ontology systems for services, workflows, simulated entities and simulated environments are introduced into the framework to facilitate the matching process. The framework is further extended to separate the platform-dependent and platform-independent part, thus let people use general modeling languages like PSML-S or BPEL to model the SOA application, and map to different SOA embedded system development platforms to promote reusability. Another advantage of using a general model is that one can apply various kinds of analyses without creating different models, thus avoid the incompleteness and inconsistency between models. Services, workflow templates, simulation entities, simulation environments and ontology are published as shared assets from different contributors, and assets can be accumulated over time making the development a much faster process. The simulation process is illustrated using a case study. The distinct feature of the proposed work is that SOA simulation is fully integrated with various ontology systems. While ontology is often used in SOA, it is used mainly for publishing and discovery of services at the present time. This chapter extends these capabilities so that ontology is useful in the context of SOA software development and simulation including publishing of various simulation entities, workflows and even application templates. Furthermore, this approach is also ompatible with model-driven approach commonly used in SOA software development and simulation.

BIBLIOGRAPHY

[1] Rajkumar Buyya and Chee Shin Yeo, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," Future Generation Computer Systems, pp. 599-616, 2009.

[2] Leonard Kleinrock, "A vision for the Internet," ST Journal of Research, vol. 2, no. 1, pp. 4-5, Nov. 2005.

[3] Wikipedia. Cloud Computing. [Online]. http://en.wikipedia.org/wiki/Cloud_computing

[4] Parascale. [Online]. http://www.parascale.com/

[5] Elastra. [Online]. http://www.elastra.com/

[6] Appirio. [Online]. http://www.appirio.com/

[7] Rochwerger B et al., "The RESERVOIR Model and Architecture for," IBM Systems Journal, 2009.

[8] Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya, "Aneka: A Software Platform for.NET-based Cloud Computing," in High Speed and Large Scale Scientific Computing, 2010.

[9] Ying Huang et al., "A Framework for Building a Low Cost, Scalable and Secured Platform for Web-Delivered Business Services," , 2009.

[10] Bernard Golden. (2009, January) Computer World. [Online]. http://www.computerworld.com/s/article/9126620/The_case_against_cloud_computing_part_one

[11] Mark Kyrnin. About.com. [Online]. http://compreviews.about.com/od/general/a/BuildvsBuy.htm

[12] Galen Moore. (2009, Jan) Mass High Tech. [Online]. http://www.masshightech.com/stories/2009/01/05/weekly10-Cloud-computings-SLA-obstacles-clear-in-2009.html

[13] Bo Gao, Changjie Guo, Zhihu Wang, Wenhao An, and Wei Sun. (2009, March) IBM SOA and Web Services Technical Library. [Online]. http://www.ibm.com/developerworks/webservices/library/ws-multitenant/index.html

[14] Wei-Tek Tsai, Qian Huang, Jay Elston, and Yinong Chen, "Service-Oriented User Interface Modeling and Composition," in International Conference on e-Busines Enginerring, Xi'an, 2008, pp. 21-28.

[15] Eucalyptus System. [Online]. http://www.eucalyptus.com/

[16] Qihong Shao et al., "Ranking Mortgage Origination Applications using Customer, Product, Environment and Workflow Attributes.," in IEEE Proceedings of Congress on Services, 2009.

[17] Wei-Tek Tsai et al., "Forecasting Loan Volumes For the Mortgage Orgination Process," , under review.

[18] W.T. Tsai, Bingnan Xiao, Ray A Paul, and Yinong Chen, "Consumer-Centric Service-Oriented Architecture: A New Approach," in SEUS-WCCIA, 2006, pp. 175-180.

[19] Mark Chang, Jackson He, W.T. Tsai, Bingnan Xiao, and Yinong Chen, "UCSOA: User-Centric Service-Oriented Architecture," in IEEE International Conference on e-Business Engineering, 2006, pp. 248-255.

[20] Wei-Tek Tsai, Chun Fan, and Yinong Chen, "DDSOS: A Dynamic Distributed Service-Oriented Simulation Framework," in the 39th Annual Simulation Symposium (ANSS), Huntsille, AL, 2006, pp. 160-167.

[21] force.com. [Online]. http://developer.force.com/appengine

[22] W.T. Tsai et al., "Modeling and Simulation in Service-Oriented Software Development," Simulation, vol. 83, no. 1, pp. 7-32, 2007.

[23] Changjie Guo, Wei Sun, Ying Huang, Zhihu Wang, and Bo Gao, "A Framework for Native Multi-Tenancy Application Development and Management," in The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007), Tokey Japan, 2007, pp. 551-558.

[24] Wei-Tek Tsai, Qihong Shao, and Wu Li, "OIC: Ontology-based Intelligent Customization," , 2010.

[25] William N Robinson and Yi Ding, "A survey of customization support in agent-based business process simulation tools," ACM Transactions on Modeling and Computer Simulation, vol. 20, no. 3, Sep. 2010.

[26] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. (2006, June) Microsoft

MSDN. [Online]. http://msdn.microsoft.com/en-us/library/aa479086.aspx

[27] Ralph Mietzner and Frank Leymann, "Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors," in Services Computing, 2008.

[28] Ralph Mietzner, Andreas Metzger, Frank Leymann, and Klaus Pohl, "Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications," in ICSE Workshop on Principles of Engineering Service Oriented Systems, 2009, pp. 18-25.

[29] Kuo Zhang et al., "A Policy-Driven Approach for Software-as-Services Customization," in IEEE Int. Conf. on, and Enterprise, 2007.

[30] Workday. [Online]. www.workday.com

[31] Duncan Jones and Patrick Connaughton. (2011, June) selectica. [Online]. http://www.determine.com/uploads/resource-pdfs/Forrester-Wave-2011.pdf

[32] Wei-Tek Tsai, Bingnan Xiao, Ray Paul, Qian Huang, and Yinong Chen, "Global Software Enterprise: A New Software Constructing Architecture," in The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services, The 3rd IEEE International Conference on E-Commerce Technology, 2006.

[33] (2012, Oct.) Wikipedia Template Method Pattern. [Online]. http://en.wikipedia.org/wiki/Template_method_pattern

[34] Wikipedia, Is-a.

[35] (2012, Oct.) Wikipedia Web Ontology Language. [Online]. http://en.wikipedia.org/wiki/Web_Ontology_Language

[36] Wei-Tek Tsai, Qian Huang, Jay Elston, and Yinong Chen, "Service-Oriented User Interface Modeling and Composition," in ICEBE '08. IEEE International Conference on e-Business Engineering, 2008.

[37] OWL-S: Semantic Markup for Web Services. (2004, Nov.) W3C. [Online]. http://www.w3.org/Submission/OWL-S/

[38] Wei-Tek Tsai, Qian Huang, Jingjing Xu, Yinong Chen, and Ray Paul, "Ontology-based Dynamic Process Collaboration in Service-Oriented Architecture," in the Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, 2007, pp. 39-46.

[39] Wikipedia, Human Resource Management. [Online]. http://en.wikipedia.org/wiki/Human_resource_management

[40] SQL Server 2012. Feature Selection (Data Mining). [Online]. http://msdn.microsoft.com/en-us/library/ms175382.aspx

[41] (2014, July) nielsen. [Online]. http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html

[42] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," Mobile Networks and Applications, vol. 16, no. 3, pp. 270--284, 2011.

[43] Joseph A Paradiso and Thad Starner, "Energy scavenging for mobile and wireless electronics," Pervasive Computing, IEEE, vol. 4, no. 1, pp. 18--27, 2005.

[44] Sangtae Ha, Soumya Sen, Carlee Joe-Wong, Youngbin Im, and Mung Chiang, "Tube: time-dependent pricing for mobile data," ACM SIGCOMM Computer Communication Review, vol. 42, no. 4, pp. 247--258, 2012.

[45] Afkham Azeez et al., "Multi-tenant SOA middleware for cloud computing," in Cloud computing (cloud), 2010 ieee 3rd international conference on, 2010, pp. 458--465.

[46] Bo Li, Wei-Tek Tsai, Haiying Zhou, and Dech Zuo, "A SaaS-based software modeling for bank intermediary business," in Proceedings of the 4th International Conference on Multimedia Technology, Sydney, 2015, p. 75.

[47] Dapeng Chen, Qingzhong Li, and Lanju Kong, "Process Customization Framework in SaaS Applications," in Web Information System and Application Conference (WISA), vol. 10, 2013, pp. 471--474.

[48] Ren Xiaojun, Zheng]Yongqing, and Kong Lanju, "SaaS Template Evolution Model Based on Tenancy History," in Intelligent System Design and Engineering Applications (ISDEA), 2013, pp. 1242--1247.

[49] Wikipedia. [Online]. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[50] Apache. (2016, July) Apache Kafka. [Online]. http://kafka.apache.org/

[51] Apache. (2016, July) Apache Hadoop. [Online]. http://hadoop.apache.org/

[52] LinkedIn. (2016, July) Pinot. [Online]. https://github.com/linkedin/pinot

[53] Apache. (2016, July) Apache Samza. [Online]. http://samza.apache.org/

[54] Joseph A. Whitney, Adin Scannell, Philip Patchin, Stephen M. Rumble,Eyal de Lara, Michael Brudno, M. Satyanarayanan H. Andrés Lagar-Cavilla, "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing," EuroSys'09, p. 12, April 1–3, 2009.

[55] Jeanna Matthews, Tal Garfinkel, and Jeff Wheeler, "Virtual Machine Contracts for Datacenter and Cloud Computing Environments," in Proceedings of the 1st workshop on Automated control for datacenters and clouds, 2009, pp. 25-30.

[56] Daniel Nurmi et al., "The Eucalyptus Open-Source Cloud-Computing System," in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 124-131.

[57] J. E. Smith, "Characterizing Computer Performance with a Single Number," Communications of the ACM, vol. 31, no. 10, pp. 1202-1206, 1988.

[58] Kaivalya M. Dixit, "The SPEC Benchmarks ," Parallel Computing, vol. 17, no. 10-11, pp. 1195-1209, December 1991.

[59] Ricardo Jimenez-Peris, Marta Patino-Martinez, Bettina Kemme, Francisco Perez-Sorrosal, and Damian Serrano, "A System of Architectural Patterns for Scalable, Consistent and Highly Available Multi-Tier Service-Oriented Infrastructures," in Architectuing Dependable System VI.: Springer Berlin / Heidelberg, 2009, pp. 1-23.

[60] Nicholas R.May, "A Redundancy Protocol for Service-Oriented Architectures," in ICSOC 2008 International Workshops, Sydney, Australia, 2009, pp. 211-220.

[61] Lin Huo, Yi Fang, and Heping Hu, "Dynamic Service Replica on Distributed Data Mining Grid," in 2008 International Conference on Computer Science and Software Engineering , 2008, pp. 390-393.

[62] Zakaria Maamar, Quan Z. Sheng, and Djamal Benslimane, "Sustaining Web Services High-Availability Using Communities," in Third International Conference on Availability, Reliability and Security, 2008, pp. 934-841.

[63] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka, "What Service Replication Middleware Can Learn From Object Replication Middleware," in Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006) , 2006, pp. 18-23.

[64] LUkasz Juszczyk, Jaroslaw Lazowski, and Schahram Dustdar, "Web Service Discovery, Replication, and Synchronization in Ad-Hoc Networks," in Proceedings of the First International Conference on Availability, Reliability and Security, 2006, pp. 847-854.

[65] Jorge Salas, Francisco Perez-Sorrosal, Marta Patiño-Martínez, and Ricardo Jiménez-Peris, "WS-replication: a Framework for Highly Available Web Services," in Proceedings of the 15th international conference on World Wide Web, 2006, pp. 357-366.

[66] C. Engelmann, S. L. Scott, C. Leangsuksun, and X. He, "Transparent Symmetric Active/Active Replication for Service-Level High Availabilit," in Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007, pp. 755-760.

[67] Nuno Laranjeiro and Marco Viieira, "Towards Fault Tolerance in Web Services Compositions," in Proceedings of the 2007 workshop on Engineering fault tolerant systems, 2007.

[68] (2010, June) Oracle and BEA. [Online]. http://www.oracle.com/bea/index.html?CNT=index.htm&FP=/content/solutions/soa/

[69] (2010, Jun) Cisco Service-Oriented Network Architecture. [Online]. http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns477/white_paper_Cisco_sona_support_optimize_soa_web2_0_applications.html

[70] (2010, June) HP's Approach to Service-Oriented Architecture. [Online]. http://h71028.www7.hp.com/enterprise/w1/en/technologies/soa-overview.html

[71] (2010, June) IBM Service Oriented Architecture. [Online]. http://www-01.ibm.com/software/solutions/soa/

[72] (2010, June) Microsoft SOA & Business Process. [Online]. http://www.microsoft.com/soa/

[73] (2010, June) SAS Service-Oriented Architecture. [Online]. http://www.sap.com/platform/soa/index.epx

[74] Raymond A. Paul, "DoD Towards Software Services," in the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, Sedona, AZ, 2005, pp. 3-6.

[75] (2010, June) Microsoft Robotic Studio. [Online]. http://msdn.microsoft.com/en-

us/robotics/default.aspx

[76] IBM, "IBM SOA Foundation: Providing What You Need to Get Started with SOA, white paper," September 2005.

[77] IBM. (2010, June) WebSphere Training. [Online]. http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=page&c=a0000329

[78] Wei-Tek Tsai, "Service-Oriented System Engineering: A New Paradigm," in Proceedings of the IEEE International Workshop on Service-Oriented System Engineering (SOSE), 2005, pp. 3-8.

[79] (2010, June) Simple Object Access Protocol. [Online]. http://www.w3.org/TR/soap/

[80] Wikipedia. (2010, June) Infrastructure as a Service. [Online]. http://en.wikipedia.org/wiki/Infrastructure_as_a_Service

[81] Intel. (2010, June) Intel Service Oriented Infrastructure. [Online]. http://www.intel.com/technology/itj/2006/v10i4/2-service/3-framework.htm

[82] Wikipedia. (2010, June) Service Oriented Infrastrcuture. [Online]. http://en.wikipedia.org/wiki/Service_Oriented_Infrastructure

[83] Wei-Tek Tsai, Xinxin Liu, and Yinong Chen, "Distributed Policy Specification and Enforcement in Service-Oriented Business Systems," in Proceedings of the IEEE International Conference on e-Business Engineering, Beijing, 2005, pp. 10-17.

[84] Wei-Tek Tsai, Xinxin Liu, Yinong Chen, and Rayment A. Paul, "Simulation Verification and Validation by Dynamic Policy Enforcement," in Proceedings of the 38th annual Symposium on Simulation, San Diego, CA, 2005, pp. 91-98.

[85] Bingnan Xiao, Wei-Tek Tsai, Qian Huang, Yinong Chen, and Rayment A. Paul, "SOA collaboration modeling, analysis, and simulation in PSML-C," in Proceedings of the IEEE International Conference on e-Business Engineering, 2006, pp. 639-646.

[86] Rajkumar Buyya and Anthony Sulistio, "Service and Utility Oriented Distributed Computing Systems: Challenges and Opportunities for Modeling and Simulation Communities," in Proceedings of the 41st Annual Simulation Symposium, Ottawa, 2008, pp. 68-81.

[87] Wikipedia. (2010, June) Software as a Service. [Online]. http://en.wikipedia.org/wiki/Software_as_a_Service

[88] Brutzman Don, Michael Zyda, and J.Mark Pullen, "Extensible Modeling and Simulation Framework (XMSF) Challenges for Web-Based Modeling and Simulation," , 2002.

[89] (2003) XMSF SAIC Web-Enabled RTI. [Online]. XMSF SAIC Web-Enabled RTI

[90] Bohu Li et al., "Research on Service Oriented Simulation Grid," in the Proceedings of Autonomous Decentralized Systems, 2005. , 2005, pp. 7-14.

[91] IBM. (2010, June) PLM: Product Lifecycle Management. [Online]. http://www-01.ibm.com/software/plm/

[92] (2010, June) Towards Open Grid Services Architecture. [Online]. http://www.globus.org/ogsa/

[93] M. P. Singh and M. N. Huhns, Service-Oriented Computing.: John Wiley & Sons, 2005.

[94] IBM. (2010, June) INNOV8. [Online]. http://www-01.ibm.com/software/solutions/soa/innov8/index.html

[95] Scott Swegles, "Business Process Modeling with SIMPROCESS," in Proceedings of the 29th conference on Winter simulation, Atlanta, GA, 1997, pp. 606-610.

[96] (2010, Feburary) INOA, ISTF. [Online]. http://www.iona.com/solutions/it_solutions/istf.htm

[97] Wei-Tek Tsai, Chun Fan, Yiinong Chen, and Rayment A. Paul, "A Service-Oriented Modeling and Simulation Framework for Rapid Development of Distributed Applications," Simulation Modelling Practice and Theory, vol. 14, no. 6, pp. 725-739, August 2006.

[98] Wikipedia. (2010, June) High Level Architecture (HLA). [Online]. http://en.wikipedia.org/wiki/High_level_architecture_%28simulation%29

[99] (2010, June) Web Service Business Process Execution Language version 2.0. [Online]. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[100] Wei-Tek Tsai, Qian Huang, Xin Sun, and Yinong Chen, "Dynamic Collaboration Simulation in Service-Oriented Computing Paradigm ," in the 40th Annual Simulation Symposium (ANSS), Norfork, VA, 2007, pp. 41-48.

[101] Wei-Tek Tsai, Xinyu Zhou, and Xiao Wei, "A Policy Enforcement Framework for Verification and Control of Service Collaboration ," Information Systems and E-Business Management, vol. 6, no. 1, pp. 83-107, January 2008.

[102] Wei-Tek Tsai, Xinyu Zhou, and Yinong Chen, "SOA Simulation and Verification by Event-Driven Policy Enforcement ," in the 41st Annual Simulation Symposium (ANSS), Ottawa, 2008, pp. 165-172.

[103] T. O'Reilly. What Is Web 2.0. [Online]. http://oreilly.com/web2/archive/what-is-web-20.html

[104] Martin.C. Cooper, "Semantic Distance Measures," Computational Intelligencef, vol. 16, no. 1, pp. 79-94, December 2002.

BIOGRAPHICAL SKETCH

Xin Sun was born in Jiaohe city, Jilin province, People’s Republic of China. He attended Peking University (PKU) in China, where he received the Bachelor of Science in computer science in 2006. Xin further pursued his doctoral degree in the area of service-oriented architecture and cloud computing at Arizona State Univerity (ASU) under the supervision of Dr. Wei-Tek Tsai. Xin co-authored over 20 research papers, and one of them was cited over 300 times.

Currently, Xin works as an engineering manager at LinkedIn, building the best mobile and web applications to help professioals around the world to better connect to each other.