SwinFlow-Cloud Workflow System
- Architecture, Design, and Implementation

by

Dahai Cao

B. Eng. (Shenyang Aerospace University)

M. Eng. (Tsinghua University)

A thesis submitted to
Faculty of Science, Engineering and Technology
Swinburne University of Technology

for the degree of
Doctor of Philosophy

August 2014

To my parents, my wife, Wei Du,
my daughter, Xiran, my son, Jiasheng,
and my parents-in-law

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the thesis. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Dahai Cao

August 2014

Acknowledgements

I sincerely express my deepest gratitude to my coordinate supervisor, Professor Yun Yang, for his seasoned supervision and continuous encouragement throughout my PhD study. His guidance, wisdom, enthusiasm have made me both more mature as a person and more confident to be a good researcher. I feel fortunate to have him as mentor and friend for the past years.

I thank Swinburne University of Technology and the Faculty of Science, Engineering and Technology (previous Faculty of Information and Communication Technologies) for offering me a full Research Scholarship throughout my doctoral program. I also thank the Research Committee of the Faculty of Science, Engineering and Technology for providing me with financial support to attend conferences.

My thanks also go to my associate supervisor Dr. Jinjun Chen, to my review panel members Professor Chenfei Liu, Dr. Diana Kuo, Dr. Wei Lai, and to staff members, research students and research assistants at previous CS3 and current SUCCESS for their help, suggestions, friendship and encouragement, in particular, Dr. Xiao Liu, Dr. Dong Yuan, Dr. Qiang He, Dr. Gaofeng Zhang, and Dr. Wenhao Li, Feifei Chen, etc.

I am deeply grateful to my parents Kerang Cao and Guirong Chi for raising me up, teaching me to be a good person, and supporting me to study abroad. Last but not least, I thank my wife, Wei Du, my daughter, Xiran, my son, Jiasheng, and my parents-in-law, Guiying Liu, Chaohua Du, my elder sister, Jun Cao, my elder brother, Bo Cao for their love, understanding, encouragement, sacrifice and help. I know words can never be enough to express my appreciation to them. So I will show them more outstanding achievements as time goes by.

Abstract

Nowadays, workflow technology is relatively mature and has been one of the most popular components of process aware systems over the last two decades. From the early centralised to recent decentralised such as Grid-based workflow and Peer-to-Peer (P2P) based, workflow has improved processing capacity significantly. However, there are still many challenges in front of today's workflow systems. One of the challenges is that few of workflow architectures can efficiently support a large number of concurrent workflow instances, i.e. instance-intensive workflows, which requires a workflow management system (WfMS) to automatically scale out during peak time to tackle rapidly increasing throughputs and scale in during off-peak time to save resource usage cost.

Cloud computing has many appealing features such as virtualisation, scalability, pay-as-you-go, etc. It is able to offer powerful capacity to support large-scale applications so that it has become a favourite paradigm for large-scale applications. Current workflow research has introduced cloud features to improve workflow performance. However, the workflows with cloud features mainly focus on scientific workflows rather than instance-intensive workflows.

This thesis introduces the workflow concepts and defines cloud workflow concepts and its features. Cloud workflow is a WfMS that is hosted in cloud, or staged in or out cloud over the Internet, and can economically and dynamically scale out or in to support large-scale workflow applications. Besides all functional features of a WfMS, the cloud workflow generally has following non-functional features: template with workflow services for instantiation; automatic scaling mechanism; load balancing mechanism; alarming mechanism; billing mechanism; and cost-effective service provision.

Based on the concepts and features above, we innovatively enhance the traditional workflow reference model of workflow management alliance (WfMC) through introducing new components and thereby propose novel client-cloud architecture for cloud workflow to handle instance-intensive workflows. The architecture consists of the client side and the cloud side. Here, client-cloud means that the client side communicates with an elastic pool of workflow enactment services on the cloud side. In the architecture, besides the functional components identified by WfMC, we further propose several fundamental and essential non-functional components, including workflow accompaniment tools and workflow relevant services. The workflow accompaniment tools cover cloud workflow relevant definition tools and administration and monitoring tools; while the workflow relevant services cover the load balancing service, the alarm service, the auto-scaling service, the workflow service component image service and the billing service, which can coordinate to elastically support workflow management.

Based on the client-cloud architecture, we design the functional components identified by WfMC including process definition tool, process simulation tool, administration tool, workflow enactment service and so on, to support the fundamental workflow services. We design process representation model on the basis of the process meta-model and the primitives identified by WfMC and corresponding management tools to support workflow modelling. On the other hand, we design the workflow enactment services including workflow engine, task transaction engine, and navigation engine to support workflow execution.

We further design the fundamental and essential non-functional components including build time definition tools; runtime administration tools and service components for organisation modelling, version controlling, tool agent invoking, billing, alarm, auto-scaling and so on. In the alarm service, we design an alarm estimation model for the alarm mechanism, which can measure the overall status of the cloud workflow system and send a notification to the auto-scaling service once the system is overloaded. In the auto-scaling service, we design a scaling estimation model for the auto-scaling mechanism, which can estimate and calculate the resource demands for the scaling-out or scaling-in actions, to guarantee the architecture to provision high-performance, cost-effective and sustainable workflow services. Furthermore, we investigate the coordination between the services in the client-cloud architecture. The coordination keeps all workflow enactment services recoverable and prevents them from running out due to overloading, or releases the idle to avoid the waste.

This research work proposes two principles which reveal the influence to the sustainability of the coordination between the services. The principles are: (1) if the alarm estimation-time plus the workflow enactment service launching-time are less than the workflow enactment service decaying-time, then the workflow enactment services can be kept recoverable; (2) if the load balancing service check-time is less than the alarm estimation-time, then the load balancing service is able to monitor timely the state change.

It is different from the traditional client-server architecture where the client communicates with a static workflow enactment service or a cluster of workflow enactment services. Although the client-cloud architecture also has centralised workflow enactment services in cloud, the number of the services in the client-cloud architecture is able to scale out or in on demand while its counterpart in the client-server architecture is fixed and not elastic. It is also different from the decentralised architectures such as Grid and P2P where there is generally no central server(s) to manage all system components. But the client-cloud architecture is able to maintain a resizable pool of workflow enactment services which provision workflow services sustainably.

With our novel client-cloud architecture, we design and implement a scalable SwinFlow-Cloud (***Swin***burne work***Flow*** system on ***Cloud***) prototype. Through our evaluation of the prototype on the Amazon AWS cloud, we demonstrate that our client-cloud based workflow system is able to handle large-scale instance-intensive workflows.

The major contributions of this research are novel client-cloud architecture including the functional and non-functional components, and the alarm estimation model for the alarm mechanism and the scaling estimation model for the auto-scaling mechanism as well as two primary principles for sustainable coordination between the services in the architecture. The architecture promises a new solution for revamping traditional client-server model based workflow systems to handle large-scale instance-intensive workflows. The alarm mechanism can accurately estimate the status of cloud workflow and the auto-scaling mechanism can accurately predict the future resource demands to support the scalability of the client-cloud architecture. The two principles guarantee effective coordination of the services to appropriately handle all client requests.

The Author's Publications

1. **D. Cao**, X. Liu, Y. Yang, *Novel Client-Cloud Architecture for Scalable Instance-Intensive Workflow Systems*, Proc. of 14th International Conference on Web Information System Engineering (WISE 2013), Part II, pages 270-284, Nanjing, China, October 2013.

2. D. Yuan, X. Liu, L. Cui, T. Zhang, W. Li, **D. Cao**, Y. Yang, *An Algorithm for Cost-Effectively Storing Scientific Datasets with Multiple Service Providers in the Cloud*, Proc. of 9th IEEE International Conference on e-Science (e-Science 2013), Beijing, China, October 2013.

3. X. Liu, D. Yuan, G. Zhang, W. Li, **D. Cao**, Q. He, J. Chen and Y. Yang, *The Design of Cloud Workflow Systems*. Springer, 97 pages, 2012, (ISBN: 978-1-4614-1933-4).

4. X. Liu, Y. Yang, **D. Cao** and D. Yuan, *Selecting Checkpoints along the Time Line: A Novel Temporal Checkpoint Selection Strategy for Monitoring a Batch of Parallel Business Processes*, Proc. of 35th International Conference on Software Engineering (ICSE NIER 2013), pages 1281-1284, San Francisco, May 2013.

5. X. Liu, Y. Yang, **D. Cao**, D. Yuan and J Chen, *Managing Large Numbers of Business Processes with Cloud Workflow Systems.* Proc. of 10th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2012), pages 33-42, Melbourne, Australia, January-February 2012.

6. X. Liu, Y. Yang, D. Yuan, G. Zhang, W. Li and **D. Cao**, *A Generic QoS Framework for Cloud Workflow Systems*. Proc. of the International Conference on Cloud and Green Computing (CGC2011), pages 713-720, Sydney, Australia, December 2011.

Table of Contents

List of Figures

Chapter 1
Introduction

This thesis presents the client-cloud architecture, design, and implementation of a cloud workflow system, addresses the functional and non-functional service components in the architecture, and an alarm mechanism and an auto-scaling mechanism for instance-intensive workflows. Moreover, the sustainable coordination between the service components is also investigated. We propose two principles to facilitate a sustainable coordination between the services in the client-cloud architecture: (1) if the alarm estimation-time plus the workflow enactment service launching-time are less than the workflow enactment service decaying-time, then the workflow enactment services can be kept recoverable; (2) if the load balancing service check-time is less than the alarm estimation-time, then the load balancing service is able to monitor timely the state change. On the basis of the innovative architecture, we design and implement a SwinFlow-Cloud prototype for demonstration and evaluation purposes. The novel research reported in this thesis presents a new approach to construct a workflow system which supports large-scale instance-intensive workflow applications.

This chapter is structured as follows. First, we introduce cloud workflow management in Section 1.1. Then, Section 1.2 introduces concept of large-scale instance-intensive workflow. Section 1.3 outlines the key issues of this research. Finally, Section 1.4 presents an overview of the remainder of this thesis.

1.1 Introduction to cloud workflow management

Workflow has been widely applied in process aware applications of business processing and scientific computation since the workflow reference model was proposed by workflow

management coalition (WfMC) in 1995 [44]. Workflow technology has greatly facilitated the automation of the process logic and improved the process management efficiency. Today's workflow management system (WfMS) has been one of the core components in many software architectures to satisfy requirements of various workflow applications. With workflow applied in more fields, more domain-specific requirements facilitate the architecture of WfMSs to be evolved. From the early centralised architecture to recent decentralised architecture, the architectures are able to provision more and more powerful capability in performing workflow. However, for the instance-intensive workflow, few of architectures of WfMSs can efficiently support with high scalability, high availability, high reliability, and cost-effectiveness.

Recently, cloud computing is deeply impacting on and changing the architecture, deployment, and provision of the traditional and current software applications [7, 11]. Cloud computing is "a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualised, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet" [34]. Cloud computing enables computation capability to be utilised and delivered like water, electricity, or nature gas. It offers a new approach to tackle the exponentially growing workflow applications. This section introduces the workflow concept and relevant terminology and further addresses the concept, features, and functionalities of a cloud workflow.

1.1.1 Workflow concept and relevant terminology

In workflow communities, there are many similar concepts for defining workflow. In 1990s, WfMC proposed a glossary to define the workflow, WfMS concepts and other important concepts [84]. Here, we follow six primary concepts or definitions in the terminology and glossary for workflow management:

1. Business process is *a set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships*. As indicated by the definition, business process is an abstract concept to define a business flow.

2. Process is *a formalised view of a business process, represented as a co-ordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common goal*. As indicated by the process definition, process concept only considers the business process at one perspective of process. There are other perspectives to consider the business process, such as data perspective, application perspective, etc. The perspectives concern much with the domain-specific issues.

3. Process definition is *the representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc*. In contrast to the business process or process, it is more concentrate and can be considered as an operable or executable implementation of a business process or process. In addition, the process definition contains more information or data than the process. But for simplification, process definition can be denoted as process. Thus, process definition is usually considered as a synonym of workflow definition.

4. Process instance is *the representation of a single enactment of a process*. Here, the process of the definition refers to process definition. That is, a process instance is corresponding to the process definition. The relationship between business process (or process), process definition and process instance is similar to that of algorithm, Java class and Java object in Java object oriented computing. A process instance is an execution of a process definition to achieve the expected goal and a process definition is an implementation of a business process or process.

5. Workflow is *the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*. Workflow is an abstract automatic enactment of a business process. As indicated by the name, it does not only represent the concept of business process but also the dynamic and flowable characteristic of a business. Thus, workflow is considered as mainstream workflow research terminology instead of business process. The process definition is interchangeable with workflow definition and the process instance is also interchangeable with the workflow instance. In this

thesis, a workflow definition is a synonym of a process definition and a workflow instance is a synonym of a process instance. In addition, process can be considered as process definition or workflow in our context. For a wider scope of workflow management, process is synonym of workflow, while, from the process perspective, process is synonym of process definition.

6. WfMS is *a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications*.

The above six concepts are our research foundation. Furthermore, we will inherit the traditional workflow reference model of WfMC to architect cloud workflow [44]. The traditional reference model will be reviewed in detail in Section 2.1.

1.1.2 What is cloud workflow

Cloud workflow is *a WfMS that is hosted in cloud, or staged in or out cloud over the Internet, and can economically, dynamically scale out or in to support large-scale workflow applications*. Nowadays, workflow communities such as industry, open source, and research communities are introducing cloud computing to revamp the existing WfMSs. Nevertheless, some WfMSs only integrate cloud computing through invoking APIs of cloud infrastructure and do not change their architectures, while some other WfMSs are only staged or migrated in cloud and do not utilise any features of cloud computing. Such WfMSs should not be considered as cloud workflows.

The definition of the cloud workflow contains a few key points. Firstly, the cloud workflow is a WfMS which hosts functional and non-functional workflow services. That is, it can define a process definition, and then exports the definition in workflow enactment service to create a process instance for execution. A process instance is a single enactment of a process definition.

Secondly, the cloud workflow is a cloud application which is either a new WfMS which is designed to adapt cloud computing or a traditional WfMS which can be staged and deployed in cloud over the Internet. On the other hand, the definition indicates that a cloud workflow

can be encapsulated and delivered inside or outside the cloud over the Internet. That is, workflow services can be provisioned to workflow users like tap water.

Thirdly, the cloud workflow utilises the features provided by underlying cloud infrastructure to automatically, elastically, theoretically unlimitedly, scale out or in on demand. Automatic scaling is not a new concept for traditional WfMS architectures. But the automatic scaling of cloud computing provides more elastic, flexible, dynamic, economic than that of the traditional WfMS architectures. Hence the cloud workflow has more efficacies in resource utilisation than the traditional architectures.

Fourthly, the large-scale workflow application refers to the application which occupies a great deal of resources when a WfMS runs on it. The resources include workflow service resources such as enactment service resources, storage service resources, organisation service resources, user service resources and so on, and underlying system resources such as CPU, memory, network, IO and so on. The large-scale workflow can be categorised as different workflow classes according to different research foci. From the perspective of computation resources, e.g., CPU, it can be denoted as computation-intensive workflow; from the perspective of data management, it can be denoted as data-intensive workflow; from the perspective of enactment resources, it can be denoted as instance-intensive workflow, etc. In this thesis, we mainly focus on the instance-intensive workflow.

Fifthly, the cloud workflow offers economical services to support cost-effective workflow applications for significantly saving hardware and software costs. That is, the cloud workflow owners need not frequently give a trade-off between WfMS performance and investment on hardware and software when the scale of workflow applications increasing, as they did in the past. Furthermore, the cloud workflow may lease workflow services to multiple organisations and host workflow applications to the organisations. The organisations pay for subscriptions of the workflow services and get services quickly without the needs for the investment in expensive high-performance hardware, software, and even workflow administration.

From the perspective of cloud computing, everything can be considered as a service, that is, infrastructure as a service (IaaS), platform as a service (PaaS), or software as a service (SaaS), etc. With a view of the traditional process definition, WfMS is generally categorised as an application middleware and provisions workflow enactment services. When staged in

cloud, relative to the underlying cloud infrastructure, a WfMS is hosted on top of cloud software stacks and can be categorised as a SaaS. On the other hand, relative to the workflow applications, a WfMS is an underlying management platform to workflow services. Thus a WfMS can also be categorised as a platform as a service (PaaS). In this thesis, we consider a WfMS as SaaS.

1.1.3 Features of cloud workflow

As discussed above, cloud workflow is a WfMS. That means that the functional features in cloud workflow are compatible with those of the workflow reference model proposed by WfMC. Moreover, the cloud workflow concerns with more non-functional areas and offers the following prominent non-functional features:

1. A template with workflow services for instantiation: A cloud workflow can be scaled out to support more workflows through instantiating new workflow service components from the template, which is preconfigured at build time or dynamically configured to meet the changing performance demands at runtime.

2. Automatic scaling mechanism (auto-scaling): Scaling has existed in many traditional WfMS architectures and is not an exclusive concept in the cloud workflow. However, such scaling is implemented manually at the underlying infrastructure level by enhancing hardware capability or adding clusters. The scaling mechanism in cloud workflow automates the similar operations on the basis of cloud infrastructure. Moreover, it is an application-level mechanism other than the infrastructure-level provided by cloud and includes horizontal scaling, which adds new workflow service components that have same capability as the existing components; and vertical scaling, which changes on-the-fly the assigned underlying or platform resources to the existing components, for example, assign more physical CPU or Java virtual machine memory resources to the running workflow service components [80].

3. Load balancing mechanism: Load balancing has existed in many traditional WfMS architectures and is not an exclusive concept in the cloud workflow. Some cloud infrastructures also provide an infrastructure-level load balancing mechanism. But the mechanism of cloud workflow is an application-level mechanism which estimates the

status of workflow service components with finer granularities before dispatching requests.

4. Alarming mechanism: This mechanism monitors the overall status of all the workflow service components and estimates whether the overall status of the workflow service components reaches or exceeds some predefined thresholds. If the thresholds are met, alarm is able to notify the automatic scaling mechanism to take actions to leverage it. It is an application-level mechanism other than the infrastructure-level.

5. Billing mechanism: This mechanism is derived from the pay-as-you-go model of cloud computing [34]. A cloud workflow is a platform which manages massive workflow applications and is able to bill the applications on the basis of payment policies of the underlying pay-as-you-go model.

6. Cost-effective service provider: A cloud workflow aims at providing economic workflow services to support large-scale workflow applications. Therefore, the investments of workflow customer can be reduced significantly.

1.2 Introduction of large-scale instance-intensive workflow

Large-scale instance-intensive workflow refers to the workflow applications which are able to produce a large number of concurrent workflow instances and occupy a great deal of workflow enactment resources and other related resources such as storage resources, communication resources, etc. The workflow applications have the following significant characteristics:

1. Huge volume of workflow instances. The number is normally over a million or even a billion. The instances commonly swallow massive system resource though the individual occupation of one instance may be insignificant. There are many typical examples of the workflow applications, e.g., the mobile phone or landline telephone communication charge workflow, bank cheque processing workflow, network flow charge workflow, securities exchange workflow, etc. The workflow applications may generate enormous numbers of workflow instances all the time.

2. Regular or irregular upsurge or ebb in the volume with wide amplitude. The number of workflow instances is changing regularly or irregularly with a big range. For example, the number of a mobile phone charge workflow may regularly increase few times rapidly on New Year's Eve and regularly decrease after the holidays but may irregularly increase or decrease due to some public incidents or large disaster such as earthquake.

3. Light weight computation and data intensity. In contrast to computation-intensive or data-intensive workflow, instance-intensive workflow normally has no large-scale computation and datasets during the execution.

As discussed above, the instance-intensive workflow commonly emerges in the enterprise-level workflow applications.

1.3 Key issues of this research

As discussed above, few of current workflow systems are able to tackle instance-intensive workflow appropriately. Therefore, the most important key issue of this thesis is to investigate novel client-cloud architecture for instance-intensive workflow. This architecture will inherit all functional components of the traditional workflow reference model from WfMC and further add some new non-functional components. These new service components collaborate to guarantee the scalability, availability and reliability of cloud workflow. They are the bridges between cloud workflow and cloud infrastructures. Thus, another key issue of this research is the investigation of functionalities of the new non-functional components. In this thesis, we focus on the alarm and auto-scaling mechanisms in the client-cloud architecture. They are both the primary features of the cloud workflow. We will propose an alarm estimation model and a scaling estimation model to support those mechanisms.

Furthermore, this research work investigates the sustainability of the coordination between the functional and non-functional service components in the client-cloud architecture. The key issue of the sustainability is to keep all workflow enactment services recoverable. We will analyse and reveal the constraints between the services and derive two principles. When ignoring the principles, it may be not sustainable for service coordination. Moreover, this

thesis discusses the influence of some other elements to the coordination. The two principles are expected to be helpful for the traditional or legacy WfMS to be staged in cloud.

On the basis of the client-cloud architecture, we will represent our designs and implementation of cloud workflow – SwinFlow-Cloud prototype. The key issues of the implementation is to design a flexible and extensible structure regardless the client side or cloud side, that is, all workflow system components are designed as plug-and-play components and can be easily plugged in to or removed out from the system. Moreover, the prototype is able to be staged in cloud, i.e., Amazon AWS in this thesis, and demonstrated to support the instance-intensive workflow.

1.4 Overview of this thesis

This thesis covers the multiple aspects of cloud workflow, including concepts, features, architecture, functionalities, design and implementation. It represents comprehensive and in-depth efforts of cloud workflow research.

In Chapter 2, the familiar traditional workflow reference model from the WfMC is briefly reviewed. Then it further analyses the traditional WfMS architectures including centralised and decentralised, and investigates some representative commercial and open source workflow systems based on those architectures. The cloud workflow research and some well-known systems are reviewed. The research on the alarm and auto-scaling mechanisms in recent years is also surveyed.

In Chapter 3, a motivating example is introduced to illustrate research problems. These problems are given an in-depth analysis and the system requirements which facilitate the design of cloud workflow are refined.

In Chapter 4, this thesis presents the novel client-cloud architecture and addresses the service components in the architecture. Then we discuss the coordination between the services to provision the sustainability and the elements which can influence the coordination.

In Chapter 5, we present the design of the functional components which are inherited from the traditional workflow reference model. The design covers the build time and runtime

functions. The build time function refers to the process management. The runtime functions cover the workflow enactment service, workflow administration and monitoring.

In Chapter 6, we present the design of the models and functionalities of the non-functional aspects. The design covers organisational management, version management, tool agent management, cloud workflow relevant service definition functionalities, cloud workflow relevant service administration and monitoring functionalities, and the workflow relevant services. In this chapter, our main contributions are that we propose an alarm estimation model and a scaling estimation model and further reveal the constraints between the elements in Chapter 4 and propose two primary principles to guarantee the sustainability of the coordination.

In Chapter 7, the implementation of the cloud workflow prototype, SwinFlow-Cloud, based on the system design described in Chapters 5 and 6 is addressed in detail. We firstly introduce the technology which is used to develop SwinFlow-Cloud. Secondly, we address the system development. Thirdly, we give an overview of system runtime environments, i.e., Amazon AWS. Fourthly, we present the implementations of critical functional and non-functional components.

In Chapter 8, a demonstration experiment is presented for evaluation. This chapter uses the motivating example in Chapter 3 as case study. We use an experiment to evaluate whether SwinFlow-Cloud can meet the requirements refined in Chapter 3 and compare the results of the experiment with the data of the traditional non-cloud workflow system.

In the last chapter, Chapter 9, we summarise the work presented in this thesis, the major contributions of this research, and consequent further research works.

Chapter 2

Literature Review

In this chapter, a brief introduction of the WfMC's reference model is given in Section 2.1. Then Section 2.2 discusses various traditional workflow system architectures for supporting large-scale applications, some representative commercial or open source workflow systems, and their advantages and disadvantages. Section 2.3 introduces the current cloud workflow research including Hadoop based workflow and other commercial or open source cloud workflow and auto-scaling mechanism.

2.1 The WfMC's workflow reference model

WfMC, found in 1993, published the workflow reference model in October 1994 [44]. The reference model identifies the fundamental functions and the interfaces of a WfMS which greatly contributes to the workflow research in past two decades. Figure 2.1 outlines the components and interfaces of the reference model.

- Process definition tools: A variety of tool kits may be used to model, transfer, analyse, describe and document a process definition. The tools may interface (import/export process definitions) to the workflow enactment service via the process definition import/export interface (Interface 1).

- Workflow engine: A service or "engine" that provides the runtime execution environment for a process instance.

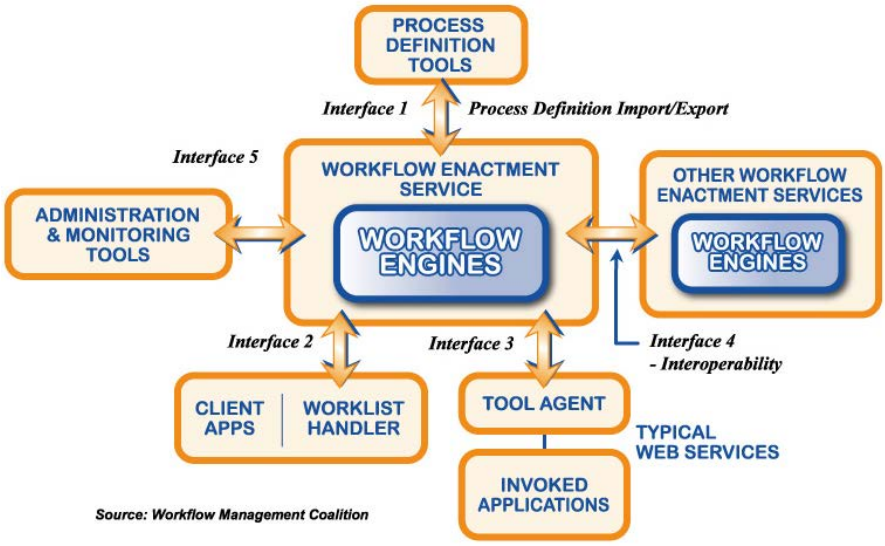

Figure 2.1 The WfMC's workflow reference model - components and interfaces

- Workflow enactment service: A service that may consist of one or more workflow engines in order to create, manage and execute process instances. The service may interface to other components via the interfaces (Interfaces 1-5).

- Administration & monitoring tools: A variety of tool kits may be used to monitor, administrate, control, and query a process instance. The tools may interface to the workflow enactment service via administration & monitoring interface (Interface 5).

- Client apps/worklist handler: The worklist handler may be the user-friendly software which interacts with the end-user in the activities which need to involve human resources. The worklist handler may interface to the workflow enactment service via the client application interface (Interface 2).

- Tool agent: The agent software or API packages may be accessed to the workflow enactment service. Typically it is a web service or Java API package or C API library. The tool agent may interface to workflow enactment service via the invoked applications interface (Interface 3).

- Other workflow enactment services: The heterogeneous workflow enactment services are produced by different workflow system vendors. The services may interface to the workflow enactment service via the workflow application interoperability interface (Interface 4).

At an abstract level, in a WfMS, a process definition consists of a collection of tasks (activities or steps) and navigation rules among them to accomplish a specific goal. It is defined or depicted in XML based process definition language (XPDL) [21] or other definition formats and then delivered to an engine in a workflow enactment service for instantiation and execution. The engine will collect all essential information and create one or more process instances using the definition as template. Process instances will be executed and navigated step by step under the control of the engine. Client application tools or worklist handler can fetch current work items from the engine for interaction and submit them back to the engine after completion. Administration and monitoring tools are able to control the process through workflow operations such as launching, initiation, suspension, termination, and completion, etc.

The release of the reference model is an important milestone in workflow research. Nowadays it has been a mainstream standard to identify a WfMS in workflow communities. Many workflow systems are designed based on or compatible with the reference model. For leading the workflow research and development, the WfMC further released other standard documents such as XPDL, and the workflow terminology and glossary [21] [84], etc.

With workflow application areas expanding, two important branches emerge in workflow research communities: business workflow and scientific workflow [91]. The former means workflow applications that aim at implementing business targets and are normally used in enterprises, governments, or non-profit organisations. The latter means workflow applications that aim at implementing a complicated scientific computation, modelling, analysis, simulation, mining, derivation, etc., and are generally used in scientific research institutions, laboratories, or universities. The instance-intensive workflow that this thesis focuses on can be normally considered as a business workflow; while the computation-intensive workflow can be generally considered as a scientific workflow. Although the reference model does not change or is mended in component structure, WfMC needs to confront the new challenges to adapt the new progress. The reference model was proposed long before the advent of the cloud computing. For current cloud workflow, this reference model is not able to meet the requirements of large-scale cloud workflow applications.

2.2 An overview of traditional WfMS architectures for large-scale workflows

This section gives an overview of various WfMS architectures and analyses several representative commercial or open source WfMS and summaries the disadvantages for supporting large-scale instance-intensive workflow applications.

2.2.1 Centralised WfMS architecture

The centralised WfMS architecture means that all workflow enactment services are centralised in the central logical or physical hosts or platforms while workflow users or applications interact with them through sending requests to the hosts and receiving responses from the hosts on networks. The most dominant paradigm of the architecture is the client-server model based WfMS (C/S WfMS). The central logical or physical hosts or platforms are considered as the server side of WfMS, i.e., workflow server; while the workflow users or applications are considered as the client side, i.e., workflow client. With the revolution of the C/S model and development of the Internet and Web, the C/S WfMS has been a mainstream architecture for WfMS and adopted by many vendors or open communities of WfMS.

The workflow server in the early C/S WfMS normally hosts in a standalone, high-performance workflow server to bear all workflow instances. With the scale of workflow instances increasing, the disadvantages emerge: the standalone workflow server becomes a potential or realistic bottleneck of performance. Once the system resource of the workflow server runs out, the whole WfMS will fail or shut down. It urges the owners of WfMS to invest the higher performance hardware to try their best to reduce the possibility of the failure. However, the standalone C/S WfMS cannot guarantee high reliability, stability, availability, and adaptability in the performance due to lack of the scalability.

The cluster is introduced in the centralised architecture for improving the scalability. The cluster in computer science means that “a group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine” [77]. The term “whole computer” means a system, here, it is a workflow server, which can run on its own, apart from the cluster; each workflow server in a cluster is typically

referred to as a node. The cluster has many advantages comparing with the standalone workflow server:

- More powerful scalability: The cluster is a group of nodes to leverage the workflow workload instead of one standalone workflow server. Moreover, we can configure large clusters which have an enactment power far surpassing the current scale of workflow instance, or we can configure modest clusters which are possible to add new nodes to the cluster in small increments.

- High availability: Each node in a cluster is a standalone workflow server. Therefore, the failure of any node does not mean loss of service. The high fault-tolerance also means high reliability and stability.

- Superior price/performance: It is possible to build a cluster with equal or greater workflow enactment power than a single large workflow server, at much lower cost.

Although the cluster uses a group of workflow servers to replace a standalone workflow server, the cluster has its disadvantages:

- Complex configuration: The structure of clusters needs to be preconfigured before launching a WfMS. The cluster has various structures of nodes, such as passive standby, active secondary, or nodes sharing memory, etc., so that system designers need to spend a great deal of time to estimate the workflow workload at build time stage. At runtime stage, system administrators may need to frequently monitor and adjust the structures for meeting the high demand.

- Redundancy: A cluster usually builds redundant nodes, which significantly surpass the current workload of workflow instances, to provide high availability, scalability, and reliability. If the master nodes in cluster cannot bear more workload of workflow instances, the cluster will allocate the workload to the redundant nodes for leveraging. If the master nodes can bear all workload, the redundant nodes will be left unused. Thereby the scalability obtained by the redundancy is inflexible and it increases the cost of system investment.

- Limited scalability: The cluster undoubtedly enhances the scalability of the centralised architecture. However, as mentioned in the last item, the scalability is inflexible and

lacks elasticity. In other words, to ensure the scalability, it needs to provide a cluster which is much larger than current requirements. Moreover, it can just scale out but cannot scale in which means the waste of resources.

Next some representative commercial and open source WfMSs are used to demonstrate our analysis:

- IBM Business Process Manager (IBM BPM): This product is a typical WfMS to support large throughputs and high concurrency workflow instances[1]. There are four deployment topology patterns: (1) single cluster, (2) two clusters of remote messaging, (3) three clusters of remote messaging and remote support, (4) four clusters of remote messaging, remote support, and Web. For supporting a large-scale workflow, the ideal pattern is the last one which is able to ensure a powerful scalability. But it needs a investment at the large cost of hardware infrastructures. The hardware infrastructures bring the complex system configurations during deployments. Although the product provides a deployment wizard to generate deployment environments for simplifying the workload, system architects are not allowed to make changes and regenerate the deployment environment once it is generated. Also, if a generated resource is modified, that change is not reflected in the deployment environment.

- SAP NetWeaver Business Process Manager (SAP NetWeaver BPM): It is an important component of the SAP NetWeaver platform component set[2]. The cluster of the SAP NetWeaver BPM is based on SAP NetWeaver Application Server[3]. For an example of building a cluster on SUSE Linux Enterprise High Availability Extension, there are four deployment patterns: (1) simple stack standalone, (2) simple stack high availability, (3) en-queue replication high availability, (4) en-queue replication high availability external database. For supporting large-scale applications, the ideal pattern is the third or fourth but it heavily increases the complexity of the cluster configuration.

- jBPM: jBPM is a the flexible, lightweight, and open source WfMS[4]. For clustering to support the larger workflow instances, jBPM needs to configure and install jBoss[5],

[1] http://www-03.ibm.com/software/products/en/category/BPM-SOFTWARE
[2] http://global.sap.com/platform/ netweaver/components/index.epx
[3] http://www.sap.com/pc/tech/application-foundation-security/software/application-server/index.html
[4] http://www.jboss.org/jbpm/

Apache Zookeeper[6], Apache Helix[7], etc. As shown in practice, the cluster of jBPM provides more powerful scalability than a standalone jBPM server but it does not avoid the disadvantages mentioned above.

In summary, although the cluster of the centralised workflow architecture increases the availability, scalability and reliability, it is implemented at the cost of the configuration complexity and hardware investments. Moreover, the scalability of a cluster is problematic once the scale of workflow instances upsurges rapidly.

2.2.2 Decentralised WfMS architecture

The decentralised WfMS architecture has been another area on workflow research for handling challenges of large-scale applications since last century. The architecture means that workflow instances are distributed and executed in various enactment components which are dispersed at separate locations or platforms although there is still a unified workflow specification to define all relevant components. The dominant feature of the architecture is its loosely coupled components, which may be homogeneous or heterogeneous, complex or simple, organisational or cross-organisational, logical or physical to support workflow. In the architecture, the workflow can be graphically or textually depicted in any definition languages or scripts in definition tools. When a workflow definition is delivered to the architecture, it will be interpreted and mapped into many distributed components for execution. The typical paradigms of the architecture include Grid based WfMS, P2P based WfMS, Agent based WfMS, and hybrid WfMS.

2.2.2.1 Grid based WfMS

The Grid based WfMS is based on the Grid computing paradigm [25, 33, 92]. The WfMS includes two types of model: abstract model and concrete model. A workflow user designs a abstract model graphically or textually at build time then submits it to a Grid based WfMS for execution. The WfMS maps the abstract model onto an executable concrete workflow model which binds the specific Grid resources at runtime. After the concrete workflow model is launched, WfMS does not execute the tasks in the workflow but merely coordinates the execution of tasks. Many Grid based WfMSs initiate to orchestrate many cyber-infrastructures

[5] http://www.jboss.com
[6] http://zookeeper.apache.org/
[7] http://helix.incubator.apache.org/

such as the supercomputers, experiment devices, or extra-large computation application, etc., for automating the research procedures in a workflow way to make it easier for scientists with little IT knowledge. Therefore, Grid nodes in a Grid based workflow are more heterogeneous than cluster nodes and normally hosted in various research communities. It is more suitable for large-scale scientific workflow applications to support large-scale complex data set analysis, simulation, or computation in high-energy physics, gravitational-wave physics, geophysics, astronomy or bioinformatics, etc. The Grid based WfMS has some significant advantages as follows:

- Domain-specific scalability: Due to the capabilities in specific domains, Grid nodes in a WfMS are able to provision more powerful scalability to the workflow applications, which meet domain-specific process requirements, than cluster nodes. If clusters tackle the workflow applications, they need to configure very large and many clusters for scalability.

- Efficient orchestration: The Grid based workflow can efficiently and dynamically orchestrate the distributed resources which are located in various communities or organisations to promote inter-organisational coordination.

However, the Grid based workflow has its disadvantages as follows:

- Unbalanced scalability: Although a Grid based workflow has more powerful scalability than clusters in specific domains, the scalability is on the basis of workflow applications meeting domain-specific requirements. For example, a supercomputer with astronomical computing applications is joined as a node for a Grid based workflow. It is able to represent extraordinary capacity and scalability on astronomical computation tasks when tackling an astronomical workflow. However, this supercomputer is not suitable for performing bioinformatics workflow applications where its capacity and scalability may be of limited power. Therefore, the overall scalability in a Grid based workflow is unbalanced.

- High cost: Grid nodes are usually large-scale and complex and expensive computation devices or software and distributed in many physical locations, it is at a higher cost for Grid infrastructure to organise the nodes into Grid and maintain them.

In the last decade, the Grid based workflow communities has put great efforts to research and develop many Grid based WfMS and solved various complicated scientific issues. Next, we select two successful systems to demonstrate the Grid based WfMS.

- Kepler: It is an open source scientific WfMS[8]. It is based on the Ptolemy II System[9] and inherits the actor-oriented modelling paradigm. Kepler's application areas include bioinformatics, computational chemistry, eco-informatics, and geo-informatics, etc.

- Pegasus: Pegasus[10], which stands for planning an execution in Grid, is a framework that maps abstract workflows onto Grid resources [26, 27]. It enables scientists to build workflows in abstract representations without worrying about the details of the underlying execution environment. Through a complex mapping, Pegasus is able to generate a concrete workflow and optimistically execute the workflow in Grid. The mapping means that firstly Pegasus needs to look up a resource which is appropriate in many characteristics such as physical locations, usability, complexity, solution cost, etc.; then it locates the appropriate resources to execute the components and generates an executable workflow of jobs that can be submitted to the Grid. Pegasus has been used in a number of scientific domains including astronomy, bioinformatics, earthquake science, gravitational wave physics, ocean science, limnology, and so on.

In summary, although the Grid based workflow improves the domain-specific scalability, its scalability is unbalanced and its cost is high.

2.2.2.2 P2P based WfMS

The P2P computing paradigm is that "no central coordination, no central database, no peer has a global view of the system, global behaviour emerges from local interactions, all existing data and services should be accessible, peers are autonomous, and peers and connection are unreliable" [2]. It is another popular decentralised technology for architecting distributed systems. In P2P environment, there is no client or server but only peer which is accessible to its computing resources. The status of any one peer is equal to that of another one. A peer can join or leave P2P anytime. Some research efforts have been put into the decentralised workflow area and developed some P2P based WfMSs for supporting workflow [31, 74, 85, 86]. The P2P based WfMS means a WfMS in which all components, denoted as workflow

[8] https://kepler-project.org/
[9] http://ptolemy.eecs.berkeley.edu/ptolemyII/
[10] http://pegasus.isi.edu/

peer, are built on the basis of a P2P infrastructure to support workflow. The P2P based workflow architecture has some significant advantages as follows:

- Decentralisation: The whole WfMS is decentralised in many peers in P2P infrastructures. The characteristic means a little possibility of single point of failure and enhances the availability, fault-tolerance, and robustness of a whole WfMS.

- Dynamicity: A peer can easily join P2P for workflow functionality and leave P2P any time. It is more flexible than Grid nodes.

- Powerful scalability: New peers can easily scale out to undertake a particular role or functionality in a workflow. In contrast to the Grid based workflow, a peer is able to provide more common capacities to tackle the workflow instances. Therefore, the scalability is more balanced than the Grid based workflow.

- Powerful interoperability: The resource or data of every peer is shared by and accessible to all other peers. P2P is more interoperable than Grid.

However, P2P based WfMS has its disadvantage as follows when tackling the instance-intensive workflow:

- High collaboration costs in peers: The WfMS completes workflow instances depending on the collaboration between many autonomous peers with various roles and functionalities. With the scale of workflow instance upsurging, the WfMS is able to scale out by more new peers to tackle the increasing workload. There is massive data to be transferred or accessed between the peers through requests or responses. High collaboration cost will be the main bottleneck of the WfMS when handling the instance-intensive workflow. The cost of the P2P based architecture is higher than that of the centralised architectures.

Here, we introduce two representative P2P based WfMSs:

- SwinDeW (*Swin*burne *De*centralised *W*orkflow) [86]: It adopts a flat, flexible, and loosely coupled architecture without the centralised data storage and control engine. The architecture consists of four layers: application layer, service layer, data layer, and communication layer. The application layer defines application-oriented semantics to fulfil workflow functions; the service layer is a core of the WfMS and consists of peer

management service, process definition service, process enactment service, monitoring and administration service; the data layer consists of the distributed data repositories (DRs) that store workflow-related information; the communication layer is the underlying network infrastructure which enables data transfer between physical machines through point-to-point connections. Workflow Participant Software (WfPS) is application layer software that provides interfaces to interact with a workflow participant and other WfPS, requesting services and responding to requests. As a basic working unit in SwinDeW, a peer consists of a WfPS and a set of DRs. It resides on a physical machine and enables direct communications with other peers to execute workflow. Furthermore, peer management service in the service layer provides system services such as peer configuration, discovery, or registration, etc. SwinDeW has some extension versions such as SwinDeW-B [74], SwinDeW-S [75], SwinDeW-G [89], SwinDeW-C [55], etc. They are all based on the P2P computing paradigm.

- A WWPD and WWP based workflow architecture: This architecture is on the basis of the Web Workflow Peers Directory (WWPD) and Web Workflow Peer (WWP) [31]. The WWPD is an active directory system that maintains a list of all peers (WWPs) that are available to support Web workflows. Each WWP encapsulates adequate functionality and knowledge for workflow and to decide which WWP needs to be activated next in the workflow. When executing a workflow instance, the WWPD assists WWPs to locate other WWPs and use their services and resources. This architecture, workflow enactment functionality and data are distributed among the WWPs. It is completely decentralised as no central workflow engine is employed to coordinate workflow. The WWP encapsulates some necessary knowledge to perform the workflow and also to delegate other knowledge of the workflow to other WWPs.

In summary, although P2P based workflow implements a powerful scalability through decentralising WfMS architecture, its collaboration costs increases when challenging the instance-intensive workflow.

2.2.2.3 Agent based WfMS

Software agent concept is originated in Artificial Intelligence (AI) research[66]. Generally a software agent is “reactive, autonomic, collaborative, inferential, continuously temporal, personal, adaptable and mobile” [12]. The agent based WfMS means all workflow enactment services are dispersed in an agent computing paradigm. Its key design is to take each

workflow functional component as a proactive and autonomic agent that is able to communicate and negotiate with other agents using interoperable agreements. Thus, the workflow enactment service consists of agent sets. Some characteristics of agent based WfMS are similar to P2P based WfMS, such as the loosely coupled architecture, autonomy, collaboration, etc. However, obvious differences exist between them. For example, P2P based WfMS has no central control. Any peer needs to discover and recognise other peers; while agent based WfMS has two special agents: one is responsible for creating, deleting, and naming other agents; the other stores the directory of all other agents for looking up. The main advantages of the agent based WfMS architecture for instance-intensive workflow are as follows:

- Centralised fundamental service: Not like P2P infrastructure, agent infrastructure has some centralised fundamental service such as naming, looking up, recognising, registering, etc., to support an efficient interoperation between agents. While in P2P based architecture, generally it needs to implement a functional component through underlying APIs.

- Decentralisation: An agent based WfMS is all decentralised in many agents in agent infrastructure. The characteristic is very similar to P2P based WfMS. However, peers undertake a workflow function through collaborations while an agent is able to undertake a complete function independently.

- Powerful scalability: New agent for workflow functions can be easily registered in an agent based WfMS for scaling out to undertake a particular role or functionality in a workflow.

- Powerful integration competency: According to the characteristics of agent, it is superior to build an integration component for accessing external applications outside a WfMS. The applications have just the interoperability protocols, standards, or APIs which are compatible with those of the WfMS even if they are completely heterogeneous to the WfMS.

As discussed above, some advantages or characteristics of agent-based WfMS is very similar to that of P2P based WfMS. This is the reason why an agent paradigm is used to

construct P2P architecture. However, agent based WfMS has also disadvantage as follows for instance-intensive workflow:

- High collaboration costs in agents: Like P2P based WfMS, agent based WfMS completes workflow instances through the collaboration of many agents with various roles and functionalities. As the workflow instances increasing rapidly, it unavoidably results in higher collaboration costs similar to the P2P based WfMS.

Next we introduce two agent based WfMSs:

- JBees: It is a distributed and adaptive WfMS, which is combined with an agent paradigm and coloured Petri nets formalism, with monitoring and controlling capabilities [30]. The workflow enactment services in JBees consist of some agent components: management agent, storage agent, process agent, resource agent, resource broker agent, monitor agent, and control agent, etc. Each agent provisions a workflow relevant functionality to support workflow. A new process agent is created by the management agent for executing a new workflow instance. Then the execution of the instance starts from the process agent through requesting the further instance data to the storage agent. The functionality of the process agent is similar to the workflow engine in the reference model shown in Section 2.1. For executing workflow, the resource agent or the resource broker agent participates in a resource allocation. During the execution, the monitor and control agent watch the status of the workflow instance for administration. Furthermore, JBees offers an adaptability from the perspective of agent based WfMS architecture. If a running workflow instance is modified according to the changing requirements, the management agent will create a new process agent for executing the modified instance. With the number of the workflow instances increasing, the agent number and the workload on the agents such as the storage agent, the resource agent, the resource broker agent and so on, will be stretched.

- AWfMS: It is an agent-based workflow management system (AWfMS) with resource description framework (RDF)[11] ontology schema to enable design chains cooperation, design knowledge reuse and coordination in real time [45]. The architecture is a three-tier system structure consists of user presentation layer, application/business logic

[11] http://www.w3.org/RDF/

layer, and data access layer. The core application/business logic layer has workflow management mechanism (WfMM) and agent communication mechanism (ACM). The WfMM utilises a hierarchical agent structure to manage workflow: the interface agent including user agent; the workflow agent including system maintenance agent, workflow monitoring and control agent, workflow maintenance agent, and workflow execution agent; the resource agent including buffer agent and workflow enactment agent. The agents collaborate to complete workflow instances though data requests and responses between them.

In summary, the agent based WfMS architecture is not popular in workflow research communities[46]. The architecture mainly aims at the business workflows, especially the workflows with human participations, other than the scientific workflows. For the instance-intensive workflow, the architecture does not represent the more powerful capacities than P2P based WfMS.

2.2.2.4 Hybrid WfMS

Hybrid WfMS architecture is not a new concept for workflow communities. The hybrid workflow architecture aims to take advantages of more than one paradigm to optimise the approaches to the research issues. Undoubtedly, the hybrid may introduce some optimisations and advantages in architecting a WfMS. However, we have investigated the dominant characteristics of the underlying infrastructures for architecting the hybrid WfMS in previous sections. Here, we introduce two hybrid workflow architectures:

- SwinDeW-G (SwinDeW for Grid): It is a typical hybrid decentralised WfMS, which is derived from existing work on the P2P based WfMS, SwinDeW, which is mentioned above but redeveloped as Grid services and deployed on SwinGrid, a Grid infrastructure of Swinburne University of Technology, with communications between peers conducted in a P2P fashion [89]. This enables peers in SwinDeW-G to access to Grid nodes in SwinGrid with the resources required by the peers. Moreover, the peers can use P2P to exchange various information required to execute a workflow. In SwinDeW-G, a workflow instance can be executed by different peers that may be distributed at different Grid nodes. In contrast to Grid based WfMS, this hybrid WfMS utilises the advantage of the point-to-point communication in P2P, which is more efficient in the transmission time and reduce the bottlenecks caused by the mediated approaches, to transfers data between workflow enactment services. Furthermore,

unlike normal Grid services, SwinDeW-G is always considered more dynamic than Grid based WfMS due to the easy joining and leaving of peers.

- Multi-Agent Based P2P WfMS: This system is built by plugging multi-agents into a P2P based WfMS architecture [6]. In this architecture, P2P is responsible to build time functions: workflow modelling, workflow storing and distributing workflow to workflow agents; while multi-agent is responsible to runtime functions: workflow process instantiation, task coordination, exception handling, and workflow monitoring, etc. Agents reside in different peers and can execute a workflow. Furthermore, agents in the peers can monitor workflow instances and react to unforeseen events. Through the architecture, it can build a virtual organisation involved in the inter-organisational workflow. The workflow has no central control and is represented by workflow peers with workflow agent engines. In other words, this system utilises the characteristics of no central control and dynamicity in P2P and collaboration in multi-agent to facilitate an inter-organisational cooperation.

In summary, although the hybrid WfMS does not receive sufficient attention in workflow research communities, the research is helpful to solve some specific issues. However, for the instance-intensive workflow, the WfMS does not ultimate the disadvantages of the underlying infrastructure.

As discussed above, we overviewed the various WfMS architectures which are proposed and may be considered as a potential alternative for the large-scale instance-intensive workflow. The popularisation of WfMS architectures depends on development of the underlying computing paradigms and the specific requirements. For example, the C/S WfMS architecture is used to construct a WfMS due to the development of Web, the Internet and N-tier software architecture. The cluster is introduced to architect a WfMS due to the exponential growth of the scale of workflow. The Grid based WfMS is proposed due to the development of Grid computing and the requirements of resource sharing and system integration which are desirable for sophisticated scientific workflows. The P2P based WfMS is proposed due to the development of P2P and problems of the centralised workflow architecture such as limited scalability, etc. Unfortunately, based on the analysis in this section, for the large-scale instance-intensive workflow, none of the WfMS architectures above is able to provide a satisfactory solution for supporting high scalability, high availability, high reliability, and cost-effectiveness.

2.3 An overview of cloud workflow research

With the advent of cloud computing, cloud workflow has been received more and more efforts by workflow communities. In this section, we firstly discuss the novel Hadoop based workflow. Secondly, we give an overview of the cloud workflow in industry communities. Finally, we focus on the state-of-the-art of cloud workflow in research communities.

2.3.1 Hadoop based WfMS

Recently, Apache Hadoop project has been almost a de facto standard for processing large datasets[12]. It offers an open source software library which is a framework for developing reliable, scalable, and distributed software to support large concurrent computation utilising a simple programming model and scale out from few of computers to many computers. To enhance parallelisation, Hadoop based applications are deployed in the cyber-infrastructure which can provision massive computation resources. Cloud infrastructure is able to meet the requirements of Hadoop based applications. Therefore, Hadoop and cloud computing are often discussed or mentioned together by many researchers. Hadoop based WfMS is one of the Hadoop applications. Here, we categorise Hadoop based WfMS into cloud workflow due to it can be hosted in cloud to support Hadoop computing.

This project includes five modules: Hadoop common, common utility kits for other modules; Hadoop distributed file system (HDFS), a distributed file system that provides high-throughput access to application data; YARN, a framework for job scheduling and cluster resource management; MapReduce, a programming model for parallel processing of large data sets. MapReduce [24] includes two abstract programming interface functions: Map and Reduce. Programmers need to implement the Map function and Reduce function and submit the functions to Hadoop. In Hadoop, the processing over clusters includes two steps:

- Map step: A master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. The worker nodes handle the smaller problems, and pass the results back to its master node.

[12] http://hadoop.apache.org/

- Reduce step: The master node collects the results to all the sub-problems and combines them in some way to form the output - the solution to the problem it was originally trying to solve.

MapReduce has been proven to be useful abstraction and greatly simplifies large-scale computations on processing parallelisable across huge datasets using the clusters with a large number of computers (node). Recently, Hadoop has been introduced as a core functionality of some WfMSs for solving the large-scale workflow applications[67].

- CloudWF, developed in 2009, is a Hadoop-based computational WfMS [94]. It is able to flexibly integrate and invoke workflow applications. The XML-based workflow definition, which contains blocks and connectors as workflow components, is staged in CloudWF and stored in HBase[13]. Every block has either a MapReduce or a workflow application. Each connector has block-to-block dependency which may involve file copies between connected blocks. Both blocks and connectors can execute independently, that is, the workflow's execution is decentralised by multiple Map or Reduce tasks. The workflow enactment on Hadoop is able to greatly improve scalability and parallelisation of workflow. However, we cannot find the experiments and evaluations about CloudWF. As indicated by the authors, CloudWF needs to be further tested through large-scale biological image processing workflow although theoretically it has high scalability.

- Oozie: Apache Oozie[14] is an open source, scalable, reliable, and extensible workflow scheduler system to orchestrate Apache Hadoop jobs. Oozie workflow definition consists of actions and navigation rules among them to be a directed acyclic graph (DAG) at build time. This graph can contain two types of nodes: the control nodes provide the navigation rules for beginning and ending a workflow and control the workflow execution path with possible decision points known as fork and join nodes; while the action nodes trigger the execution of actions. The significant advantages are that Oozie is able to simplify and automate the process of managing coordination among jobs and is fully integrated with the Apache Hadoop stack and supports Hadoop jobs for Apache MapReduce, Pig[15], Hive[16], and Sqoop[17] through setting an action

[13] http://hbase.apache.org/
[14] http://oozie.apache.org/
[15] http://pig.apache.org/

node as a MapReduce job, a Pig application, a file system task, or a Java application. After instantiation, Oozie is able to execute the workflow by the rules and perform the actions when the rules are satisfied at runtime.

In addition, there are some other similar workflow engines such as Hamake [18], Azkaban[19], and Cascading[20], etc., to offer similar functions for Hadoop job scheduling.

Given workflow architecture, Hadoop based WfMSs are undoubtedly a perfect alternative for processing the large-scale applications with high scalability, availability, reliability, and cost-effectiveness. Especially, it can integrate the MapReduce job into workflow. However, it aims at processing the workflow applications with extra-large computations or datasets, i.e., the computation-intensive workflow mentioned in Section 1.1.2, other than the instance-intensive workflow. These extra-large computations or datasets can be decomposed into many Map or Reduce tasks and orchestrated in workflow definition for executions.

2.3.2 Cloud workflow in industry communities

In industry communities, some vendors such as IBM, SAP, or Amazon of the commercial WfMSs provide functionalities or versions utilising cloud features. For example, IBM BPM on Cloud is a new version of IBM BPM mentioned in Subsection 2.2.1[21]. It is dedicated to and hosted in IBM SmartCloud Enterprise data centres in the United States, Canada, and Europe[22]. IBM BPM on Cloud includes all build time and runtime functions such as workflow design, execution, administration, and optimisation, and so on. It is only staged in cloud from the traditional architecture. One of the advantages is that workflow customers can get started with the workflow management quickly and conveniently through a subscription-based monthly consumption without complicated installation, deployment, and configuration of IBM BPM on cloud. However, although the system is secure and reliable as announced by IBM, the security of deploying a core system in public cloud is still a sensitive consideration for the industry communities.

---

[16] http://hive.apache.org/
[17] http://sqoop.apache.org/
[18] http://code.google.com/p/hamake/
[19] http://data.linkedin.com/opensource/azkaban
[20] http://www.cascading.org/
[21] http://www-03.ibm.com/software/products/en/business-process-manager-cloud/
[22] http://www.ibm.com/cloud-computing/us/en/

Like IBM on Cloud, Amazon Simple Workflow (SWF) is also a build-in WfMS and hosted in Amazon AWS[23]. Its functionalities are similar to IBM BPM on Cloud, whereas its design is prone to integrating and orchestrating various cloud applications or external applications. Although workflow designers do not need to concern about managing the infrastructure plumbing of workflow automation, like IBM BPM on Cloud, they need to write complex scripts or programs to implement the activity workers, which depict workflow tasks; deciders, and navigation rules. The scripts or programs decrease the usability to workflow customers.

2.3.3 Cloud workflow in research communities

In workflow research communities, there are two prominent directions in cloud workflow area. The first direction is that some research tends to shift from traditional workflow research to cloud workflow. The second direction is that some new WfMSs based on or utilising cloud computing are developed.

At present, most researchers focus on the first direction. They put great efforts to extend or shift their existing work to cloud. For example, Yet Another Workflow Language (YAWL) is a workflow language which is developed based on Petri Nets. Using this workflow language, Wil van der Aalst and Arthur ter Hofstede developed YAWL, a famous standalone WfMS, in 2002 and contributed on many workflow researches[24]. Recently, for saving cost and sharing infrastructure resource in Netherlands, a standalone YAWL is staged in cloud to be a cloud workflow, named as YAWL in the Cloud, but has no change on the existing architecture [72]. In cloud, there will be many YAWLs running to support the requests from cloud client side. Each YAWL is unaware of the others. To overcome the possible duplications of identifier of workflow instances among YAWLs, YAWL in the Cloud uses unique global cloud identifiers and maintains a mapping between global (cloud assigned) identifiers and local (engine specific) identifiers. Furthermore, YAWL in the Cloud adds some new components to support the processing of requests and responses. At front end, there is a single Web portal point to access YAWL in the Cloud. At back end, YAWL in the Cloud adds router components, which are responsible for mapping between global and local identifiers, or route the requests to the correct YAWL engine; a central database, which is used to store the global

[23] http://aws.amazon.com/swf/
[24] http://www.yawlfoundation.org/

identifier; a management component, which is responsible for communications with the routers and the central database; load balancers, which are responsible for both the incoming and outgoing requests. As indicated by the authors, the partial functions in YAWL in the Cloud are implemented and now the work is ongoing. Unfortunately, the design of the architecture of YAWL in the Cloud is not compensated by the scalability.

Cloudbus Workflow Engine (CWFE), developed by the Cloud Computing and Distributed Systems (CLOUDS) laboratory in the University of Melbourne, is derived on the basis of Grid Workflow Engine[25]. CWFE includes some key components to support users to execute and manage workflow applications: user interface applications, which provide build time functions for workflow, runtime functions for administrations; workflow engine core, which provides runtime function for workflow enactment; plugins, which support workflow to interoperate with the heterogeneous environments and platforms such as clusters, grids, clouds. Here CWFE considers cloud as an accessible application and provides a plugins mechanism to integrate cloud management tools. Therefore, CWFE is a WfMS which is prone to utilising cloud features to complete workflow.

SwinDeW-C (SwinDeW for Cloud) [55] is developed based on SwinDeW-G [89] and hosted in SwinCloud, which is a cloud computing simulation environment on the basis of Grid infrastructure of Swinburne University of Technology. The architecture of SwinDeW-C inherits the most of components of SwinDeW-G and renovates SwinDeW-G peers as SwinDeW-C peer through adding QoS management components, data management components, and security management components. SwinDeW-C aims at solving the large-scale workflow applications such as insurance claims or pulsar searching through introducing QoS constraints and security considerations and data persistence management. However, the experiment results show that renovation based on P2P does not efficiently eliminate the disadvantages of SwinDeW-G.

In the second direction, few of new cloud workflows are proposed for solving the specific issues. For example, a workflow engine for computing clouds is proposed to integrate various types of cloud applications into one workflow through specifying the tasks in the workflow [35]. This architecture of the WfMS proposes a cloud abstract layer which defines unified abstract APIs and is able to map the APIs onto concrete APIs for secure interoperation between the WfMS and the cloud applications or cloud infrastructures. The functionality of

[25] http://www.cloudbus.org/workflow/

the layer is similar to that of the plugins in CWFE above. The advantage of APIs or plugins is able to significantly reduce the architectural change in the existing WfMS. However, they only utilise cloud feature in the traditional WfMS to complement the disadvantages of the WfMS and do not overcome the disadvantages.

CloudDragon is an OpenNubula[26] based cloud workflow which is able to support scientific workflow [96]. This system architecture implements an integration of Swift scientific WfMS[27] and OpenNubula cloud infrastructure through a four-layer architecture: a client layer, which is responsible for build time functions of workflow; a service layer, which is a Swift scientific WfMS based cloud workflow service[28]; a middleware layer, which consists of the components connecting this layer with the underlying infrastructure layer, including cloud resource manager, a virtual cluster provider, and a task execution service; and an infrastructure layer which is OpenNubula. In this architecture, Swift scientific WfMS is considered as a gateway to OpenNubula. Regarding the relative positions between the WfMS and the cloud, the integration is outlined as four levels: Operational-Layer-in-the-Cloud, Task-Management-Layer-in-the-Cloud, Workflow-Management-Layer-in-the-Cloud, and All-in-the-Cloud. In other words, they can be presented as: a cloud workflow where a WfMS is outside of cloud, or partly staged in cloud, or is entirely staged in cloud [62]. In our experience, the relative positions cannot describe the complex integration in architecture. For example, they cannot describe the integration of P2P workflow and cloud: one part of P2P based workflow enactment service is inside cloud while the other part is outside cloud.

As discussed above, cloud computing is facilitating the workflow research communities to renovate the existing workflow architecture to tackle scientific workflow applications. For the large-scale instance-intensive workflow, the renovation has not advanced more significant breakthrough with the high scalability, high availability, high reliability, and cost-effectiveness.

2.3.4 Auto-scaling mechanism research

Auto-scaling, i.e., the capability or mechanism automatically scaling up/out or scaling down/in during demand spikes to maintain performance, is one of highlight features of cloud

[26] http://opennebula.org/
[27] http://swift-lang.org/main/
[28] http://swift-lang.org/main/

computing and is able to support dynamic resource provisioning or withdrawing to meet the changing requirements, and also a fundamental service for developers to implement cost-effective applications. It includes two types of scaling: "horizontal scaling (i.e. adding new server replicas and load balancers to distribute load among all available replicas) or vertical scaling (on-the-fly changing of the assigned resources to an already running instance, for instance, letting more physical CPU to a running virtual machine (VM))" [80]. As indicated by the scaling definition, the horizontal scaling is relatively easy to implement, while the vertical scaling is hard to implement because the most common operating systems do not support on-the-fly (without rebooting) changes on the available CPU or memory to support this vertical scaling.

In recent years, cloud communities have put efforts to propose several algorithms or models for improving the mechanism. However, in the open source communities, some well-known cloud infrastructures do not recognise auto-scaling mechanism as fundamental service components and hence do not provide it.

In the industry communities, auto-scaling is considered as an important service component to construct or deploy cloud applications. For example, in Amazon AWS, auto-scaling is a Web service which supports to automatically launch or terminate one or more Amazon Elastic Compute Cloud (Amazon EC2) virtual machine instances, i.e., Amazon EC2 instances for short, through user-defined auto-scaling policies[29]. This Web service is one of infrastructure-level components and hosted in Amazon EC2.

Auto-scaling in Amazon EC2 provides both horizontal scaling and vertical scaling. For the vertical scaling, Amazon provides an offline (with rebooting) changing to implement, i.e., Amazon EC2 cloud application owners need to manually shut down their Amazon EC2 instances and change the type of the instances using Amazon EC2 management tools then restart the instances. The instances will have new computation capacities as it has been changed from old system configurations to new system configurations.

For the horizontal scaling, the auto-scaling in Amazon EC2 firstly constructs a launch configuration and an auto-scaling group. The launch configuration is a script or command about Amazon EC2 instance type, Amazon EC2 image which is used to instantiate new instance, and some security information, etc., while the auto-scaling group is an Amazon EC2

[29] http://aws.amazon.com/autoscaling/

instance pool and identifies the maximum number, the minimum number, the normally desired number, and the scaling number per time of running Amazon EC2 instances. Secondly, the auto-scaling in Amazon EC2 creates a specific conditions based scaling plan for the auto-scaling group and then starts by launching the minimum number (or the desired number, if specified) of instance(s) and then starts executing the scaling plan. To be aware whether the conditions in the scaling plan are met, the auto-scaling in Amazon EC2 needs to periodically sample, makes a statistics of the sampling results and compares with a predefined threshold. If the comparison shows that the conditions are met, the auto-scaling group will scale out by launching the scaling number of Amazon EC2 instances until the specified maximum number, or scale in by terminating the scaling number of Amazon EC2 instances until the specified minimum number. The auto-scaling in Amazon EC2 is able to ensure that each Amazon EC2 instances in the auto-scaling group is running at a good status through a periodic individual health check. If it finds any unhealthy instance, it terminates this instance, removes it from the auto-scaling group and launches a new one and adds it into the group.

The auto-scaling in Amazon EC2 is simple and easy to be integrated into the cloud applications [32]. However, the infrastructure-level scaling is neither practical nor feasible for many cloud applications, especially for the long-term running cloud workflow with high throughput requirement. For vertical scaling, if WfMS administrators take the vertical scaling actions manually, they must shut down the Amazon EC2 instances with a running WfMS or workflow enactment components. It likely results in the losses of the executing workflow instances or the incoming requests and even data inconsistency in the WfMS. For horizontal scaling, the auto-scaling group is able to terminate any unhealthy Amazon EC2 instances. However, in our experience, the infrastructure-level horizontal scaling cannot be aware of the status of the cloud applications running in Amazon EC2 instances if only utilising the approaches provided by Amazon EC2. That means that it likely results in the problems above in the same way. This risk is fatal to any large-scale WfMS. Thus it can be seen that all the infrastructure-level auto-scaling mechanisms would face the similar issues like Amazon auto-scaling.

In research communities, there are mainly two directions in auto-scaling mechanism [36]. One direction focuses on a prediction model of resource demand. Roy et al. propose an approach based on model predictive control ideas. It can predict the application workload and assess the system behaviour over prediction horizon using a performance model. Then, using

the principles of the receding horizon control, it iterates over the number of look-ahead steps and computes cumulative cost for selecting each possible resource allocation [69]. Gong et al. present a novel PRedictive Elastic reSource Scaling (PRESS) scheme for cloud systems [38, 39]. PRESS first uses signal processing techniques to identify repeating patterns called signatures for predictions. If no signature is discovered, PRESS uses a statistical state-driven approach to capture short term patterns in resource demand, and uses a discrete-time Markov chain for predictions. The resource prediction models are repeatedly updated when resource consumption patterns change. Shen et al. present a system that automates fine-grained elastic resource scaling for multi-tenant cloud computing infrastructure, CloudScale system [76]. The system also uses a prediction model to estimate the resource demand online. But the estimation has inaccuracy inevitably. So the system provides the complementary under-estimation error handling schemes to correct the errors. They also developed a light-weight schema for cloud providers. This schema uses the dynamic patterns from application resources demands to adjust the resource allocations.

Utilising a predictive model to forecast the future possible workload is a good option for supporting the auto-scaling mechanism [9, 15, 41, 42, 48, 51, 53, 95]. However, the research in this direction mainly focuses on the infrastructure-level auto-scaling [5, 10, 40, 50, 81]. Furthermore, researchers propose some infrastructure-level models or frameworks to monitor the cloud resource usages [3, 7, 37, 43, 64, 68]. In cloud workflow, the infrastructure-level auto-scaling and monitoring are hard to be accurately aware of the usage of the interior cloud workflow resources such as thread pool [49, 62, 78]. It only abstractly estimates the usage of the common resource of CPU, memory, etc. Therefore, it is very difficult to precisely predict the future workloads on one WfMS. Therefore, the infrastructure-level prediction model based auto-scaling mechanism is not suitable to cloud workflow.

The other direction focuses on using some specific performance metrics to estimate the future resource requirements of cloud applications [8, 13, 17, 28, 63, 65]. Mao et al. utilise a deadline that users specify in jobs as a basic performance requirement to find a scaling plan minimising the cost in cloud applications [59]. But for workflow applications, the deadline may not be generally an important metric. For example, for scientific workflow, the workflow is generally a computation-intensive application that exhausts long time to complete. The workflow application owners are more concerned about the experimental results than the deadline. Mao et al. also focus on response time, an application-level performance metric, in

the previous works [60, 61]. For workflow applications with many manual tasks, it is a metric to measure a workflow system performance [20, 23, 47, 82]. But for the workflow applications with many system invocations, the response time may not be over concerned [71]. Therefore, in this direction, it is very important to select appropriate metrics for the auto-scaling mechanism.

The auto-scaling research mainly focuses on vertical scaling in cloud infrastructures [17-19, 22, 83, 87, 88, 90]. However, as mentioned above, vertical scaling is currently not a very feasible solution for the scalability of cloud applications.

Furthermore, for cloud workflow, the scaling needs to be aware of both the system resource usage and the interior application resource usage to estimate whether to take actions to support scalability of cloud workflow [4, 16, 18, 29, 52, 73, 79]. Unfortunately, the research of the application-level auto-scaling mechanism is very limited in research communities.

2.4 Summary

In this chapter, we firstly introduced the WfMC's traditional workflow reference model and then gave a broad overview of traditional WfMS architectures, including the centralised and decentralised architectures, and the current state-of-the-art of cloud workflow research. In the overview, we gave an in-depth analysis on the advantages and disadvantages of the WfMSs for supporting large-scale workflow applications with about 20 representative WfMS illustrations. Furthermore, we investigated the research of auto-scaling mechanism in the cloud along the line of discussing the cloud workflow research.

Chapter 3

System Requirements and Analysis

The research in this thesis is motivated by a real world workflow application. In this chapter, a motivating example of mobile charge workflow is given for discussing the disadvantages of using the existing WfMSs to solve the instance-intensive workflow to be addressed in Section 3.1. In Section 3.2, we analyse the system requirements through an in-depth discussion on the example. On the basis of the analysis, Section 3.3 refines the research problems in this thesis.

3.1 Motivating example

China Mobile Limited is a leading mobile services provider in China and has the world's largest mobile customer base[30]. There were a total number of customers exceeding 740 million by June 30, 2013, as announced in its 2013 interim results[31]. Mobile communication is the main operating business and major source of the revenue for China Mobile Limited. Suppose the 740 million customers send two short messages on average every day, there would be 1.48 billion messages to be billed and charged. Using mobile short message service (SMS) charge workflow to bill and charge, it would be 1.48 billion workflow instances to be processed every day. This example meets the first characteristic of the instance-intensive workflow in Section 1.2, i.e. huge number of workflow instances. The number of the mobile SMS charge workflows is regularly many times more in public holidays such as Chinese New Year than that of an average day. This meets the second characteristic of the instance-intensive workflow in Section 1.2, i.e. regular or irregular spikes in the volume. The execution time of a mobile SMS charge workflow instance can be only a few milliseconds on average. This meets the

[30] http://www.chinamobileltd.com/en/global/home.php
[31] http://www.irasia.com/listco/hk/chinamobile/announcement/a112098-e00941_ann_0815_0115.pdf

third characteristic of the instance-intensive workflow in Section 1.2, i.e. short response time. Therefore, the workflow can be regarded as a representative instance-intensive workflow example.

Here the object of workflow processes is a mobile service usage record, which is a data record produced by a telecommunication exchange equipment. In telecommunication, the record is defined as call detail record (CDR)[32]. A CDR consists of unique record ID, calling party (phone number), called party (phone number), starting time, terminating time, call type (voice, SMS, etc.), calling location, etc. Fig. 3.1 depicts a high level structure of the mobile SMS charge workflow. There are nine abstract tasks in this workflow:

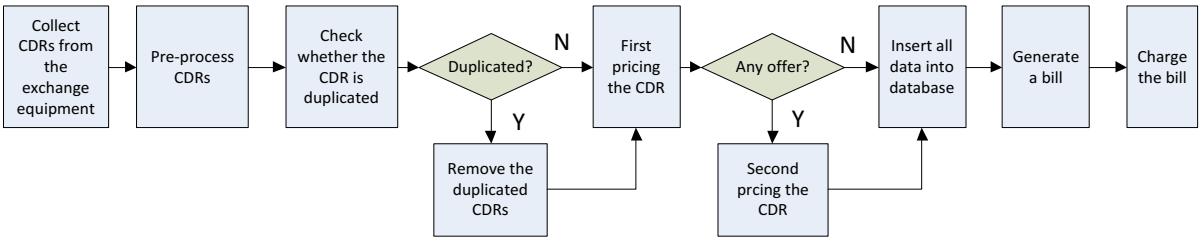

Figure 3.1 Mobile SMS charge workflow

1. Collect CDRs from the exchange equipment: In China Mobile Limited, every CDR stores the detailed information of a standard message that does not exceed 70 words. For the extra-long message, there may be multiple CDRs. This task is to obtain one or more CDRs of a message from the telecommunication exchange equipment and store the CDRs to a workflow data variable.

2. Pre-process CDRs: In general, CDRs are produced by various telecommunication exchange equipment and have the equipment-specific formats that cannot be recognised and used to generate the bill. This task is to interpret the equipment-specific formats into a unified format that can be used in the workflow. Furthermore, this task merges multiple unified CDRs into one CDR to ensure that a CDR is corresponding to one short message.

3. Check whether the CDR is duplicated: Because the exchange equipment may produce duplicated CDRs for one short message, this task is to check whether the merged CDR is uniquely corresponding to one short message. If there are multiple merged CDRs to be corresponding to one short message, the duplicated CDRs will be removed.

[32] https://supportforums.cisco.com/docs/DOC-13842

4. Remove duplicated CDRs: When the multiple merged CDRs are checked, this task removes the duplicated from the CDRs to make sure that every message has unique CDR to be processed after pre-processing task.

5. First pricing CDRs: This task calculates the amount of the CDR of the short message according to the China Mobile Limited SMS charge policies. The amount is a standard price excluding any special policies such as promotion offers. This task generates a consumption detail manifest for this short message.

6. Second pricing CDRs: This task calculates the amount of the CDR of the short message according to the special policies. This amount is the price that is applied to special policies. If the manifest has been generated, the task will update it using new pricing.

7. Insert all data into database: This task inserts all data including CDR, the consumption detail manifest into database through invoking a database service.

8. Generate a bill: This task generates a bill for this short message to charge. This bill includes the primary information in the CDR and the consumption detail manifest.

9. Charge the bill: This task charges the user according to the consumption detail manifest. In fact, in real product, it is impossible for a system directly charging a user unless this user prepaid some money for SMS usage. Furthermore, a system must merge all bills of a user mobile services usage in one charge period, including voice bills, SMS bills, Internet bills, etc. Therefore, we add the "Charge the bill" task in order to complete the description of a mobile SMS charge workflow.

At present, in China Mobile Limited, the mobile SMS charging system bears huge workload for processing SMS charge workflow every day. The statistics of China Mobile Limited indicates that mobile users sent 744.5 billion short messages in 2012[33], i.e., about 2.03 billion short messages per day on average. However, the volume of the SMS often has a steep increase during the public holidays. Especially, during the Chinese New Year holiday, i.e. the Spring Festival, short messages have become one of the most popular ways to exchange greetings. Most of people send many greeting short messages to their relatives and friends. The statistics of China Mobile Limited indicated that a total of 8.83 billion short messages

[33] http://www.chinamobileltd.com/en/business/business.php

were sent on the Chinese New Year's Eve before 2012 Spring Festival (January 23, 2012)[34]. In other words, the volume of SMS on the Chinese New Year's Eve is over four times than the daily average. In contrast, the volume of the SMS is often lower than the daily average after the peak of the holidays.

For the traditional centralised WfMS, its architecture cannot guarantee an economic and flexible scalability to tackle the instance-intensive workflow. The centralised WfMS normally configure more clusters to improve scalability to leverage the upsurge in instance-intensive workflows. Aiming at the characteristics of the instance-intensive workflow, system designers need to monitor and analyse the massive historic or real time workflow execution data for configuring the clusters to surpass the maximum workflow workload. Obviously, on one hand the configuration cannot guarantee the new upsurge when a scale of workflow instances is much larger than the existing situation. On the other hand, it is likely redundant when the scale of workflow instances is far lower than the existing situation.

Theoretically, the traditional decentralised WfMSs such as Grid based WfMS or P2P based WfMS are also able to support the processing of instance-intensive workflows. For example, the infrastructure and Grid nodes of Grid based WfMS is distributed on various logical or physical locations. After creating a workflow instance, Grid based WfMS maps the tasks in the workflow instance onto the specified Grid node for execution. The dispersed Grid nodes can guarantee the availability and reliability individually. There exist data transformation and delivery among the Grid nodes according to the navigation rules during the execution. When one billion workflow instances launch, the workloads and costs of the data transformation and delivery can be enormous and defer the executions. Thereby, the Grid based WfMS normally cannot guarantee to complete the workflow instances in time with high availability and reliability. There exist similar issues in the P2P based WfMS when executing a large number of workflow instances.

A cloud workflow system has inherent capability in supporting the instance-intensive workflow. However, as discussed in Section 2.3, current cloud workflow research puts more efforts on computation-intensive workflow. At present, China Mobile Limited produces hundreds of billion SMS bill records in its business system every year. It needs to spend long time in locating a bill or making statistics of bills according to the specific conditions in such a big dataset using traditional database query. If using the Hadoop based cloud workflow, as

[34] http://www.cnii.com.cn/telecom/2013-02/18/content_1091763.htm (in Chinese)

mentioned in Subsection 2.3.1, this query can be decomposed as multiple Map or Reduce tasks in workflow and executed efficiently to obtain results. In our experience, if decomposing the tasks in the mobile SMS charge workflow as multiple Map or Reduce tasks, because the light weight computation workload, a traditional WfMS executes such a workflow instance more efficiently than the Hadoop based cloud workflow which firstly distributes all the tasks in clusters and then collects all data from the clusters to construct final charge data.

Therefore, the analysis above motivates us to research a new cloud workflow system architecture to support instance-intensive workflow.

3.2 System requirements

With the perspective of system requirement analysis, the functions of WfMS architecture include both functional and non-functional aspects. The functional aspect mainly regards the essential and inherent functions, which are clearly addressed in the reference model in WfMC in 1995 [44], such as workflow modelling, enactment, administration, monitoring, worklist handler, and definitions of various interoperation interfaces and so on, while the non-functional aspects mainly regard quality requirements of a WfMS such as performance, security, scalability, availability, reliability, etc. With the maturity of the workflow research, workflow communities put more focus on the non-functional aspects, which are important for designing a WfMS and can also be considered as the extended functional aspects, to meet the domain-specific requirements, such as workflow data management, storage management, data mining, integrity, organisation management, etc.

According to the cloud workflow definition, it inherits the functions that the workflow reference model represented but it presents the more powerful functions to support instance-intensive workflow. In addition, in this thesis research, we put great efforts to satisfy the following functional and non-functional requirements.

R1. Cloud workflow architecture: This is a functional requirement for system architecture. To support instance-intensive workflow, the architecture should cover the concepts and features of cloud workflow.

R2. High throughputs: To tackle large-scale workflow applications, the cloud workflow should be able to execute workflow instances with high throughputs. The throughput

refers to the sum of the workflow instances which a cloud workflow completes in a time period. The completion means an entire lifecycle of a workflow instance from launching to the end. The throughput is an important metric to measure the performance of a cloud workflow.

R3. Sustainable scalability: The cloud workflow should be able to utilise the underlying cloud features to scale out by more workflow enactment service components to leverage the increasing requests from the cloud side and scale in to save the exhaust of resources. The scalability should be sustainable when the cloud workflow is running. In this thesis, we mainly focus on horizontal scaling.

R4. High availability: An available workflow enactment service component means that the component is running and can provide services whenever receiving a request. If the component runs out due to heavy workloads, it will be unavailable and result in failures of workflow instances. Cloud workflow must prevent any workflow enactment service component from running out. In a cloud workflow, the availability is represented on the availability of each workflow service component. The high availability means that any workflow enactment service component is available anytime when it processes instance-intensive workflow. The cloud workflow should provide sustainable coordination to guarantee the availability of each workflow enactment service component.

R5. High reliability: In the cloud workflow, the high reliability is represented on the low failure rate or the short recovery time when a failure emerges. The novel cloud workflow should have capabilities to reduce the failure rate or recover in a shorter time.

R6. Cost-effectiveness: The cloud workflow should be able to support a cost-effective execution of instance-intensive workflows and control the costs of cloud workflow in use of cloud computing resources.

With the above objectives archived, the instance-intensive workflows will be expected to be supported efficiently by the novel cloud workflow system.

3.3 Research problem analysis

As briefly addressed in Section 1.3, the major research problems in this thesis are on designing the novel cloud workflow architecture and implementing a prototype to meet the requirements as listed in the previous section.

The cloud workflow definition and features have extended the workflow management concepts in the traditional workflow reference model. Therefore, to meet requirement R1 addressed in the previous section, besides for implementing the concept and the features, the cloud workflow architecture further enhances the traditional workflow reference model by adding new services for supporting cloud features. The new features enable the traditional workflow reference model to be a new reference model, communicate with cloud and collaborate with other services of the cloud workflow architecture to elastically support the workflows. The components in the traditional workflow reference model such as the process definition tools, administration and monitoring tools, worklist handler and so on will be Web based or desktop based applications of the cloud workflow architecture, while the workflow enactment service will be a service provider and reside on the cloud side and may be configured as a template for duplication.

To meet requirement R2 addressed in the previous section, the cloud workflow needs to coordinate mechanisms of the auto-scaling, the load balancing and the alarm defined in Subsection 1.1.3. The coordination enables the cloud workflow architecture to scale out for supporting high throughputs. To achieve the coordination, the alarm mechanism needs to make a comprehensive awareness of status of individual workflow enactment service component and whole cloud workflow. The load balancing mechanism can balance workloads of the workflow enactment service components through checking the status data. The auto-scaling mechanism makes a decision to take scaling actions through computing the future system status on the basis of the current or historic status data. In theory, the cloud workflow can scale out by infinite workflow enactment services to concurrently support unlimited throughputs.

To meet requirement R3 addressed in the previous section, the cloud workflow needs to sustainably scale out or in. It means that the coordination between the service components in cloud workflow is sustainable. The sustainability in nature is that the behaviours of service

components satisfy some constraints on some elements related to cloud workflow. The constraints guarantee that the load balancing mechanism is able to get the proper status data from the workflow service components; the alarm mechanism is able to get the proper estimation results from the workflow service components; the auto-scaling mechanism is able to scale out by adding more new workflow enactment service components before the existing workflow service enactment service components are running out and vice versa. A sustainable coordination can prevent the service components from resource running out and keep them available when they are approaching to the boundary of running out.

To meet requirement R4 addressed in the previous section, the cloud workflow needs to guarantee that all workflow enactment service components can provision available workflow services throughout the runtime and avoid the resource running out. In cloud workflow, the alarm mechanism provides an accurate estimation to facilitate workload control of workflow enactment service components to avoid overloading.

To meet requirement R5 addressed in the previous section, one approach is that the cloud workflow guarantees the correctness of workflow instances. Therefore, the workflow enactment service component in the cloud workflow needs to guarantee the correctness of each workflow instance.

To meet requirement R6 addressed in the previous section, the cloud workflow needs to administrate and monitor the costs of system through the billing mechanism and control the costs to meet the budget.

3.4 Summary

In this chapter, we gave a motivating example to present the characteristics of the instance-workflow application and addressed the disadvantages of the traditional WfMS architectures tackling the workflow applications. Based on the motivating example, we analysed the system requirements that a cloud workflow system needs to satisfy when processing the workflow applications. Furthermore, this chapter discussed the research problems on the basis of the system requirements.

Chapter 4
A Client-Cloud Architecture for Cloud Workflow

In this chapter, according to the analysis in Section 3.3, we propose the client-cloud model based architecture to satisfy the system requirements addressed in Section 3.2. In Section 4.1, we give an overview of the novel client-cloud model based architecture. Section 4.2 presents the client side of the architecture. Section 4.3 addresses the services on the cloud side in detail. Section 4.4 discusses the advantages of the architecture and the benefit for designing cloud workflow.

4.1 An overview of the architecture

According to the definition of cloud workflow in Section 1.1 and the analysis in Section 3.3, we propose novel architecture for cloud workflow to meet requirement R1 addressed in Section 3.2. The architecture is mended from the architecture which is addressed in our previous work [14, 54, 56-58, 93]. The mended architecture presents a client-cloud model at an abstract level and consists of two sets of components: the client side components and the cloud side components, as shown in Fig. 4.1. The client-cloud model based architecture, i.e., the client-cloud architecture for short, is derived from the concepts of C/S WfMS, but the similarities and differences exist between the two architectures.

Like the client side in C/S WfMS, the client side in the client-cloud architecture can communicate with the cloud side through various protocols over the Internet such as HTTP, HTTPS, SOAP, or SSH and so on, and consists of the functional workflow management tools, such as the process definition tools, administration and monitoring tools and so on, and further introduce new components. The new components are defined as workflow accompaniment tools. As indicated by the name, the new components accompany a WfMS and satisfy the non-functional aspects of workflow management. The workflow accompaniment tools can be

considered as a set of important supplementary components and facilitate WfMS to satisfy the more domain-specific requirements. For cloud workflow, the non-functional aspects mainly include the workflow service image component, load balancing, alarm, auto-scaling and billing mechanisms. To facilitate the cloud workflow to satisfy the requirements addressed in Section 3.2, the mechanisms need build time tool components to customise specifications and runtime tool components to administrate and monitor the services on the cloud side. These tools are composed of the cloud workflow relevant service definition tools and cloud workflow relevant service administration and monitoring tools, other tools such as organisation management tools, etc.

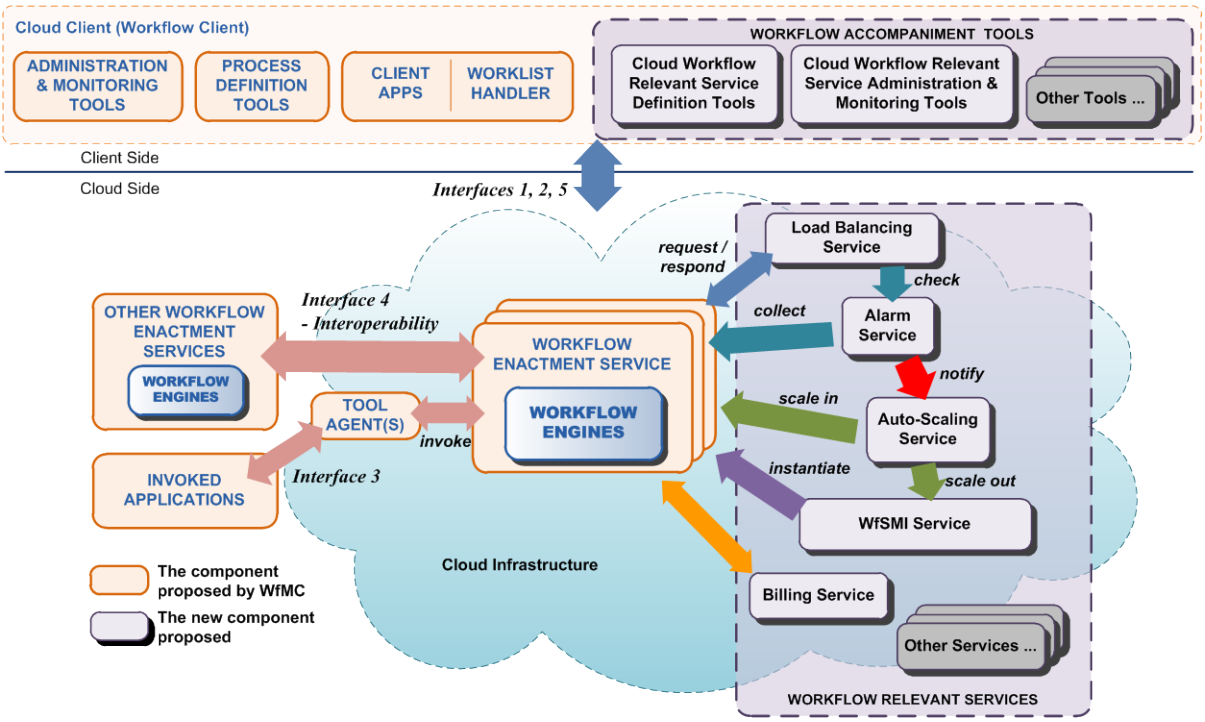

Figure 4.1 Client-cloud model based architecture

The server side in C/S WfMS is one static server or fixed clusters of servers, while the cloud side is a group of scalable virtualised service components. The servers in C/S WfMS provision workflow services individually, while the components on the cloud side can collaborate for supporting powerful instance-intensive workflow applications to meet the requirement R2 in Subsection 3.2. But the proposed novel client-cloud architecture can be categorised as a centralised approach from a high-level perspective due to the central cloud side similar to the server side.

With the view of functionality requirements, the novel architecture innovates on the basis of the traditional workflow reference model depicted in Fig. 2.1. It can be seen that the cloud side retains all the functional runtime service components and the interface definitions in the traditional workflow reference model and further introduces some new non-functional service components. The new service components are defined as the workflow relevant services. As indicated by the name, all the services which are closely related to workflow management and concern with the non-functional areas of workflow management can be categorised into the relevant services. For cloud workflow, the workflow relevant services refer to the non-functional runtime service components, including the workflow service component image, load balancing, alarm, auto-scaling and billing service components which satisfy the requirements R3 to R6 addressed in Section 3.2. All these new components will be addressed in detail in Section 4.3.

4.2 Client side

The cloud client, i.e., workflow client, on the client side is a set of Web-based or desktop-based applications and is able to construct the requests and send them to the specific workflow services on the cloud side for processing and display the results received from the services. The functional components of the workflow client are addressed in the traditional workflow reference model in detail [44]. The non-functional components, i.e., workflow accompaniment tools, include:

- Cloud workflow relevant service definition tools: The tools are to create the service-specific specifications which can be interpreted by the workflow relevant services on the cloud side and used to drive the services. For the cloud-cloud architecture, the tools are able to define the metrics with specific data structures which can support the features of cloud workflow, such as the pay-as-you-go model, the alarm mechanism, the auto-scaling mechanism, etc.

- Cloud workflow relevant service administration and monitoring tools: The functionalities of the tools are similar to the administration and monitoring tools in the traditional workflow reference model. The tools provide an interface to allow the workflow system administrators to monitor the operations of the workflow relevant services and administrate the services through the manual or the automatic ways if

necessary. For the cloud-cloud architecture, the tools can watch the status of the services including the load balancing service, the auto-scaling service, the alarm service, and the billing service and so on and administrate the specified metrics and further generate the report for analysis.

In this thesis, we focus on both the functional and non-functional aspects of cloud workflow. We utilise the functional components as a foundation to design the non-functional components of cloud workflow for satisfying the requirements addressed in Section 3.2. For constructing a complex cloud workflow, the workflow accompaniment tools in workflow client are able to contain the more components to meet various non-functional requirements.

4.3 Cloud side

As we know, the cloud side of the novel client-cloud architecture consists of a group of functional and non-functional service components, which can be composed to support the large-scale instance-intensive workflow in this thesis. The functional component is the workflow enactment service, which has been addressed in the traditional workflow reference model [44], while the non-functional components are the workflow service component image, the billing service, the load balancing service, the alarm service, the auto-scaling service, etc. In this section, we focus on non-functional services and the coordination between the last three services for scalability in detail.

4.3.1 Workflow service component image service

A workflow service component image (WfSMI) can be virtually instantiated to a new workflow enactment service component under the control of the auto-scaling mechanism of cloud workflow for meeting requirement R3 in Section 3.2. The WfSMI service is used to manage WfSMIs to support the scalability of cloud workflow. It is designed for the feature of cloud workflow: the template with workflow services for instantiation as addressed in Subsection 1.1.3.

Many cloud infrastructures offer the Web-based or command-line based tools to manage an image for cloud applications. However, the tools provide many complex steps to administrate an image. For example, in Eucalyptus cloud infrastructure, to bundle a new

machine image, it needs the designers to be proficient in the Eucalyptus operation commands. This is not always an enjoyable thing as they need to take a great deal of time to learn the comprehensive administration knowledge and skills about the Eucalyptus. As an application-level component, the WfSMI service can encapsulate APIs of various cloud infrastructures for managing images at unified abstract level and can deploy workflow enactment service components into a virtual machine created by cloud.

Like the images created by infrastructure, a WfSMI can be duplicated to multiple new workflow enactment service components when the existing components overloading. The new components can be added into the cloud workflow for balancing the workloads when the existing components cannot handle. A WfSMI presents a cloud-based alternative solution to upgrading system functions other than traditional hardware-based upgrading. We can preconfigure a WfSMI to upgrade a workflow enactment service component for adapting the changing requirements, even for troubleshooting the faults of the service components. Once the WfSMI is instantiated, the cloud workflow will be upgraded or the faults will be fixed.

4.3.2 Billing service

The billing service, designed for the feature of cloud workflow: the billing mechanism as addressed in Subsection 1.1.3, implements a pay-as-you-go model in cloud workflow and enables the workflow services to be cost-effectively delivered to workflow customers like gas, electricity.

As a SaaS, cloud workflow is a cloud application on top of software stacks and hosted in cloud computing environment, in the meanwhile, it is a workflow management platform to support workflow applications to achieve business or scientific goals. Therefore, a cloud workflow is not only billed by cloud infrastructures but also a biller for charging the workflow applications. Hence, the billing service enables cloud workflow to be a cost-effective workflow service provider which meets the feature of cloud workflow: cost-effective service provider.

Although many cloud infrastructures provide the billing service, the service only charges cloud workflow and cannot bill the resource utilisation in cloud workflow. Therefore, the billing service of cloud workflow is an essential service component to support workflow applications.

4.3.3 Load balancing service

The load balancing service, designed for the feature of cloud workflow: the load balancing mechanism addressed in Subsection 1.1.3, is a Web portal of the cloud side. All requests and responses between the client side and the cloud side are passed through this service.

Some cloud infrastructures, e.g., Amazon AWS, provide the infrastructure-level load balancing service. The service can be integrated into a cloud application for balancing the workloads between multiple service components. However, the service is under the control of cloud infrastructures. It cannot be aware of the status of cloud workflow accurately. The application-level load balancing service reside inside cloud workflow and is able to estimate the status of cloud workflow more accurately to avoid the overloading before dispatching a request.

4.3.4 Alarm service

The alarm service, designed for the feature of cloud workflow: the application-level alarm mechanism addressed in Subsection 1.1.3, can give an early alarm to the performance bottleneck resources in cloud workflow through the estimation of the service.

Comparing to the alarm service at the cloud infrastructure level, the implementation of the alarm mechanism at the application level can be aware of the runtime status of the cloud workflow at finer granularities. The alarm service of cloud workflow can even obtain the runtime details of every workflow instance such as its CPU usage, memory usage, thread number, execution progress, disk read/write bytes, etc., which is difficult to be obtained by the cloud infrastructure-level applications. The cloud infrastructure can only be aware of the rough resource usages. Whereas the details are very important for the alarm service to accurately estimate the status of cloud workflow. Moreover, it is also important for the auto-scaling service to take actions with more appropriate scale.

4.3.5 Auto-scaling service

The auto-scaling service is a core service component which provisions the scalability of cloud workflow. It is designed for the application-level automatic scaling mechanism addressed in Subsection 1.1.3.

The auto-scaling service is provided by few of cloud infrastructures such as Amazon AWS but other cloud infrastructures do not provide the service such as Eucalyptus Cloud[35], Openstack Cloud[36] and so on. However, the service implemented at the application level is able to more accurately analyse the future resource demands and scale out by a more appropriate number of workflow enactment service components to balance the workloads or scale in to save costs.

4.3.6 Coordination of services for sustainability

In previous subsections, we addressed several services to support cloud workflow. The services work together to prevent the critical resources in cloud workflow from running out. When processing instance-intensive workflow, the critical resources are very likely to run out quickly. The resources may be any competitive resource such as CPU, memory, IO devices, or even workflow enactment service components. In this thesis, we mainly concern about the workflow enactment service components when discussing the coordination of the services, i.e., the critical resources refer to the workflow enactment service components. If the resources run out, it may result in a fault status of the resources, such as unavailable, or no response for a long time, etc. We denote the fault status of the resources as an unrecoverable status; otherwise, the status of the resources is recoverable. The recoverable can be further separated into the idle status, the under loaded status, the normal working status, the busy status, and the extra-busy status; while the unrecoverable status generally refers to the overloaded status. If a part of or even the whole resources in a cloud workflow enter the unrecoverable status, they will result in the cloud workflow crashing down. Thus, the cloud workflow has not sustainability in this case. The sustainability of cloud workflow is to keep any resource on a recoverable status without the large fluctuations and release the idle resources to avoid the waste. It is inherent to cloud workflow to meet requirements R1 to R6 presented in Section 3.2.

To achieve sustainability, it necessitates to compose the abovementioned services to coordinate: the load balancing service checks the status of the resources and chooses an appropriate one for dispatching requests; the alarm service watches periodically the status of

[35] https://www.eucalyptus.com/
[36] https://www.openstack.org/

the resources and notifies the auto-scaling service when necessary; the auto-scaling service scales out by more resources to balance workloads and scales in for cost saving.

In essence, the coordination provisions a sustainable scalability to guarantee the scaling-out timely before the existing resources run out, or the scaling-in after the existing resources are idle. Supposing that the load balancing service, the auto-scaling service, and the alarm service are working at healthy or normal status, then, the elements that impact the coordination on provisioning a sustainable scalability are:

- The pending time: It is the time when a scaling-up action spends in instantiating the workflow enactment service components. If the time is too long, one or more existing resources may have become unrecoverable before new resources are ready. It will breach the sustainability of the scalability. To shorten the time, one strategy is that cloud workflow instantiates new resources as quickly as possible; and the other strategy is that cloud workflow prepares some instantiated resources somewhere which can be made available in shorter time once necessary.

- The decaying time: It is the time when cloud workflow spends in tackling a workflow application from the recoverable status to the unrecoverable status. If the time is too short, one or more existing resources would enter the unrecoverable status before the new resources are ready. It will breach the sustainability of the scalability. In theory, the higher configurations of hardware or software, the longer decaying time. For instance-intensive workflow, cloud workflow needs to prolong the time for keeping the existing resources recoverable through adding new resources.

- The time period: To monitor or control the resources, the alarm and load balancing service periodically check the status of the existing resources. If the period is too long, the services cannot be timely aware of status of the existing resources. Theoretically, the shorter the period is, the more sensitive the services are to status of the existing resources. If the time period is not appropriate, it may also breach the sustainability of the scalability. Cloud workflow needs to define the appropriate time period for checking.

- The predefined thresholds to estimate the system status: The thresholds can be used to compare to determine whether cloud workflow reaches a specific status or not. If the

selected thresholds are not accurate, it is difficult to determine whether the resources are unrecoverable or recoverable. Therefore, selections of appropriate thresholds will impact the sustainability of the scalability.

- The quantity of the available resources: A large quantity of the available resources can guarantee to balance more workload of workflow and may result in an idleness of part of the resources when workloads decrease. If the quantity is too small, it is more likely to breach the sustainability of the scalability due to the heavy workloads decaying the resources quickly. Cloud workflow needs to define the appropriate number of the resources to keep an efficient processing according to normal workloads of a workflow application.

- The quantity of the resources to scale out or in: The appropriate amount of the resources guarantees a stable performance of cloud workflow and economic usages when workload spikes of workflow are emerging. If the quantity is not appropriate, it will not be able to balance the surplus workload so that the incoming requests will not be able to be handled correctly. It will breach the sustainability of the scalability.

The elements above can cause the scalability of cloud workflow being unsustainable when they are predefined. We will further consider their impacts in cloud workflow and address them in the Chapter 6.

4.4 Discussion

The client-cloud architecture takes a great effort to provision scalable workflow services for large-scale instance-intensive workflow and eliminate the disadvantages of traditional WfMS architectures. Undoubtedly, a client-cloud WfMS has more powerful capacity, availability, reliability, and scalability than a C/S WfMS. Moreover, the scaling of the former is more economic and convenient than that of the latter based on hardware upgrade regardless horizontal or vertical scaling.

In contrast to the decentralised WfMSs, all workloads of workflow instances are evenly loaded on all virtualised workflow service components under the control of sustainable coordination of the services on the cloud side. Any component is not overloaded and never a bottleneck at any time. Therefore, the overall workload of the client-cloud architecture is more

balanced than the decentralised architectures. Moreover, cloud workflow provides an application provision service for integration with various external homogeneous or heterogeneous legacy applications or WfMSs. Another difference from the decentralised architectures which are logically or physically dispersed in many locations is that the communication costs of all the service components of the cloud-cloud model based architecture are greatly lower than that of the decentralised architectures because the cloud-cloud architecture is hosted in the cloud infrastructures which provide high performance communication services.

The novel service oriented client-cloud architecture is not based on the traditional computing paradigms but is completely innovated on the cloud computing paradigm and has many novel prominent characteristics. Moreover, it is designed for large-scale instance-intensive workflow applications, which is not well supported by most of current cloud workflow systems. Therefore, the research in this thesis is valuable for workflow communities.

4.5 Summary

In this chapter, we proposed novel client-cloud architecture for cloud workflow and further addressed its service components and the coordination between them in detail. Finally, we gave a discussion about the comparison between the cloud-cloud architecture and the traditional centralised or decentralised architectures and other cloud workflow systems.

Chapter 5
SwinFlow-Cloud: Functional Design based on Client-Cloud Architecture

We design a cloud workflow prototype system, named as SwinFlow-Cloud, based on the client-cloud architecture addressed in the previous chapter. In this chapter and next chapter, we will represent our novel design for the system. This chapter firstly represents the design of the functional aspects, while next chapter will represent the design of the non-functional aspects. In this chapter, Section 5.1 gives an overview of the build time and runtime functions. Section 5.2 addresses the build time functions, while Section 5.3 addresses the runtime functions.

5.1 Overview of the functional design

The functional design focuses on the fundamental and essential components in SwinFlow-Cloud. The functional areas include the build time functions and runtime functions. The former covers process modelling, verification, and simulation, while the latter discusses the workflow instance concept in SwinFlow-Cloud and addresses workflow enactment service, administration and monitoring.

The build time design addresses the process management functions for graphically defining, verifying and simulating a process. A process can be designed through adding, deleting, modifying the tasks and the relationship between them. Then the process can be verified on syntax and semantics for checking the possible errors or faults, and prompted to workflow designers. Any error can result in a failure of a process execution. Workflow

designers can amend the design according to the results of process verification. Then a well-verified process can be simulated in similar runtime environments for further checking the possible runtime errors or faults. If there are no errors in the process, it will be an executable process at runtime.

The runtime design focuses on the workflow enactment. A well-defined process can be transferred to the workflow enactment service to run. The workflow enactment service can instantiate a process instance (workflow instance) and launch, initialise, execute, terminate, or complete the process instance. The administration and monitoring function can watch the status of one or more process instances and control the execution of instances.

5.2 Build time functions

This section addresses the build time management in the functional aspects of SwinFlow-Cloud. For the build time functions, the traditional workflow reference model mainly focuses on the process management and proposes the process definition meta-model and primitives for supporting the management. Therefore, in our build time design, we also focus on the process management, including process modelling, verification, and simulation.

5.2.1 Process modelling

Process modelling refers to constructing or defining a process definition using a network of tasks with partial orders. In this thesis, we propose a simple process definition model based on the process definition meta-model and the primitives identified in the Terminology and Glossary proposed by WfMC [84], as shown in Fig. 5.1.

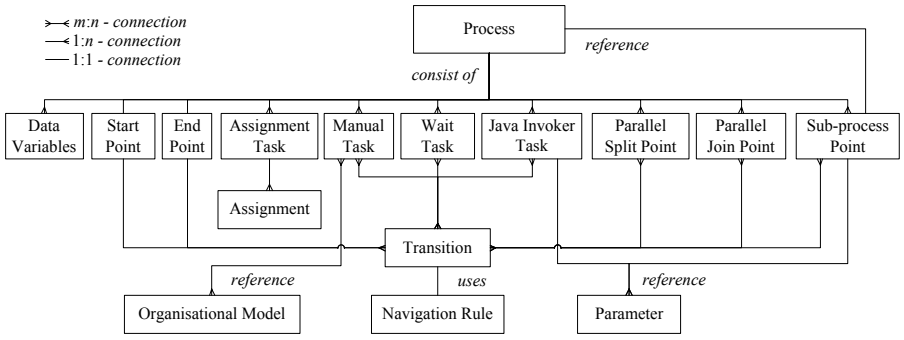

Figure 5.1 Process definition meta-model in SwinFlow-Cloud

5.2.1.1 Process definition

A process definition, i.e., process, consists of various types of the data variables, which may have initial values, including Integer, Float, Boolean, String, etc., task nodes, including Java API invoker task, manual task, assignment task, waiting task, etc., point nodes, including start point, end point, sub-process point, parallel split point, parallel joint point, etc., edges, i.e., transitions, and navigation rules.

The elements in the process are able to explicitly represent workflow meta-model elements of WfMC, such as workflow relevant data, activities, AND-Split, AND-Join, OR-Split, OR-Join, transitions, transition conditions, etc. The workflow relevant data can be depicted by the data variables. The activities can be depicted by the task nodes. The AND-Split can be depicted by the parallel split point. It can connect one input transition for a sequential relationship or more input transitions for a choice relationship and one output transition for a sequential relationship or more output transitions for a parallel relationship. The AND-Joint can be depicted by the parallel joint point. It can connect one input transition for a sequential relationship or more input transitions for a parallel relationship and one output transition for a sequential relationship or more output transitions for a choice relationship. No specific and individual OR-Split and OR-Join definitions are in the process. The start point is an entry point of a process and only connects one output transition for a sequential relationship or more output transitions for a choice relationship, while the end point is an exit point of a process and only connects one input transition for a sequential relationship or more input transitions for a choice relationship. Except the start point, the end point, the parallel joint point and the parallel split point, other point and task nodes are able to connect one input or output transition to represent a sequential relationship and are considered as OR-Split or OR-Join when they connects more output or input transitions. An edge connects two nodes to construct a directed graph. We also denote the edge as transition. The navigation rules attach on the transitions for navigating the process execution.

Furthermore, a process may have specific resource demand during execution due to domain-specific objectives. For example, a process needs powerful network bandwidth resources to support remote access. Hence, it should be assigned to a workflow enactment service with more powerful network bandwidth for execution. Cloud workflow can offer workflow enactment service on demand. Therefore, the process in SwinFlow-Cloud can be specified the particular resource demand when it is modelled. The load balancing service can

dispatch the requests of the process to the appropriate workflow enactment service for execution according to the resource demand specification.

5.2.1.2 Data variables

Data variables of a process store the data, which are initialised or generated at runtime, and are able to be used to construct the Boolean expressions in the navigation rules for controlling the process routing. We define seven fundamental types of variables to guarantee that the primary data can be stored in a process:

- Integer variable: It can store a positive or zero or negative integer number with a large range. The variable can be transformed to an Integer object in a programming language for calculation.

- Real variable: It can store a positive or zero or negative decimal number with a large range. The variable can be transformed to a Float or Double object in a programming language for calculation.

- Boolean variable: It can store a Boolean data, i.e., “TRUE” or “FALSE”. The variable can be transformed automatically to a Boolean object in a programming language for calculation.

- String variable: It can store a character string data, e.g., “abc”. The variable can be transformed to a String object in a programming language for operations.

- Binary object variable: It can store a complex type of data. The data can be a procedure object, a word file, a BMP file, or a JPEG file, etc. The variable is not used for operations but to support workflow users to modify the complex object in a specific user interface components.

- Date variable: It can store a date object and can be transformed to Date object in a programming language to support various date operations. The data can be represented as multiple date formats in a specific user interface components.

- Time variable: It can store a time object and can be transformed to Time object in a programming language to support various time operations. The data can be represented as multiple time formats in a specific user interface components.

5.2.1.3 Task and point nodes

Task nodes undertake actions to support the workflow. There are five types of task nodes:

- Java API invoker task: It can invoke a Java API in a Java JAR package on a remote host. The task node contains the detail for an invocation, which is provided in a tool agent, such as URL, JAR file name, package name, Java class name, invoked method name, input or output parameters and types, etc. We will discuss the tool agent in Subsection 6.2.3. The task node can implement an access to a remote Java API and deliver data through the information above. The design meets the integrity of workflow applications with Java applications.

- Web service invoker: It can invoke a Web service on a Web site. The task node contains the detail for an invocation, which is provided in a tool agent, such as URL, service name, data format definitions, input or output parameters, etc. The task node can implement an access to a Web service and deliver data through the information above. The design meets the integrity of workflow applications with Web services.

- Manual task: It can provision an interaction with workflow users. This task node can be attached with an external client application as a client interface and represent the form to workflow users for submission at runtime. It can also be bound with resource assignment strategies or rules for allocating this task to appropriate workflow users. Workflow users are able to submit the form to the task when they finish interacting with the form. The task can collect the data input by the users and transfer them to the process for use at runtime. The design meets the primary function of workflow applications.

- Assignment task: It can implement one or more assignments of a value from a data variable to another data variable in a process, that is, the value of a computation expression can be assigned to a data variable. The assignment offers flexible and independent data transfer in a process. The design enables the data transfer more flexible and explicit and improves the automation of a workflow application.

- Wait task: It can enable a process to be pending on a path at runtime through defining a waiting time period from a minimum of one millisecond to a maximum of a few days. The time period is a number which can be manually assigned at build time or

calculated from data expressions at runtime. The design enables a process to meet the necessary deferring requirements when some tasks exhausting a long time are executing.

Besides the task nodes, the point nodes are very important for modelling workflow and able to control the process routing. There are five types of point nodes in our process:

- Start point: This node is an entry point when starting an execution of a workflow instance. Every process has only a single start point in the structure. If a process does not have the start point, it is a structurally wrong process. Workflow enactment service will look up the start point to begin the execution of a workflow instance after the instance is launched and initialised. If a process is a sub-process of its parent process, the start point of the process can obtain the data transferred from the parent process and evaluate the data to the data variables of this process. The design of the start point reduces the computation of workflow enactment service to look up the entries of a process for starting an execution.

- End point: This node is an exit point when ending an execution of a workflow instance. Every process has only a single end point in the structure. If a process does not have the exit point, it is also a structurally wrong process. A correct process means that all executable paths are able to reach the end point. Ending a process means that all executing paths reach the end point. If a process is a sub-process of its parent process, the end point finally executed in the process can transfer data to the parent process. The design of the end point reduces the computation of workflow enactment service to determine whether a process can be ended.

- Parallel split point: This node implements the AND-Split function proposed in the Terminology and Glossary by WfMC. A process splits one path into multiple paths at this point without any conditions. The tasks on the split paths will automatically independently execute simultaneously. This parallel point commences a parallel task block in a process. Any parallel block must start at a parallel split point. The design offers a parallel capacity of a process to execute tasks concurrently.

- Parallel joint point: This node implements the AND-Joint function proposed in the Terminology and Glossary by WfMC. The multiple paths can be merged into one path

at this point without any conditions. This point completes a parallel block in a process. Any parallel block should end at a parallel joint point. However, if every split parallel path ends at the end point, there may be not a parallel joint point for ending the parallel block. The design offers a parallel capacity of a process to execute tasks concurrently.

- Sub-process point: It is dedicated to connecting a sub-process. The sub-process will be addressed in Subsection 5.2.1.4 in detail. Any process can be attached at this point as a synchronous or asynchronous sub-process at build time but a sub-process point can only attach a process as sub-process. The point can define the input parameters for inputting data to the sub-process and the output parameters for outputting data from the sub-process. The design enables processes to hierarchically represent a complex workflow.

As an edge with a directed arrow, the transition enables two nodes to be a directed graph through a connection. The node that connects the end of the transition without the arrow is the source node of the transition, while the node that connects the end of the transition with the arrow is the target node of the transition. The transition is the input edge of the target node and is output edge of the source node. In the nodes above, except that the start point has no input transition and the end point has no output transition, the other nodes have both input transition(s) and output transition(s). The navigation rule attached on a transition is a condition to control process routing. Not all transitions need a navigation rule. The output transition of the parallel split point has no navigation rule, that is, the target node of this transition will be executed without any conditions. The input transition of the parallel joint point has no navigation rule, that is, the parallel joint point will be executed without any conditions.

A process supports control-flow patterns: sequence, parallel split, synchronization, exclusive-choice, simple merge, and structured loop [1, 70]. The patterns are also basic structures of a process definition identified in the Terminology and Glossary proposed.

5.2.1.4 Sub-process

A process is denoted as a sub-process if it is attached in a sub-process point in a process. The process where the sub-process point exists is denoted as parent process. In theory, on one hand, a parent process can have infinite sub-processes through attaching them onto the sub-process points of the parent. On the other hand, a process can be attached on multiple sub-

process points as its sub-process. Therefore, a process with multiple sub-processes has a tree structure. The root of the tree is denoted as main process.

As mentioned in discussions of the sub-process point, all sub-processes in the main process have two execution patterns: synchronous execution with the parent and asynchronous execution with the parent. Synchronous execution means that the parent process will be pending automatically at the path where the sub-process point is after the sub-process launching until the sub-process ends. Asynchronous execution means that after the sub-process launching, the parent process will not be pending and continue at the path where the sub-process point is, but the parent process will check whether the sub-process ends before ending.

The communication between the main or the parent process and the sub-process is to transfer data between two data variables with the same data types existing in the two processes respectively. A parent process is able to collect the output data from its sub-process which synchronously executes with its parent, while it is not able to collect the output data from its sub-process which asynchronously executes with its parent because under the asynchronisation conditions, the parent process has completed when its sub-process is executing so that the parent cannot collect the output data from the sub-process.

The sub-process enables SwinFlow-Cloud to model hierarchically a workflow application with a top-down design.

5.2.2 Process verification

Process verification refers to syntax verification and semantics verification. The syntax verification is to check the build time syntax correctness of a process according to the process defined in the last subsection, including process completeness, such as a process must have an entry point and an exit point, a process cannot have only an entry point or an exit point, etc., and structure correctness, such as a parallel block must have both a parallel split point and a parallel joint point, the input transitions of a parallel joint point cannot have navigation rules, the output transitions of a parallel split point cannot have navigation rules, etc.

The semantics verification is to check the runtime semantics correctness of a process. For example, a task node has three output transitions with navigation rules, if no navigation

rule is met, it will result in a termination of a process at these transitions. The verification needs to debug a process for estimation at build time in order to predict possible errors.

5.2.3 Process simulation

Process simulation is to build a simulation environment to test whether a process can achieve the expected goals at runtime. The simulation can reduce risks of process design faults and verify the semantics correctness of a process at runtime.

The process simulation of SwinFlow-Cloud can obtain details of an execution of a process such as overall log details of the process at runtime, size of every node, details of every data variables, etc. It can debug a process with the operations step by step, such as launch, run, pause, resume, terminate, etc. Similar to the operations on a process, it can also operate a task node, such as run by one step, run by multiple steps, pause, terminate, skip, etc. The point nodes are only used to construct the directed graph and are not real task. In theory, the execution time of the point nodes is zero. Thus, the simulation does not provide pause, terminate, or skip operations at the point nodes.

5.3 Runtime functions

The runtime functions include the workflow enactment service, and administration and monitoring tools. We firstly address the design of the workflow instance before represent the design of the runtime functions.

5.3.1 Workflow instance

In SwinFlow-Cloud, a well-defined process definition can be instantiated to an executable workflow instance by the workflow enactment service. A workflow instance consists of data variable instances and task instances. The data variable instances are the real data objects which are interpreted from the data variables for computation or comparison. The references or specifications in the task instances are interpreted to bind with real resources or map onto APIs or systems. A workflow instance has a life cycle from the beginning to the end. The life cycle of a workflow instance is accompanied with state transformation, as shown in Fig. 5.2.

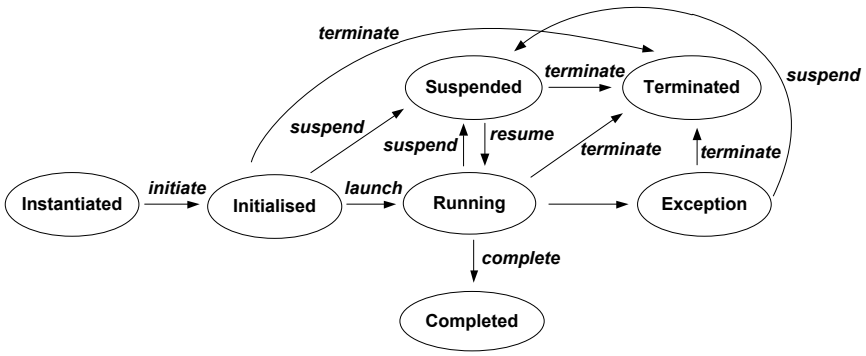

Figure 5.2 State transformations for a workflow instance

The transformations between states (represented by the arrows) take place in response to a phase ending. The basic states are:

- Instantiated: Workflow instance has been created, but the whole instance is not initialised yet, including any associated process or task state update and data variable.

- Initiated: Workflow instance has been initialised, including states of the process and all tasks initialised, and the initial value of all data variables computed, and possibly, the data triggering this workflow instance creation stored in the instance.

- Running: Initialised workflow instance has started execution and the start point is started. It is in the running state until the end point in this workflow instance has been fulfilled.

- Completed: Workflow instance has fulfilled the end point for completion and the some internal post-completion operations such as data collecting or transferring to the parent process will be performed if it is a sub-process, and then the workflow instance will be destroyed.

- Suspended: Workflow instance is quiescent and no task is started until the workflow instance has returned to the running state.

- Terminated: Execution of the workflow instance has been forcedly stopped before its normal completion due to some abnormal causes and then the workflow instance will be destroyed.

- Exception: Workflow instance has one or more errors in execution. This state is different from the suspended state: the execution of the workflow instance does not

stop on the exception state but the execution stops on the suspended state. This means that when multiple paths in the workflow instance are executing in parallel, any error in a task instance may only result in a stop on the path where the task instance is, but does not influence other paths.

Besides a workflow instance, its task instances have a life cycle. The following state transformation of a task instance depicts its life cycle, as shown in Fig. 5.3.

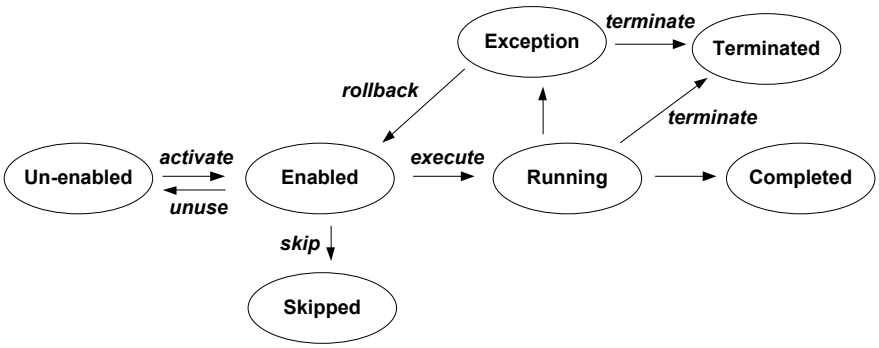

Figure 5.3 State transformations for a task instance

- Un-enabled: A task instance is created but has not yet been activated because the navigation rules on its input transitions have not been met.

- Enabled: A task instance is activated because all the navigation rules on its input transitions have been met but the task instance is not initialised yet. The enabled state may transform to the un-enabled state because any of the navigation rules on its input transitions are not met, or to the running state because the executing command drives, or to the skipped state because the skipping command drives.

- Skipped: A task instance is skipped and the target nodes of the output transitions of this task instance will be activated under controls of the navigation rules on them.

- Running: A task instance is executing and not non-interruptible during the execution but it can be forcedly terminated just as a Java process can be forcedly terminated by Java console. The state of the task instance can change to the terminated state. Actually any error in the task instance can result in its state from the running state to the exception state. If the task instance completes successfully, its state will transfer to the completed state.

- Completed: A task instance is completed successfully and activates the next task instance if any navigation rule on the output transitions has been met.

- Exception: A task instance has some errors during its execution. This state may be transferred to the terminated state through a terminating command, or to the enabled state through a rollback command.

- Terminated: A task instance has been terminated by force.

5.3.2 Workflow enactment service

The workflow enactment service in SwinFlow-Cloud offers a runtime environment which is able to support one or more workflow engines. Its functionalities concern with the workflow management and aim at achieving workflow goals. This service consists of the following modules: workflow engine, task transition engine, and navigation engine. In this subsection, we address the details of these function modules.

5.3.2.1 Workflow engine

A workflow engine controls execution and state transformations of a workflow instance and drives the workflow instance from one phase to the next: instantiation, initialisation, execution, completion and exception handling if necessary. It will be destroyed after completing the workflow instance. The workflow enactment service creates a workflow engine for every request of launching new workflow instance and transfers the associated data into the workflow instance and destroys the workflow engine after completion.

In the instantiation phase, the workflow engine creates a workflow instance referencing a workflow definition and assigns a new identity (ID) to the new instance. The workflow instance is specified as the instantiated states and stores ID of the workflow definition as a reference, workflow creator information and workflow instance creating timestamp. If the workflow definition has a main process and one or more sub-processes, the workflow instance will just have the main process because the sub-processes attached on the sub-process points may not execute during the execution of whole workflow instance.

In the initialisation phase, all the elements in the workflow instance will be initialised for supporting workflow enactment. All data variables are specified as null if no initial

expression, or as the calculated initial value if having an initial expression. All the task instances are initialised as the un-enabled state except that the start point is specified as the enabled state. The workflow engine creates three queues for each workflow instance: enabled task queue, running task queue, and exception task queue and further puts the start point into the enabled task queue.

If the main process in a workflow instance is a sub-process of another workflow instance, the former will be attached in the sub-process point in the latter and map the data variables in the former onto the input or output specifications in the sub-process point in the latter. Finally, the workflow instance is specified as the initialised state.

An initialised workflow instance can automatically launch under the control of the workflow engine or manually launch through a launching command with the state from initialised to running. The instance can also be terminated by a terminating command and suspended by a suspended command or due to any errors during the initialisation phase.

In the execution phase, the workflow instance is specified as the running state and de-queues the enabled task instances one by one to execute. Every enabled task instance de-queued from the enabled task queue will be specified as the running state and en-queues the running task queue. The workflow engine creates a task transaction engine for executing the running task instance. The transaction engine will be addressed in detail in the next subsection. Once the running task instance is completed, it activates the next task instances using the navigation engine and changes its state to the completed state and is de-queued from the running task queue. If the running task instance has any error during execution, it will be specified as the exception state and de-queues from the running task queue and en-queued to the exception task queue. Any exception task instance in a workflow instance will result in the state of the workflow instance from the running state to the exception state. If the workflow instance is a sub-process, the exception will change the state of the parent to the exception state until the state of the main process changes to the exception state.

In the exception handling phase, the workflow engine is on the quiescent status for waiting for the manual interference by workflow users.

In the completion phase, the workflow engine checks whether all task queues for the running workflow instance are empty. If the task queues are empty, that means that the

workflow instance has completed and is specified as the completed state. The workflow engine will be destroyed after the workflow instance is completed.

5.3.2.2 Task transaction engine

The task transaction in SwinFlow-Cloud guarantees the completeness of the execution of a task instance and the data consistency. It offers an interface framework to a task instance for execution. The framework consists of three primary operations:

- Begin: A transaction of a task instance collects the data to support the execution and temporarily stores the data for recovering when rolling back.
- Commit: A transaction executes the task instance non-interruptedly. If the execution has any exception, the transaction will terminate through throwing an exception and rollback.
- Rollback: A transaction rolls back and recovers all data to the initial status of the transaction.

The interface framework for supporting the execution of a task instance is extensible for developing the new task in SwinFlow-Cloud. If new task meets and inherits the meta-model of the process in Subsection 5.2.1, the task instance of the task can be supported by the task transaction.

The task transaction in SwinFlow-Cloud executes under the control of the task transaction engine. The transaction engine can lock the task instance and the associated data to guarantee three operations run sequentially and correctly before a task transaction begins and unlock the task instance after the commission of the transaction. It can automatically terminate, roll back and record the exceptions when any error occurs during the operations. The transaction engine can be destroyed after a transaction terminated or completed. If a transaction engine is destroyed due to termination, then the workflow engine can create a new task transaction engine for each transaction execution when a task instance restarts.

5.3.2.3 Navigation engine

The navigation engine can navigate the execution of the workflow instance to the transitions of which the value of the navigation rule is true. On one hand, it is created to check the input

transitions of a task instance before a task transaction begins, or to calculate the navigation rules on the output transitions of a task instance after the task transaction is committed. On the other hand, it is created to recover the input transitions once the task transaction rolls back.

5.3.3 Workflow administration and monitoring

The workflow administration and monitoring in SwinFlow-Cloud enables workflow users to monitor and control the execution of the workflow instances. The component consists of three modules:

- Workflow system administrator: It is the control panel of the workflow enactment service and can start, restart, stop the service and further monitor the performance of the service, including the utilisation of underlying resources of the service, such as CPU, memory, network traffic, I/O, etc., and application-level resources of the service, such as thread pool, connection pool, session number, etc.

- Workflow instance searcher: It is able to search workflow instances through specifying the conditions such as ownership, workflow definition reference, creator, launch time, etc. The search results represent instance identity, instance name, version number, instance creator, launch time, size in memory, instance current state, last update time, and reference of workflow definition.

- Workflow instance administrator: It is similar to the workflow process modelling tools. It can represent graphically the current and historic execution details of a workflow instance and provide the flexible commands to manually interfere the execution, such as suspending, terminating, resuming an instance, or skipping, rolling back, terminating a task instance, etc.

5.4 Summary

In this chapter, firstly, we gave an overview of the functional design in SwinFlow-Cloud. Secondly we presented the build time function, i.e., process management, including process modelling, verification, and simulation. Finally, the runtime functions were presented for

workflow enactment service, including workflow engine, task transaction engine, and navigation engine, and workflow administration and monitoring.

Chapter 6
SwinFlow-Cloud: Non-functional Design based on Client-Cloud Architecture

This chapter presents the non-functional design of SwinFlow-Cloud based on the client-cloud architecture proposed in Chapter 4. Section 6.1 gives an overview of the non-functional design. Section 6.2 addresses the design of build time functions. Finally, Section 6.3 addresses the design of runtime functions.

6.1 Overview of the non-functional design

The system design focuses on the non-functional areas in the workflow accompaniment tools or the workflow relevant services as depicted in Fig. 4.1, including the build time and runtime functions. The former covers version management, organisation management, tool agent management, cloud workflow relevant service definition functionalities, administration and monitoring functionalities, while the latter covers cloud workflow relevant service as defined in Section 4.1.

The build time design addresses the management functions which can define the relevant specifications for cloud workflow. Version management efficiently facilitates a collaborative process design. Organisation management constructs a hierarchical model to integrate all perspectives of workflow. Tool agent management bridges the external applications and cloud workflow for invocations. Billing management defines the system-level or application-level budget plans for control the costs of the utilisation of cloud services or workflow service resources. WfSMI management is able to configure WfSMIs supported by

the cloud infrastructures. Alarm management represents the critical system resources and provide choices to the estimation model. And auto-scaling management defines scaling-out or scaling-in strategies for the scaling actions in the auto-scaling service.

The runtime design represents the administration and monitoring functionalities, the workflow relevant service functionalities, and further proposes two core estimation models to support coordination between the services. In particular, billing administration and monitoring functionality can monitor utilisation of budget plans and generate a report for analysing of the utilisation. WfSMI administration and monitoring functionality can monitor utilisation of a WfSMI and analyse the performance influence of the instantiation of the WfSMI. And alarm administration and monitoring functionality can monitor utilisation of resources and optimise the specifications of utilisation.

The design of the workflow relevant services includes the module structures in the services and their functionalities. Billing service can interpret the plans predefined in the cloud workflow relevant service definition tools on the client side and generate many rules to monitor the utilisation of various cloud services or workflow service resources. WfSMI service can manage the image configuration defined in the cloud workflow relevant service definition tools and support the generation of the images through invoking APIs of cloud infrastructure. Alarm service can estimate status of all workflow enactment services and give an alarm notification to auto-scaling service for taking scaling action when the status has been under loaded or overloaded. In this service, we propose an alarm estimation model for quantifying status and utilisation of the specific resources to estimate whether SwinFlow-Cloud is overloaded or under loaded. Load balancing service can check status of each workflow enactment service and choose an available one to dispatch the request. Auto-scaling service can estimate resource demands and create scaling engines to implement scaling actions. In this service, we propose a scaling estimation model to calculate resources that need to be scaled. To efficiently support a sustainable coordination, we further discuss the elements which influence the coordination of the services above and reveal constraints between the elements and propose principles for a sustainable coordination.

6.2 Build time functions

In this section, we address the design of build time functions. The build time functions are to manage the models or specifications to the runtime functions. The models include organisation, version, and tool agents; while the specifications include budget plans, WfSMIs, resource utilisation metrics, and scaling strategies, etc. The design covers organisation, version, tool agents, billing, WfSMI, alarming, and auto-scaling management.

6.2.1 Version management

SwinFlow-Cloud introduces the concepts of version control architecture to provide a flexible approach to manage the model changes or evolution, such as processes and WfSMI and so on. The architecture consists of the version state and control operations, as shown in Fig. 6.1.

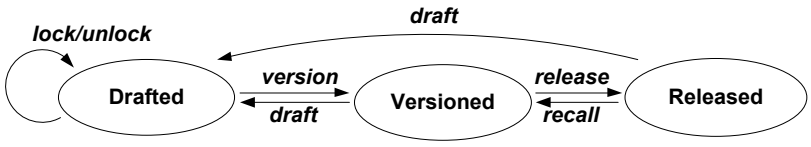

Figure 6.1 Transformation between the version states

In Fig. 6.1, there are three version states which can be transformed from one state to another through control operations:

- Drafted: This is an editable version of a model with a version number. When a model is on this version, any workflow designer can exclusively modify the model through a lock operation. After the modification, the designer will unlock the model to share it to other workflow users. The lock or unlock operations will be addressed below. This version enables multiple workflow designers to collaborate to design or modify a model.

- Versioned: This is a non-editable version of a model, which is created from the draft, with a version number. When a model is on this version, it is read-only and any workflow designer cannot modify it. This version is also defined usually as a fixed version.

- Released: This is a version of a model which is possible for workflow enactment service to execute. The version has no new version number and inherits the version number of the fixed version.

Version management includes multiple version operations:

- Lock: Locking means that a workflow user exclusively gains the edit privileges on an unlocked draft version of a model through a specific operation so that other users cannot gain the edit privileges any longer through any operation. The locking operation does not change the state of the draft version but change the properties of the model such as the designer, or update time, etc.

- Unlock: Unlocking means that a workflow user waives the edit privileges, which are gained from the locking operation, on a locked draft version through a specific operation and other users can gain the edit privileges through the locking operation. The unlocking operation does not change the state of the draft version but change the properties of the model.

- Version: This operation is to fix a draft as a new version with a version specific number for a process to protect the edited model from being modified by any other one, that is, it changes a model from the drafted state to the versioned state and automatically or manually creates a new version number, for example, the automatic version is like “2014-02-13 16:56”, or the manual version is like “v1.0”.

- Draft: This operation is to create a draft from a fixed version or a released version with a timestamp version number which is generated automatically. It changes a model from the versioned state or the released state to the drafted state.

- Release: This operation is to create a released version from a fixed version without new version number. It changes a model from the versioned state to the released state. This operation is to publish a model to runtime environments.

- Recall: This operation is to recall a released version and changes a model from the released state to the versioned state. The design is to provide an approach to recall or delete a model from the runtime environments due to some specific considerations.

The version control management in SwinFlow-Cloud facilitates collaboration in modelling and provides an approach to trace the evolution of model designs. For the convenience of operations, the version management is not designed as a standalone management module in SwinFlow-Cloud but the above operations are integrated into the other management functionalities.

6.2.2 Organisation management

The organisation structure model is an important perspective in workflow management although it is not explicitly addressed in the traditional workflow reference model. Organisation is the owner of a WfMS and a workflow application. In SwinFlow-Cloud, we design organisation model as a hierarchical forest structure, denoted as organisation space, and organise all workflow perspectives into an organisation space. In the space, various workflow users can individually participate in managing the workflow associated models or components without the needs of concerning with other users. The model enables SwinFlow-Cloud to contain many organisations and administrate multiple inter-organisation workflow applications on one platform. It facilitates SwinFlow-Cloud to cost-effectively deliver the workflow services to various organisations with a payment model. As the backbone of SwinFlow-Cloud, the organisation management is undoubtedly a critical function and covers organisation structure management, user management, and authorisation management. In the next subsections, we address these functionalities.

6.2.2.1 Organisation structure

The organisation structure is a tree model. The nodes of the tree model are constituted by organisation, division, department, project team, position, role, user, etc. The root node is the organisation and the leaf node is the user and the non-leaf nodes are the division, the department, the project team, the position, the role, etc.

- Organisation: Organisation is the largest abstract unit of an organisation structure in SwinFlow-Cloud to describe a social company, corporation, manufacturer, government, institution, branch, etc. The units usually have independent operations, collective goals, and financial systems and are constituted directly by divisions, departments, project teams, positions, roles, and users. But it does not describe a (corporate) group which owns one or more member organisations. That is to say, there

is not an organisation which contains other organisations. We design a group as a general organisation in SwinFlow-Cloud but its organisation structure does not describe the structures of the underlying member organisations.

- Division: Division is a large abstract unit in an organisation without independent financial systems. It can be constituted by departments, project teams, positions, roles, and users. The division can also be denoted as branch.

- Department: It is a middle abstract unit in an organisation and is constituted by one or more positions, project teams, roles, and users.

- Project team: It can be considered as a virtualised department in an organisation for a specific project. A project team is always a temporary department. The team consists of one or more roles and users.

- Position (work position): It is a place with specific responsibilities or duties to complete a job, such as "manager", "accountant", "clerk", etc. The position can be built in the division and/or the department and have one or more users. That means that the users on this position have same responsibilities or duties.

- Role: Similar to the position, the role is also a place with responsibilities or duties. The role is only built in a project team to undertake a specific job and can have multiple users.

- User: It is the staff who constitute an organisation. A user must be built on a position or a role.

The organisation model provides a support for workflow resource assignment, that is, workflow enactment service determines an appropriate resource to complete a manual task when multiple resources are available for this task.

Organisational structure management supports individually defining or editing an organisational structure, a divisional structure, a departmental structure, and a project team structure in a graphical editor.

6.2.2.2 User management

User is a direct workflow participant or administrator. User management in SwinFlow-Cloud refers to user information, position assignment, and user group management, etc.

- User information management: This component collects, updates, and removes a workflow user primary information, such as name, identity number, gender, phone number, email, address, duty status (e.g., on duty, on leave, quit, etc.), and work status (e.g., available, slightly busy, busy, very busy, away, offline, etc.). The primary information is used to support resource assignment in workflow management. If a user has been participating in workflow applications such as designing, administrating, launching a workflow instance, it means that the user has been referenced by other components. Then the user will not be able to be removed but only be written off using a flag because removing may result in data inconsistency. The component is able to import or export the information from or to the external system dedicated to user management.

- User work management: The component, which is associated with the position or role in organisational structure management, administrates and monitors the user working on a position. It can trace work history and assess work performance of users according to the position capability or duty requirements.

- User group management: This component creates, modifies, and removes a group for containing multiple users and can add or remove one or more users in a group. It is designed for authorisation management to grant access permissions to users. A group of users has same permissions to operate SwinFlow-Cloud. Moreover, a user can be added in multiple user groups and multiple level authorities.

6.2.2.3 Authorisation management

Authorisation, which is a security consideration in SwinFlow-Cloud, refers to the operation permission of an operator on a workflow component. The operator is the user group which is built in user management. The workflow component means all workflow system functional or non-functional modules in SwinFlow-Cloud. The operation permission refers to granting an action in a workflow component. The action is an atomic operation, for example, creating, modifying, deleting a process, etc.

The management component is to administrate all permissions on the workflow components and authorise users using the permissions for controlling access.

6.2.3 Tool agent management

The tool agent is an invocation proxy for Java API invocation task and other application invocation tasks which uses a unified approach to access the external applications residing in the various remote systems including Web server, other WfMS, FTP server, Amazon S3, Eucalyptus Walrus, and so on. The tool agent model is shown in Fig. 6.2.

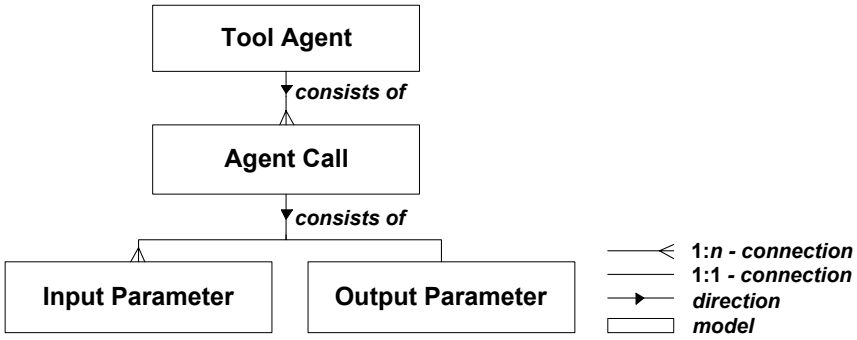

Figure 6.2 Tool agent model

A tool agent contains the information of the location and the authentication including URL, port, username, password, etc. It can carry multiple remote calls, denoted as agent call. The agent call defines class path name, method name, and parameters. The parameters are composed of input parameters and output parameters. Some agent calls may have no output parameters. The parameters consist of parameter name and data type, i.e., integer number, real number, character string, etc.

The tool agents are organisation assets in SwinFlow-Cloud and are administrated by the tool agent management in the spaces provided by owner organisations. Thus the management supports specifying a security access key to avoid the arbitrary invocations.

6.2.4 Billing management

Billing management aims at controlling the usage costs of the workflow services to meet the budget. Because the SwinFlow-Cloud is a cloud application, the management has two aspects:

cost management of cloud workflow in the cloud infrastructure and cost management of workflow applications in cloud workflow.

For the former, we design a system billing manager for owners of SwinFlow-Cloud. The manager can collect utilisation and price policies of the used cloud services through APIs of cloud infrastructures, such as computation service, storage service, security service, network service, or Web deployment service and so on, and then support the owners of SwinFlow-Cloud to make an annual system-level budget plan on the basis of the price policies of the cloud services. The budget plan is composed of cost budgets of the cloud services used in SwinFlow-Cloud. The cost budget consists of cloud service name, service utilisation metrics, price, currency, e.g. AUD, USD, RMB, etc., financial year, budget amount, cloud infrastructure name, and alarm thresholds etc. The metrics are different among cloud infrastructures, for example, for the data storage in AWS, 1 GiB per month[37], for Internet data transfer in IBM SmartCloud, 1 GB per week, 1 GB per month, 1 TB per month[38], etc. The thresholds are used to give an alarm when current utilisation of a service reaches or exceeds the budget. A designer may design multiple plans to support scalable SwinFlow-Cloud and adjust or switch to an appropriate plan to meet the changing costs of the cloud services. Furthermore, the system billing manager can also define a report template to represent the utilisation of the cloud services. The report is a critical reference for cost analysis on the utilisation of cloud services in SwinFlow-Cloud.

For the latter, we design a workflow application billing manager for owners of workflow applications. Here, SwinFlow-Cloud can be considered as workflow management infrastructure in cloud to provision workflow services. The users of the manager include two roles: one is the owners or administrators of SwinFlow-Cloud; the other is the owners or administrators of the workflow applications residing in SwinFlow-Cloud. For the two user roles, the manager offers the system-level and application-level functionalities.

The system-level functionalities include:

- Making the pricing policies for the workflow applications. Here, the policy is similar to the prices made by the cloud infrastructures. It consists of the workflow service name, e.g., “workflow enactment”, “workflow storage”, and service utilisation metrics,

[37] http://aws.amazon.com/pricing/
[38] http://www-935.ibm.com/services/ca/en/igs/cloud-development/estimator/Tool.htm?cfg=ca-en

e.g., $0.01 per workflow instance, 1,000 task instances per month, and price and currency and cloud infrastructure name.

- Specifying various plans for charging utilisation of workflow services offered by SwinFlow-Cloud. The plan includes multiple billing items on the basis of the pricing policies and offers of some bonus credits for promotions and limits or regulations of the usages of workflow services.

The application-level functionalities include:

- Making the annual application-level budget plans for the usage of workflow services according to pricing policies of SwinFlow-Cloud to meet the specified overall budget.

- Applying the plans offered at the system-level billing manager to charge consumptions of workflow services in a workflow application to meet the application-level budget plan. This is an express approach for the workflow applications using billing services and facilitates the workflow applications to migrate to cloud but not an appropriate approach because the billing items may not meet the usage requirements of the workflow application.

- Defining plans to customise the consumptions of workflow services in SwinFlow-Cloud to meet specific requirements of a workflow application. The plan consists of strategies to regularise the usage. The strategy describes the usage limit of the workflow services according to the price policies, for example, the usage of an enactment service per year is less than 200 dollars, or the 10G usage of storage services per year is not more than 50 dollars, and the information about the workflow application owners, the workflow applications, the bank account, the warning messages if exceeding the budget, etc.

Furthermore, the billing management functionality offers a charge calculator for estimating consumptions of workflow services. The calculator can represent all pricing policies defined in system-level functionality as a report. The designers can input the budget details and gain a simple estimation for the consumptions of the workflow services. The estimation can provide a reference for making a budget plan.

6.2.5 WfSMI management

WfSMI management offers an approach to manage query, create, update, discard, etc. WfSMIs are able to be instantiated to a workflow enactment service that meets the budget plan.

A WfSMI is a software configuration package with a unique image identity in SwinFlow-Cloud. The configuration includes the hardware and software aspects. The hardware aspect refers to CPU, memory, disk, and network card, etc., while the software aspect refers to operation system infrastructure such as Microsoft Windows, CentOS Linux, Unbuntu Linux, or Red Hat Linux, etc., and the application servers such as Apache Tomcat, etc.

The WfSMI has a life cycle from creating, updating to discarding. In the life cycle, the image may be frequently updated for satisfying the changing requirements. Therefore, SwinFlow-Cloud introduces version control, as addressed in Subsection 6.2.1, to manage changes of WfSMIs. Under controls of version management, designers can modify the configured WfSMIs and trace the modifications. The WfSMIs with the different version numbers can meet the requirements with more granularities. The WfSMI service of SwinFlow-Cloud can use a WfSMI configuration to generate an image through invoking APIs of cloud infrastructures. The image is an image file, which is stored in cloud and able to be instantiated a workflow enactment service to cloud infrastructures, and labelled with a unique image identity and a version number.

For creating, the management can represent the budget-met underlying hardware and software configurations obtained from cloud infrastructures and the scaling demands obtained from the auto-scaling service. Designers choose the appropriate configurations, i.e., meeting the budget plans and the scaling demands, submit them to the WfSMI service to generate a virtual machine on the cloud side. The procedure is similar to configuring a real computer. Then the designers can launch it to install an application server and deploy a standalone workflow enactment service. After the installation and deployment, the designers can test and verifies the availability of workflow enactment service component. If the testing succeeds, a new image with a unique identity and a version number will be generated through invoking APIs of cloud infrastructures. If the testing fails, the installation and deployment will be repeated again until succeeded.

For updating, the management can represent details of all existing WfSMIs. The designers can modify configurations of the WfSMIs, submit them to the WfSMI service on the cloud side for regenerating a virtual machine, reinstall an application server, and redeploy a standalone workflow enactment service component. Like creation, the designers need to test and verify the configurations of the modified WfSMIs. If the testing is successful, a new image will be generated with a new version number.

For discarding, the management can represent details of all existing WfSMIs in a list and enables to select one to specify a deprecated flag then submit it to the WfSMI service for discarding. If the image is in use for instantiations, the WfSMI service will not mark the flag until it is not in use. The deprecated images cannot be used for instantiations any more.

6.2.6 Alarm management

The alarm management can enumerate utilisation of system resources which may significantly impact on performance of a workflow enactment service component, i.e., performance bottlenecks, and specify them to the alarm service for supporting a comprehensive estimation. The estimation model in the alarm service will be addressed in Subsection 6.3.6 in detail.

The resources in a workflow enactment service component can be categorised into system-level, platform-level and application-level resources. The first covers the resources associated with underlying hardware, such as CPU, physical memory, disk, and network, and so on; the second covers the runtime platform resources, such as Java heap memory; the third covers the internal resources in a workflow enactment service component, such as the workflow instance thread resource pool, task instance thread resource pool, database connection pool, and session number and so on.

Obviously, the available capacity of the upper-level resources is determined by their lower-level counterparts. For example, size of the physical memory determines maximum size of Java heap memory, and heap size mostly determines maximum thread number that a JDK platform can contain. Hence, we can look up utilisation of critical system resources to make estimation. However, merely monitoring the utilisation of critical resources is not enough to accurately give an estimation of workflow executions because the utilisation only represents rough performance of the resources in a workflow enactment service component. For example, suppose the physical memory size is 8 GB and the Java maximum heap memory size

is specified as 4 GB, if a current occupation of a workflow enactment process in Java is 3.5 GB, then, for total physical memory size, the utilisation ratio is 43% (i.e. 3.5/8), but for the Java heap memory size, it is 87.5% (i.e. 3.5/4). The 43% is not accurate because we should use 4 GB to calculate the utilisation of the workflow enactment process due to the heap size that the process can use is 4 GB not 8 GB. Certainly, when we specify the heap memory as 5.5 GB, the utilisation ratio will be 63% (i.e. 3.5/5.5). This is important for the auto-scaling service to take actions, if the utilisation ratio is 87.5%, the service most likely needs to take actions to scale out for balancing the future incoming loads; and if it is 63%, it may not need to take actions.

To support cost-effective workflow executions, it requires that the alarm service accurately estimates workflow resource utilisation. Moreover, accurate estimation can facilitate prediction for future resource utilisation in the auto-scaling service.

Here, for simplicity, we only consider the same resources and their capacities and their utilisation in each workflow enactment service. In this chapter, we will formally discuss and model the resources, capacities and utilisation. For conveniently exploring and searching, we give a notation index which is defined and used in this thesis in appendix.

Suppose there are *m* workflow enactment service components in SwinFlow-Cloud and *n* resources in each component. For any resource, its utilisation at any moment *t* in a time period *T*, $t \in T$, $T = \{0,1,...\}$ and capacity are denoted as $u(t)$ and c. Thus, we define an utilisation metric for each resource in the components which can be used to estimate the system status.

The metric is a ratio at t, denoted as $r(t)$, which is the rate of $u(t)$ and c, with a range of values from 0 to 1, i.e., $r(t) \in [0, 1]$:

$$r(t) = \frac{u(t)}{c}$$

Here, we enumerate nine representative resource utilisation metrics:

- CPU utilisation $r_{cpu}(t)$: It is a ratio of a summary of the CPU time occupations u_{cpu} (t) of all workflow engines and task transaction engines at *t* to total CPU time occupations c_{cpu}. r_{cpu} (t) can reflect the total CPU usages in workflow enactment service component. It rises up gradually with the number of the engines increasing; otherwise, it declines with it decreasing.

- Memory utilisation $r_{mem}(t)$: It is a ratio of the summary of the occupied memory size u_{mem} (t) of all workflow engines and task transaction engines at t to the maximum memory size c_{mem}. r_{mem} (t) can reflect the total memory occupations in a workflow enactment service component. It rises up with the number of the engines increasing; otherwise, it declines with it decreasing.

- Network in utilisation $r_{net\_i}(t)$: It is a ratio of the summary of the incoming bytes u_{net_i} (t) of all the engines at t to the maximum incoming bytes c_{net_i} of the network device, such as network card. If there are multiple network devices, r_{net_i} (t) is a mean of all ratios of the individual network in utilisation. The ratio identifies the workload of network incoming traffic. It rises up with the incoming bytes increasing; otherwise, it declines with incoming bytes decreasing.

- Network out utilisation $r_{net\_o}(t)$: It is a ratio of the summary of the outgoing bytes $u_{net\_o}(t)$ of all the engines at t to the maximum outgoing bytes c_{net_o} of a network device, such as network card. When there are multiple network devices, $r_{net\_o}(t)$ is the mean of all ratios of the individual network out utilisation. The ratio reflects the workload of network outgoing traffic. It rises up with the outgoing bytes increasing; otherwise, it declines with the outgoing bytes decreasing.

- IO utilisation $r_{io}(t)$: The IO can be considered as any IO device except for the network device, such as disk, printer, scanner, sensor, sound card, etc. $r_{io}(t)$ is a ratio of the write/read bytes $u_{io}(t)$ of a IO device at t to the maximum read/write bytes c_{io} of the IO device. It rises up with the bytes increasing; otherwise, it declines with the bytes decreasing.

- Workflow engine thread pool utilisation $r_{proc}(t)$: The thread pool is a critical resource for many software applications. If the available threads in the workflow engine thread pool are used up, the new workflow instance must enter a waiting queue. Thereby the length of the queue is too long and workflow instances waits for too long, the queue may overflow or discard the incoming workflow instances. $r_{proc}(t)$ is a ratio of the waiting workflow instances $u_{proc}(t)$ at t to the length c_{proc} of the wait queue of the workflow engine thread pool. It rises up with the instances increasing, that means that the workload will rise; otherwise, it declines means that the workload will reduce.

- Task transaction engine thread pool utilisation $r_{task}(t)$: A workflow instance may have multiple task instances hence the number of task instances is more than the workflow instances after instantiation. The task transaction engine thread pool is another critical resource. $r_{task}(t)$ is another ratio of the number of the waiting task instances $u_{task}(t)$ at t to the length c_{task} of the wait queue of the task transaction engine thread pool.

- Connection pool utilisation $r_{pool}(t)$: The connection pool refers to the resource pool that manages a WfMS connecting to the external application or platform such as database, object library, cloud infrastructures, etc. The pool keeps the active connection objects for WfMS utilisation. If the available active connections are used up, the execution of workflow instances or task instances may be suspended and enter a waiting queue for waking up. $r_{pool}(t)$ is a ratio of the number of the waiting instances $u_{pool}(t)$ at t to the length c_{pool} of the wait queue.

- Session number utilisation $r_{session}(t)$: The session number is a limited resource for keeping the active connections between the client side and the workflow enactment service on the cloud side. $r_{session}(t)$ is a ratio of the session number $u_{session}(t)$ at t to the maximum session number $c_{session}$ which a workflow enactment service can offer. If it rises up with the session number increasing, it means that the workload is increasing; otherwise, the number declining means that the workload is reducing.

Thus, for n resources, $r_i(t)$, $i \in \{cpu, mem, net_i, net_o, io, proc, task, pool, session, ...\}$ or $i = 1..n$ represents the status of the resources in one workflow enactment services. Then, in m workflow enactment services, r_i is denoted as $r_{ij}(t) \in [0, 1]$, the utilisation $u_i(t)$ is denoted as $u_{ij}(t)$, the capacity is denoted as c_{ij}, $i = 1..n, j = 1..m$. The capacity of a resource is generally a constant in a service, Here, the resource configurations of one service may be different from those of the others. The capacities of each resource of the service are also different from those of the others. For example, for memory resource, it is 8GB in one service, but it is 16GB in another service. Obviously, the capacity of the former is different from that of the latter. Hence, we denote the capacity as c_{ij}.

The maximum value *1* means that a resource is fully loaded; while the minimum value *0* means that the resource is idle. Hence, *0* and *1* are two extreme cases. However, they cannot identify the status of the resource with finer granularities. We design to specify more thresholds in the range to discretise the continuous range values into multiple states for

estimating the status of the resources in the workflow enactment service. For example, in a range [0, 1], 0.2 is a threshold, 0.5 is another threshold, [0, 0.2] means that the utilisation is on an idle status, [0.2, 0.5] means that the utilisation is on a normal status. The threshold can be refined or extracted according to the experimental or historical data. In this thesis, the thresholds are gained by the experimental data. In the future work, we will research to extract the thresholds based on historical data.

6.2.7 Auto-scaling management

Auto-scaling management provides an approach to manually manage the auto-scaling specifications. The specification can be created manually or generated automatically by the auto-scaling service. It includes the scaling-out and scaling-in strategies. The former contains policy identity, image identity, scaling-out number, and deprecated flag; while the latter contains policy identity, image identity, scaling-in number, and deprecated flag. The policy identity is a unique global number. The image identity refers to the unique identity number of a WfSMI without a deprecated flag. The scaling-out or scaling-in number refers to the number of the workflow enactment service components which are scaled out or in. It is an integer number of the workflow enactment service components in the available group of the load balance service. A strategy is to be used in the scaling actions to scale out or in by the specified number to balance workloads. If a strategy has the deprecated flag, then it cannot be used in scaling actions.

The management includes querying, creating, updating, and discarding the scaling-out or scaling-in strategies. For querying, it supports conditional queries for scaling strategies. If an image identity in a scaling strategy has a deprecated flag, then the image will be not able to be used and the strategy will also be marked with a deprecated flag.

For creating, it can obtain and represent all the available images with the configuration manifests. The designers can select an appropriate image and specify the scaling-out or scaling-in number to create a new strategy with unique identity.

For updating, it can support modifying the primary information in a strategy, including the image identity, the scaling-out or scaling-in number, and the deprecated flag. The modification enables the strategy to adapt new demands.

For discarding, it can mark a strategy using the deprecated flag. If the strategy is in use of scaling, it will not be marked until it is not in use. The deprecated strategy can be removed from the auto-scaling service.

6.3 Runtime functions

In this section, we address the design of the runtime functions. The design, covering all the features of cloud workflow listed in Subsection 1.1.3, includes the administration and monitoring functionalities and the services which support the alarm and auto-scaling mechanisms. The administration and monitoring functionalities are able to administrate the utilisation of the specifications defined at build time. The services collaborate to provision the sustainable scalability to elastically support instance-intensive workflow applications.

For quantitatively analysing the resource demands, we propose the alarm estimation model to measure utilisation of resources in the workflow enactment services and the scaling estimation model to calculate the number by which the auto-scaling service needs to scale out or in.

6.3.1 Billing administration and monitoring

Billing administration and monitoring manage the budget plans to control consumptions of system resources. The functionalities include:

- Administrating two aspects of utilisation costs: the utilisation costs of cloud workflow in cloud infrastructure and the utilisation costs of workflow applications in SwinFlow-Cloud.
- Generating automatically a report of utilisation costs. The report describes billing items, charge details, statement period, and billing date, etc.
- Giving a notification on utilisation which reaches or exceeds the budget. The notification is implemented through comparing specified thresholds and real time usages of the workflow enactment services. The notification may be by email or short message.

We further design a cost analyser which is able to illustrate the consumption of the workflow enactment services and compare costs with the annual budget plans. The analysis can be used to support the decisions for making new annual budget plans.

6.3.2 Billing service

The billing service is to interpret various plans or policies defined at build time and control the cost according to the interpretation to meet the budget.

The interpretation is to generate the runtime rules to constrain the utilisation of various resources regardless those of the cloud infrastructures or cloud workflow. The rule contains budget, thresholds, utilisation metrics, charging time period, etc.

The service can watch and calculate any operation in SwinFlow-Cloud and accumulate costs of utilisation then compare the costs with the responsible rules. The billing service schedules the rules for improving access efficiency according to access frequency.

6.3.3 WfSMI administration and monitoring

WfSMI administration and monitoring manage instantiations of the WfSMIs created at build time and can analyse the influence of the instantiations to the overall system performance.

The instantiation is that a WfSMI is concurrently instantiated to multiple available workflow enactment services under control of cloud infrastructures. Hence, the detail of the instantiation contains begin and end timestamp of the instantiation, number of the created workflow enactment service components, timestamp of each component being created, timestamp of each component being available, pending time of each component, timestamp of each component being stopped, timestamp of each component being terminated, life period of each component, etc. An instantiation is a reference of a WfSMI to balance the workloads and can make an influence in the overall performance of SwinFlow-Cloud. Thus, every instantiation is recoded in the instantiation history of the WfSMI.

The functionalities of the WfSMI administration and monitoring include:

- Administrating the WfSMIs. The administration operations include starting to use a released WfSMI, stopping to use a released WfSMI with a deprecated flag, calling

back a released WfSMI, etc. The operations enable administrators to manually manage the WfSMIs for support the auto-scaling service.

- Monitoring the instantiations of the WfSMIs. It can explore historical details of the instantiations of the WfSMIs, illustrate life cycle of the WfSMIs and the instantiated workflow enactment services, and trace the instantiation frequency, instantiation time period change, and instantiation number changes in an annual budget plan.

- Analysing influence of the instantiations to performance of SwinFlow-Cloud. The analysis is to illustrate the performance change after every instantiation, and give an alarm when the instantiations cannot balance the increasing workload or may be redundant, and provide an approach to manually adjust the instantiations, such as adding the instantiation number for balancing or releasing the workload, reducing the number for saving costs.

6.3.4 WfSMI service

WfSMI service manages WfSMIs and supports scaling-out actions in the auto-scaling service and records historical details of instantiations. The service administrates various operations such as query, create, update, and discard a WfSMI, etc.

For querying, the service retrieves one or more WfSMIs according to a search or statistic condition from the client side. The search results can be returned to the client side.

For creating, the service is able to collect available software or hardware configurations from cloud infrastructures within constraints of the system-level annual budget plans and support creation of new virtual machine. It can further support the instalment, deployment and testing of application server and workflow enactment service through invoking APIs of cloud infrastructures.

For updating, the service is able to support modification of configurations of an existing WfSMI and regenerate a new image with a new version number through invoking the APIs of cloud infrastructures. The regenerated image may be assigned a new image identity or keep the original image identity on demand and be released to the cloud side for updating an existing image. The service provides two approaches to update: adding the regenerated image

as new image; replacing or overwriting the existing image using the regenerated image when the existing image is not in use.

For discarding, the service is able to remove the existing WfSMIs from the cloud side. An existing WfSMI cannot be removed directly because it may be used anytime. For removing, an image can be specified using a deprecated flag. The auto-scaling service will check the flag and choose the image which has no flag to scale out. The image can be removed when it is not in use permanently.

6.3.5 Alarm administration and monitoring

Alarm administration and monitoring manage changes of various utilisation metrics defined in Subsection 6.2.6 and updates specifications of the metrics and tunes thresholds in the alarm estimation model on demand. The estimation model in the alarm service will be addressed in Subsection 6.3.6 in detail. The functionalities include:

- Graphically representing the utilisation metrics. The representation enables administrators to manually select the resources which they concern about for monitoring, for example, the resources which are running out or idle or under loaded or overloaded. The monitoring can further provide indications on the basis of the comparisons with the preconfigured thresholds.

- Modifying the specifications of the utilisation metrics, i.e., adding or reducing one or more metrics in the alarm estimation model. The functionality enables the model to concern about the utilisation metrics which change significantly in a time period. If the model estimates all utilisation metrics, the estimation may influence the performance of workflow enactment service components because of frequently obtaining data of the critical resources.

- Tuning the thresholds of the utilisation metrics of the resources. The thresholds influence estimations to the resources and eventually influence scaling actions in the scaling service.

6.3.6 Alarm service

The alarm service collects status data of all workflow enactment services in SwinFlow-Cloud and periodically estimates the status for the load balancing service for checking the availability and the auto-scaling service for giving a scaling notification. The checking needs the alarm service to provide a comprehensively quantified estimation result of each workflow enactment service component, which can indicate whether a request is acceptable or not. The result integrates the utilisation metrics defined in Subsection 6.2.6 into a unified metric for checking. On the other hand, each system resource may be a potential performance bottleneck due to full loading or overloading during workflow enactment. The bottlenecks may result in overall unavailability of a workflow enactment service. Therefore, the auto-scaling service needs the alarm service to provide details of the performance bottlenecks in alarm notifications for taking the resources-targeted scaling actions to eliminate the bottlenecks economically.

To achieve the goals, we propose an alarm estimation model to integrate the utilisation metrics and represent the overall availability. The alarm service periodically uses the model to estimate system status. If the time period is $T = \{0,1,2, \ldots\}$, then for any moment t, $t \in T$.

If there are m workflow enactment services with n resources, then for service j, $j = 1..m$, at any moment t, an utilisation metric $r_{ij}(t) \in [0,1]$, $i = 1..n$ is a ratio of the resource utilisation $u_{ij}(t)$ and the capacity c_{ij}. Here, we define a weight $w_{ij}(t)$ as a coefficient of $r_{ij}(t)$ to represent the impact on $r_{ij}(t)$. $w_{ij}(t)$ can further assist to identify a resource bottleneck. In theory, when a workflow enactment service is idle, the utilisation of each resource is *0* and has adequate capacities to execute workflows. It means that there are no bottlenecks in the workflow enactment service and each resource has same weight $w_{ij}(t)$, denoted as w_0. For n resources, $w_0$is:

$$w_0 = \frac{1}{n} \tag{1}$$

From Eq. (1), $\sum_{i=1}^{n} w_{ij}(t) = 1$. $w_{ij}(t)$ is able to change with a $\Delta w_{ij}(t)$ when $r_{ij}(t)$ is changing. We define $w_{ij}(t)$ as follows,

$$w_{ij}(t) = w_0 + \Delta w_{ij}(t) \tag{2}$$

$$\Delta w_{ij}(t) = r_{ij}(t) - \frac{\sum_{i=1}^{n} r_{ij}(t)}{n} \tag{3}$$

From Eq. (3), $\sum_{i=1}^{n} \Delta w_{ij}(t) = 0$, $\Delta w_{ij}(t) \in [0, 1]$. Thus, it ensures $\sum_{i=1}^{n} w_{ij}(t) = 1$. In Eq. (3), when the other resource utilisation is kept smooth and steady, if $r_{ij}(t)$ changes more significantly, then $\Delta w_{ij}(t)$ is also more distinct. Using $w_{ij}(t)$, we further define a metric, called composite index *I*, to assess the overall status of a workflow enactment service. Each workflow enactment service has *I* at moment *t*, thus, for service j, $I_j(t)$ is:

$$I_j(t) = \sum_{i=1}^{n} w_{ij}(t) \cdot r_{ij}(t) \tag{4}$$

In Eq. (4), $r_{ij}(t) \in [0, 1]$, $w_{ij}(t) \in [-1, 1]$, then $I_j(t) \in [0, 1]$. Theoretically the minimum value *0* of $I_j(t)$ means that the system is idle due to no resource being used and $r_{ij}(t)$ is *0*, while the maximum value *1* of $I_j(t)$ means that all resources are in full workload and $r_{ij}(t)$ is *1*. Generally speaking, $I_j(t)$ vary between the two extreme values.

Therefore, the index model indicates the overall status of a workflow enactment service from the idle to the fully loaded. Obviously, there are other states within this range including the under loaded, the normal load, busy, extra busy, and overloaded etc. Generally speaking, the discrete states can be mapped onto the multiple continuous ranges within $[0, 1]$ through specifying the thresholds. That is, we use a threshold set $TH = \{th_1, th_2, \dots th_k\}$, to divide the open interval of $[0, 1]$ into $k + 1$ parts. When *I* changes from $[th_{k-2},\ th_{k-1}]$ to $[th_{k-1},\ th_k]$, the workflow enactment service will change from one state to another, e.g., from the under loaded to the normal load.

Based on the composite index model, we can estimate the system status of SwinFlow-Cloud through analysing the composite index obtained from each workflow enactment services. For *m* workflow enactment services, the mean of $\bar{I}(t)$ and the mean of $\Delta\overline{w}_i(t)$ are:

$$\bar{I}(t) = \frac{\sum_{j=1}^{m} I_j(t)}{m} \tag{5}$$

$$\Delta\overline{w}_i(t) = \frac{\sum_{j=1}^{m} \Delta w_{ij}(t)}{m} \tag{6}$$

$\bar{I}(t)$ and $\Delta\overline{w}_i(t)$ are stochastically variables in $[0, 1]$ and depend on T. Therefore, we can consider $\bar{I}(t)$ and $\Delta\overline{w}_i(t)$ as two stochastic processes depending on T, denoted as $\{\bar{I}(t), t \in T\}$, $\{\Delta\overline{w}_i(t), t \in T\}$. Then, $[0, 1]$ is their state space.

The estimation is to demonstrate that $\bar{I}(t)$ is changing from one state to another due to some resource utilisation spiking significantly. The spike of the resource utilisation is generally rapid when SwinFlow-Cloud tackling large-scale applications, especially the instance-intensive workflows. Therefore, the time period T of the estimation must be short to enable $\bar{I}(t)$ to reflect the current utilisation and provide an estimable basis for near future utilisation.

Thus, on the basis of the specifications above, obtaining $\bar{I}(t)$, we utilise the quadratic curve fitting regression to calculate the tendency of the state transformation, i.e., from one state to another, as shown in Fig. 6.3, and further utilise discrete-time Markov chains to predict the probability of future $\bar{I}(t)$ which stays on the new state.

For the quadratic curve fitting, we consider $\bar{I}(t)$ as an input data sequence, denoted as $(t, \bar{I}(t)), t \in T, T = \{0,1, \dots, k-1\}$. We assume that the quadratic equation is

$$\bar{I}(t) = a_0 + a_1 t + a_2 t^2 \tag{7}$$

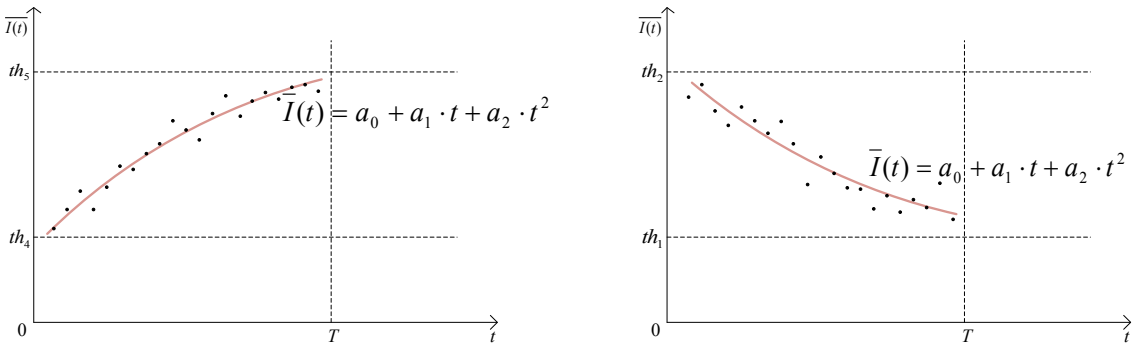

Figure 6.3 Quadratic curve fitting on $\bar{I}(t)$

Then we solve the following equation group to calculate a_0, a_1, a_2:

$$\begin{pmatrix} k & \sum_{i=0}^{k-1} t & \sum_{i=0}^{k-1} t^2 \\ \sum_{i=0}^{k-1} t & \sum_{i=0}^{k-1} t^2 & \sum_{i=0}^{k-1} t^3 \\ \sum_{i=0}^{k-1} t^2 & \sum_{i=0}^{k-1} t^3 & \sum_{i=0}^{k-1} t^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{k-1} \bar{I}(t) \\ \sum_{i=0}^{k-1} t \cdot \bar{I}(t) \\ \sum_{i=0}^{k-1} t^2 \cdot \bar{I}(t) \end{pmatrix} \tag{8}$$

After solving a_0, a_1, a_2, we can utilise first derivative to derivate the quadratic equation and get $\bar{I}'(t) = a_1 + 2a_2 \cdot t$. If $\bar{I}'(t) > 0$ or $\bar{I}'(t) < 0$, it means that $\bar{I}(t)$ is monotonically increasing or decreasing. $\bar{I}(t)$ is possible to change its status in the near future. If $\bar{I}'(t) = 0$, it means that $\bar{I}(t)$ keeps steady on the current status.

As a stochastic process, $\bar{I}(t)$ has memory-less property, that is, if we know current $\bar{I}(t_i)$, then future $\bar{I}(t_{i+1})$ has no relationship with past $\bar{I}(t_{i-1}), t_{i-1}, t_i, t_{i+1} \in T$. Therefore, when obtaining $\bar{I}'(t) \neq 0$, we utilise a discrete-time Markov chain to further calculate the probability of the state transformation. But the discrete-time Markov chain needs a state set with a finite number to build a transition matrix $\boldsymbol{P}_t$. As discussed above, we can discretise the continuous state space of $\bar{I}(t)$, $[0, 1]$ into finite open intervals using a threshold set TH and map the intervals onto finite states. We define a state set *S* for the discrete-time Markov chain, including:

- The idle state (*IDLE*): This state represents that all resources of a workflow enactment service are close to idle and do not bear any workload.

- The under loaded state (*UNDL*): It represents that the resources of the workflow enactment service are under loaded. The auto-scaling service should take scaling-down actions to stop or terminate the workflow enactment service for saving costs.

- The normal load state (*NLLD*): It represents that the resources of the workflow enactment service are working fine. The workloads are appropriate for the resources. It is a normal and practical state for cloud workflow running.

- The busy state (*BUSY*): It represents that the resources of the workflow enactment service are busy in executing workflows. The workload may be heavy for the resources.

- The extra busy state (*EXBY*): It represents that the resources of the workflow enactment service are extra busy and some resources are likely overloaded or running out at any time. It is a warning state for the resources. The workflow enactment service cannot tackle new requests and must not be distributed new requests. The auto-scaling service should take scaling-out actions to add new workflow enactment services for balancing the workloads.

- The overloaded state (*OVRL*): It represents that the resources of the workflow enactment service ran out. The service may be unrecoverable due to the overloaded workload. This is a fatal state and should not occur.

That is, there are six states in the set $S = \{IDLE, UNDL, NLLD, BUSY, EXBY, OVRL\}$ to identify the status of $\bar{I}(t)$. The six states are mapped onto the open interval $[0, 1]$ using the

threshold set $TH = \{th_1, th_2, .., th_5\}$, i.e., the *IDLE* state refers to $[0,\ th_1)$; the *UNDL* state refers to $[th_1,\ th_2)$; the *NLLD* state refers to $[th_2,\ th_3)$; the *BUSY* state refers to $[th_3,\ th_4)$; the *EXBY* state refers to $[th_4, th_5)$; the *OVRL* state refers to $[th_5,\ 1]$.

In this thesis, r_{ij} has the same state set *S* like $\bar{I}(t)$. Thus, the threshold set of r_{ij} which *S* is mapped on, denoted as TH_{ij}, has the same quantity of thresholds. But for the same *i* resources in *m* services, if *m* services have various configurations, the threshold set may be different. For example, for the memory resource, if two services have 100 MB and 200 MB memory respectively, $TH = \{th_1, th_2, .., th_5\}$ of the former may be $\{0.14, 0.22, 0.55, 0.8, 0.9\}$, while TH of the latter may be $\{0.153, 0.25, 0.574, 0.79, 0.89\}$. Then, TH of the $\bar{I}(t)$ can be calculated through TH_{ij} of r_{ij}.

Moreover, we define a random variant $X(\bar{I}(t_k))$, $t_k \in T$. Then, the probability of $X(\bar{I}(t_k)) \in S$ is $P(X(\bar{I}(t_k)) \in S)$. It means that the probability *P* of *X* getting the value of *S* is determined by $\bar{I}(t_k)$ and $\bar{I}(t_k)$ is determined by t_k. From t_k to t_{k+1}, the conditional transition probability is

$$P\{X_{t_{k+1}} = s_i | X_{t_k} = s_j\} = \frac{P(X_{t_k} X_{t_{k+1}})}{P(X_{t_k})}, t_k, t_{k+1} \in T; s_i, s_j \in S \tag{9}$$

Thus, $\boldsymbol{P}_t$ is a 6x6 transition matrix where the element p_{ij}at row *i* and column *j* describes the conditional probability of a transition from state *i* to state *j*. Thus,

$$p_{ij} = P\{X = s_i | X = s_j\} \tag{10}$$

Generally speaking, the conditional transition probability can be approximately calculated by the frequency of $\bar{I}(t_k)$being in the interval $[th_l,\ th_{l+1}]$ which s_i expresses $th_l, th_{l+1} \in TH; t_k \in T; s_i \in S$. Using $\boldsymbol{P}_t$, we can calculate the one step transition, i.e., the probability of $\bar{I}(t_{k+1})$ being on s_i at next moment t_{k+1}.

If s_i and s_j are states of *S*, $\bar{I}(t_k), t_k \in T$, is able to transit from s_i to s_j through *n* steps because the range of value of $\bar{I}(t_k)$ is [0, 1]. Thus, the discrete-time Markov chain is homogeneous and can calculate the next *n* steps transition probability of $\bar{I}(t_k)$ by applying the Chapman-Kolmogorov equation, i.e., $\boldsymbol{P}_t(n) = {\boldsymbol{P}_t}^n$.

As discussed above, we can use the discrete-time Markov chain to predict the probability of $\bar{I}(t_k)$ being on specific states when $\overline{I'}(t_k) \neq 0$. If the prediction indicates that the tendency of $\bar{I}(t_k)$ is on the *UNDL* or *EXBY* state approaching the *IDLE* or *OVRL* state, it means that cloud workflow needs to scale out or in. The alarm service will give an alarm notification to the auto-scaling service for taking scaling actions.

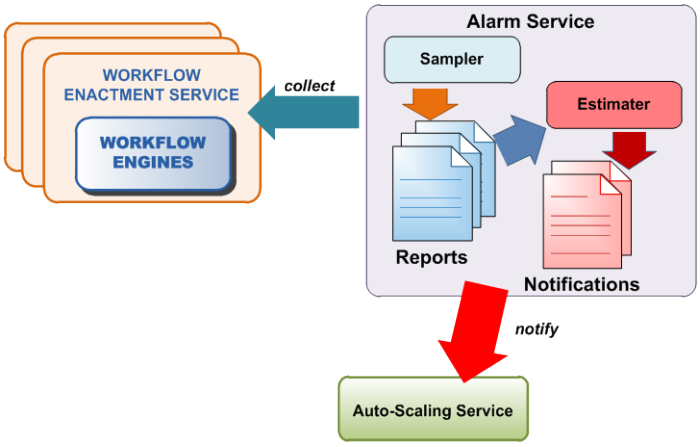

Figure 6.4 Structure of alarm service

Furthermore, we design a module structure, as shown in Fig. 6.4. The sampler periodically communicates with each workflow enactment service component, obtains a snapshot of the specified utilisation metrics and generates a report with a timestamp *t* for estimation.

The reports comprehensively describe the specified resource utilisation and the possible performance bottlenecks in the workflow enactment service components. The report consists of identity, utilisation metric item, and timestamp, etc. A utilisation metric item describes the identity of a workflow enactment service, $I(t)$, $r_{ij}(t)$, u_{ij}, c_{ij}, $\Delta w_{ij}(t)$ and $w_{ij}(t)$, $i = 1..n.$, and the moment *t*, etc.

The estimator calculates $\bar{I}(t)$ in every report, collects $\bar{I}(t)$ for a time period *T* and solves a_0, a_1, and a_2 in Eq. (7). If $\bar{I}(t)$ is on the *UNDL* or *EXBY* state and $\overline{I'}(t) < 0$ or $\overline{I'}(t) > 0$ in several continuous time periods and $\overline{I'}(t)$ is decreasing or increasing in these periods, the estimator will further predict the probability of $\bar{I}(t)$ being approaching the *IDLE* or *OVRL* state. If the prediction indicates the probability is much higher than $\bar{I}(t)$ being on other states, the alarm service will send a notification to the auto-scaling service.

The notification quantitatively describes the system status, including the details of the resource utilisation of all workflow enactment services and estimation results from the alarm service in several time periods and a timestamp. The notification is a basic evidence for the auto-scaling service to take actions.

6.3.7 Load balancing service

As presented in the client-cloud architecture, the service component balances evenly the workflow requests onto all workflow enactment service components in SwinFlow-Cloud to prevent the requests from aggregating on one component. The structure of the service component is illustrated in Fig. 6.5.

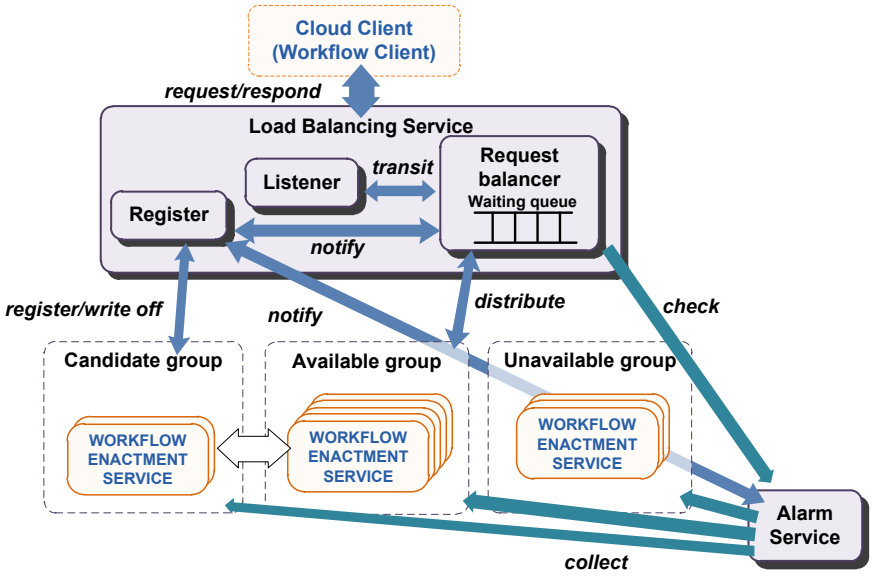

Figure 6.5 Structure of load balancing service

The component separates the workflow enactment service components into three resizable groups. The first group is composed of the components which are able to accept new workflow requests due to under loaded, denoted as available group; the second group is composed of the components which are not able to accept new workflow requests due to fully or over loaded, denoted as unavailable group. The unavailability of components means one or more critical resources may be fully or over loaded and they need time to handle the accepted requests and they can accept new requests after responding the existing requests; The third group is composed of some idle workflow enactment service components, denoted as candidate group. The components in this group can be added into the available group any time. There are three modules in the service:

- Web listener: It listens to the SOAP based workflow requests on the 80 port. The requests can be transferred to the request balancer for transiting.

- Request balancer: It can distribute the requests onto the workflow enactment service components in the available group. The workflow requests are usually attached by the session information. The session contains the information about workflow service, workflow instance, operation content, workflow user (operator), etc. Once receiving a new request, the balancer will look up appropriate service component in two groups according to the session and check whether the component is available and healthy, i.e., request-acceptable, through communicating with the alarm service. If the balancer found the service component but checked that it is not in the available group, the balancer puts the request into a queue to wait for the service component. The balancer checks the availability of all service components in two groups at every fixed time interval. It looks up the unavailable service components in the available group and moves the unavailable into the unavailable group; it also looks up the available service components in the unavailable group and moves the available into the available group. Once the service components are available, the balancer will distribute the waiting request to them.

- Register: It is able to register or deregister a workflow enactment service component and move it into or out from the candidate group, then send a notification about the detail of the service component to the request balancer and the alarm service. Once receiving a confirmation from them, it adds the service component into the available group or moves the service component out from the candidate group.

6.3.8 Auto-scaling service

Auto-scaling service estimates resource demands through interpreting notifications from the alarm service and takes the scaling-out and scaling-in actions.

The notification cannot explicitly indicate the concrete resource demands which the auto-scaling service needs to create or add to balance the workloads on current status because there are no inherent relationships between the resource utilisation in workflow enactment services and the future resource demands. For example, if the utilisation of 100 MB memory in a workflow enactment service component is mostly 90% at a moment, for the vertical

scaling, it does not mean that the utilisation of the memory will decrease to 45% in the future after we add 100 MB memory to the service component; for the horizontal scaling, if adding a new workflow enactment service component with 100 MB memory, it does not mean that utilisation of the memory of the previous service component will decrease 45% in the future. If the pervious service does not complete the current workflow instances, it may keep 90% utilisation of the memory for a long time.

Moreover, for a workflow instance, its resource utilisation is unknown and cannot be predicted because the workflow instance may have multiple patterns of execution paths, including sequential, parallel, choice, or iterative, etc. The resource demands of each execution path cannot be predicted individually if there are no historical executions as a reference. Even if there are the historical executions, for a resource, e.g., CPU, it may be much different because the capacity varies. For example, the utilisation of CPU in 32-bit system is different from that in 64-bit system when executing a workflow instance.

Therefore, the appropriate approaches to solve the unpredictable resource demands are to analyse the utilisation of the resources inside the workflow enactment service components for scaling out or in.

Based on the discussion above, we propose a scaling estimation model to analyse the utilisation of the resource metrics which the notifications contain and calculate the number or capacity of the resource demands in the future.

As addressed in the final paragraph in Subsection 6.3.6, the notification consists of the details of resource utilisation of all workflow enactment service components and estimation results in *T*. It is constituted by multiple resource utilisation sections. A resource utilisation section represents the resource utilisation and results which the alarm estimation model calculates in *T* and consists of multiple resource utilisation segments. Each resource utilisation segment consists of an overall resource utilisation item and multiple individual resource utilisation items of the service components. The overall resource utilisation item, calculated by the alarm estimation model, includes the mean of $\bar{I}(t)$, $\overline{r}_i(t)$, $\Delta\overline{w}_i(t)$ and $\overline{w}_i(t)$, the sum $\sum u_i$ of i resource and the sum $\sum c_i$ of i resource, $i = 1..n$, of n resources of all service components, at moment t in T; while the individual resource utilisation item of each service component includes $I_j(t)$, $r_{ij}(t)$, $\Delta w_{ij}(t)$, $w_{ij}(t)$, u_{ij}, c_{ij}, and TH_{ij}, $t \in T$.

For scaling analysis, if there are m service components with n resources that are providing workflow services, we can obtain m individual resource utilisation items at t, $t \in T$.

In the items, we find $r_{ij}(t)$ with $\Delta w_{ij}(t) > 0$ and TH_{ij} to calculate resource demands. We denote $r_{ij}(t)$ with $\Delta w_{ij}(t) > 0$ as $r'_{ij}(t)$. As defined in Subsection 6.3.6, the open interval $[th_4,\ th_5) \subset [0, 1]$ is corresponding to the *EXBY* state.

Here, we utilise the quadratic curve fitting methodology to fit $r'_{ij}(t)$, denoted as input data sequence, $(t,\ r'_{ij}(t))$, $t \in T, T = \{0,1, \dots, k-1\}$. Assuming the quadratic equation is

$$r'_{ij}(t) = b_0 + b_1 t + b_2 t^2 \tag{11}$$

The curve of $r'_{ij}(t)$ is similar to that of $\overline{I(t)}$ in Eq. (7), as shown in Fig. 6.6.

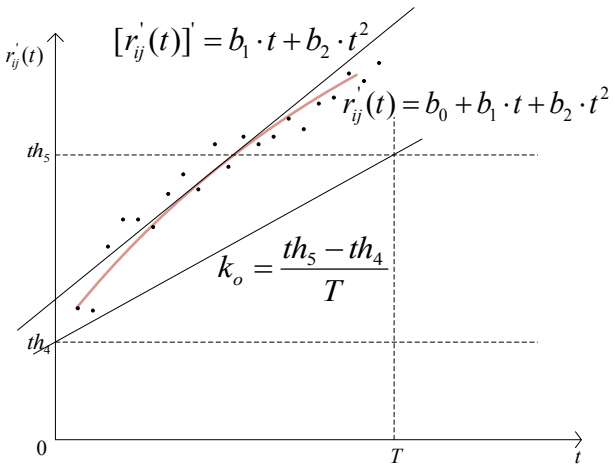

Figure 6.6 Quadratic curve fitting to estimate scaling-out

Then we solve the following equation group to calculate b_0, b_1, b_2:

$$\begin{pmatrix} k & \sum_{i=0}^{k-1} t & \sum_{i=0}^{k-1} t^2 \\ \sum_{i=0}^{k-1} t & \sum_{i=0}^{k-1} t^2 & \sum_{i=0}^{k-1} t^3 \\ \sum_{i=0}^{k-1} t^2 & \sum_{i=0}^{k-1} t^3 & \sum_{i=0}^{k-1} t^4 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{k-1} r'_{ij}(t) \\ \sum_{i=0}^{k-1} t \cdot r'_{ij}(t) \\ \sum_{i=0}^{k-1} t^2 \cdot r'_{ij}(t) \end{pmatrix} \tag{12}$$

After solving b_0, b_1, b_2, we utilise first derivative to derivate the quadratic equation:

$$[r'_{ij}(t)]' = b_1 + 2b_2 t \tag{13}$$

In Eq. (13), if $[r'_{ij}(t)]' > 0$, it means that $r'_{ij}(t)$ is monotonically increasing. $r'_{ij}(t)$ is possible to change status in the near future. We further calculate an slope, denoted as k_o:

$$k_o = \frac{th_5 - th_4}{T} > 0 \tag{14}$$

Here, we define k_o as maximum speed at which $r'_{ij}(t)$ increases. If $[r'_{ij}]' > k_o$, then we can conclude that the increase of $r'_{ij}(t)$ in *T* is higher than the maximum speed. Thus, we can collect $r'_{ij}(t)$ in the workflow enactment service components, of which increase is higher than the maximum speed, and take scaling actions. For example, if $[r'_{ij}]', i = mem$, of the utilisation of memory in *T* is higher than k_o, then, for scaling-out actions, we can configure a WfSMI with much more memory capacity to create new service components for balancing the workloads.

Based on the analysis above, we can be aware of the significantly under loaded or overloaded resources in the service components. It needs to provision much more number or capacities of the resources when taking scaling actions.

Next we further estimate and calculate the number or capacities of resource demands at one scaling-out or scaling-in action. As analysed above, it cannot determine the accurate number of the resources or service components which are scaled out or in by recovering the resource utilisation ratios to the *NLLD* state. Theoretically, we can scale out by infinite service components for decreasing $\bar{I}(t)$ to the *NLLD* state without cost considerations. But it is not a cost-effective scaling. In this thesis, we try to determine an appropriate number range of the services with the more powerful capacity of the resources to guarantee a cost-effective scaling.

For *m* service components, if $r'_{ij}(t)$ of k' resources in l' services is overloaded in *T*, denoted as $\{r'_{ij}(t), i = 1..k', j = 1..l', t \in T\}$, and the capacity of these resources is $\{c_{ij}, i = 1..k', j = 1..l'\}$.

A scaling-out action is to add one or more service components to decrease $r'_{ij}(t)$ to avoid the resources running out because $r'_{ij}(t)$ enables $\bar{I}(t)$ to reach the *EXBY* state. Through parsing the notifications from the alarm service, we can obtain $r'_{ij}(t)$ and c_{ij}. Thus, we can calculate the total utilisation su_i of the *i* resource in l' service components is:

$$su_i = \sum_{j=1}^{l'} r'_{ij}(t) \cdot c_{ij} \tag{15}$$

To decrease $\bar{I}(t)$ to the *NLLD* state, we need decrease and keep su_i in $[th_2, th_3)$ .Then, the total capacity cp_i for this utilisation of the *i* resource is:

$$cp_i = \frac{su_i}{\frac{th_2+th_3}{2}} \tag{16}$$

Here, we can obtain the maximum available resource capacity $c_{\max i}$ in billing budget plans offered the billing service and calculate the minimum quantity of the *i* resource which should be scaled out. When obtaining the set of the minimum quantities of the k' resource, we can get the maximum quantity $sc_{\min}$ in the set to notify the WfSMI service to create new workflow enactment service components. Therefore, to recover $\bar{I}(t)$ to the *NLLD* state, we need at least $sc_{\min}$ service components to balance the remainder of workloads.

Furthermore, if adding $su_{\min}$ services, the total number of services is $m + su_{\min}$ and $r_{ij}(t), i = 1..n$, in new services is *0*, thus

$$I_j(t) = \sum_{i=1}^{n} r_{ij}(t) \cdot \Delta w_{ij}(t), j = 1..m + sc_{\min}, t \in T \tag{17}$$

$$\overline{I}_{new}(t) = \frac{\sum_{j=1}^{m+sc_{\min}} I_j(t)}{m+sc_{\min}} < \frac{\sum_{j=1}^{m} I_j(t)}{m} = \overline{I(t)} \tag{18}$$

If $sc_{\min}$ is equal to m, i.e., $\overline{I}_{new}(t) = \frac{\overline{I(t)}}{2}$, but $\bar{I}(\mathrm{t}) \in [th_4,\ th_5]$, then $\frac{\bar{I}(t)}{2} \in \left[\frac{th_4}{2},\ \frac{th_5}{2}\right)$. As defined in Subsection 6.3.6, $th_5 \leq 1$, thus, $\frac{th_5}{2} \leq 0.5$, thereby, $\overline{I}_{new}(t) \leq 0.5$. Therefore, the composite index $\overline{I}_{new}(t)$ can decrease to under 0.5 after adding *m* services.

According to the analysis above, a scaling-up action should scale out by at least $sc_{\min}$ services but not more than *m* services to recover the overall composite index $\overline{I(t)}$ for balancing the remainder of the workload. We denote the scaled-out number as SCL_u, then $SCL_u \in [sc_{\min}, m]$.

A scaling-in action is to stop or terminate l', $l < m$, service components with $I_j(t)$, $j = 1..l'$ at the *UNDL* states to facilitate $\bar{I}(t)$ to stay at or recover to the *NLLD* state for a cost-effective execution. The services with $I_j(t)$ at the *UNDL* states results in $\bar{I}(t)$ decreasing to the *UNDL* state, i.e., $\bar{I}(t) \in [th_1, th_2)$. Obviously, $I_j(t)$ have two distributions in $[0, 1]$: the one is l' services with $I_j(t) \leq \bar{I}(t)$, while the other is $m - l'$ services with $I_j(t) \geq \bar{I}(t)$. The former means that the utilisation of the resources is lower than those of the latter. Therefore, for

scaling-down, we should stop or terminate l' service components. However, l' may vary due to $I_j(t), j = 1..m$. If $I_j(t) \leq th_2, j = 1..m$, then $\bar{I}(t) \leq th_2$, thereby $l' = m$. Thus a scaling-down may stop or terminate all service components. But it is only a theoretical possibility. In practice, there is a minimum number *Z* of service components in system to support the lowest limit of workflow services. That is, the scaling-down can decrease *m* to *Z* at most.

Therefore, according to the above analysis, a scaling-in action should scale in by at least l' services but not more than *m-Z* services to recover $\bar{I}(t)$ to the *NLLD* state. We denote the scaled-in number as SCL_d, then $SCL_d \in [l', m - Z]$.

The auto-scaling service supports the scaling estimation model and generates the scaling-out or scaling-in strategies for scaling actions and creates scaling engines to execute the actions.

The estimator listens and interprets the notifications from the alarm service and calculates the scaling-out or scaling-in number using the scaling estimation model. In SwinFlow-Cloud, the horizontal scaling and vertical scaling are integrated into one strategy for support the scaling.

The scaling-out number range is $[sc_{\min}..m]$, and the scaling-out capacities include the calculated CP_i of r_i', $i = 1..n$. The estimator can dynamically create multiple scaling-out strategies using the data according to the annual budget plans. For scaling number, if *m* is multiple times as many as $sc_{\min}$, the estimator will generate three strategies with three numbers: $sc_{\min}$, $\frac{m+sc_{\min}}{2}$, and *m* to scale out; if *m* is one time as many as $sc_{\min}$, the estimator will generate three strategies with two numbers: $sc_{\min}$ and *m* to scale out. The scaling-out resource capacities, as resource demands, can be sent to the image service to automatically generate the images which meet the demands. But in this thesis, the automatic generation of an image is not a research focus. We can predesign or predefine multiple images for supporting the scaling-out capacities and look up for the most approximate images to meet the resource-demands.

The scaling-in number range is $[l'..m - Z]$. The estimator can dynamically create multiple scaling-in strategies using the data. For scaling number, if $m - Z$ is multiple times as many as l', the estimator will generate three strategies with three numbers: l', $\frac{m-Z+l'}{2}$, and

$m - Z$ to scale down; if m is one time as many as l', the estimator will generate three strategies with two numbers: l' and m to scale in.

After generating these scaling strategies, the auto-scaling service creates one or more scaling actions to implement these strategies. Every action references a scaling strategy. The service further creates a scaling engine for each scaling action. Once an engine is created, it will be executed to recover $\bar{I}(t)$ to the *NLLD* state.

A scaling engine is an execution framework of a scale action and can integrate APIs of cloud infrastructures to create or remove one or more service components using an image. The execution of an engine is separated by several phases:

- Launching: The scaling engine can collect all relevant data according to the scaling action, including referenced image, number of the services which needs to scale out or in, waiting interval to query the completion status of new service components.

- Running: The scaling engine can invoke APIs of cloud infrastructures to create or remove service components according to the specification in the scaling strategy. It needs to spend some time in creating or removing a service component. The engine is able to wait for the completion of creation and removal through a query for every waiting interval specified. Once the creation or removal is completed, the engine will send a notification to the load balancing service or the alarm service for registering or deregistering.

- Completed: The scaling engine can complete and notify the auto-scaling service for launching next scaling engine. The execution of a scaling engine is exclusive for any other scaling engines and estimations in the auto-scaling or alarm service. That is, when an engine is running, the estimations of the services will be suspended. Because the number of the service components is changing during the scaling engine execution, the number may influence the estimations of the services. Therefore, the execution period of a scaling engine should be as short as possible to reduce the influence.

The auto-scaling service will send a notification to the alarm service to estimate at once for ensuring $\bar{I}(t)$ to recover the *NLLD* state. If not, the alarm service will send the notifications again. The auto-scaling service can create a new scaling engine to take actions with bigger number of scaling out or in.

6.3.9 Principles for the coordination sustainability

As addressed in subsections above, the runtime functions which every service, i.e., billing, WfSMI, load balancing, alarm, and auto-scaling, undertakes are closely associated with other services and implemented by efficient coordination between them. The coordination guarantees that SwinFlow-Cloud provisions sustainable workflow services which meet the requirements analysed in Section 3.2. As discussed in Subsection 4.3.6, the sustainability is impacted by the elements. Aiming at the design of the runtime functions of SwinFlow-Cloud in this chapter, we further discuss these elements and reveal the constraints between them. These constraints can directly or indirectly influence the sustainability of the coordination and are the references to facilitate SwinFlow-Cloud to support a workflow application.

The elements include the pending time, denoted as t_{pd}, the decaying time, denoted as t_{dc}, the time period for the alarm service, denoted as T_{al}, the time period for the load balancing service, denoted as T_{ld}, the thresholds of $\bar{I}(t)$, denoted as $TH = \{th_1, th_2, \dots\}$, the quantity of the available resources, denoted as Z, and the quantity of the resources to scale out or in, i.e., SCL_u or SCL_d.

t_{pd} reflects the efficiency of the scaling-up action, i.e., the shorter t_{pd} is, the higher the efficiency. As discussed in the last subsection, when a scaling-out action is running, the estimations of the auto-scaling service and the alarm service will be suspended. It means that the composite index $\bar{I}(t)$ may be approaching to the *OVRL* state but the alarm service cannot be aware if the efficiency of the scaling-out action is lower. It will be a fatal risk of impacting the sustainability. Hence, decreasing $t_{pd}$is a goal of configuring an efficient scaling-out action.

t_{dc} reflects on a maximum speed of SwinFlow-Cloud from the recoverable status to the unrecoverable status, i.e., the *OVRL* state. The higher the speed is, the shorter t_{dc} is. If t_{dc} is longer, it means that $\bar{I}(t)$ needs longer time to transit on the *OVRL* state. In theory, if t_{dc} is infinitely long, $\bar{I}(t)$ will never be on the *OVRL* state. For instance-intensive workflow, the scaling-out actions are to decrease the speed of the workflow applications exhausting the resources to prolong t_{dc}. That is to say, the speed of the scaling-out action creating new workflow enactment service components must be faster than that of the workflow applications exhausting the resources, i.e., it must be $t_{pd} < t_{dc}$. $\bar{I}(t)$ is able to keep recoverable and does not transit to the *OVRL* state.

Thus, if $t_{pd} < t_{dc}$, then cloud workflow is able to keep the system status recoverable and prevent from transiting to the *OVRL* state.

T_{al} reflects the sensitivity of the alarm service to SwinFlow-Cloud. As a core estimator to support the load balancing service and the scaling service, the alarm service must be sensitive to the system status. In theory, if T_{al} is zero, the alarm service will be aware of the system status in real time. But it is not possible in practice. If T_{al} is longer, it may result in system status transiting to the *OVRL* state but the alarm service will not be aware of it.

Thus, if $T_{al} < t_{dc}$, then the alarm service can monitor timely the state transitions of $\bar{I}(t)$.

Furthermore, if the alarm service is aware of $\bar{I}(t)$ transiting to the *OVRL* state, it notifies the scaling service to scale out. The scaling service takes a scaling-out action to prevent $\bar{I}(t)$ from transiting to the *OVRL* state. The action spends t_{pd} in completing the scaling. If the total time of the alarm service estimation and the scaling-out action is longer than t_{dc} i.e., $T_{al} + t_{pd} > t_{dc}$, then undoubtedly $\bar{I}(t)$ will transit to the *OVRL* state. Thus, we propose a principle for this case.

Principle 1: if $T_{al} + t_{pd} < t_{dc}$, then $\bar{I}(t)$ is able to keep recoverable.

T_{ld} reflects the sensitivity of the load balancing service to the workflow enactment services for determining whether to distribute a request. The load balancing service is aware of the status of the $\bar{I}(t)$ through the estimation of the alarm service. Thus, if the alarm service does not estimate the status of $\bar{I}(t)$ timely, the load balancing service will not be able to check the status of $\bar{I}(t)$ for distribution. On the other hand, if T_{ld} is longer than T_{al}, it may result in $\bar{I}(t)$ transiting to the *OVRL* state but the load balancing service cannot be aware of it. Thus, we propose another principle for this case.

Principle 2: when the load balancing service needs to check $\bar{I}(t)$ before dispatching a request, if $T_{ld} < T_{al}$, then the load balancing service is able to monitor timely the state transitions of $\bar{I}(t)$.

In this subsection, we discussed the constraints between the elements that affect the coordination sustainability and theoretically propose the two principles. There are still more constraints between the elements which need to be revealed. We will further research the other constraints in future.

6.4 Summary

In this chapter, we firstly gave an overview of the non-functional design in SwinFlow-Cloud and then addressed the build time and runtime function design. The build time aspects cover management for such as versioning, organisation, tool agent invocation, billing, WfSMI, alarm, and auto-scaling; while the runtime aspects cover the administration and monitoring and services. The administration and monitoring include billing, WfSMI, alarm, etc. The services cover billing, WfSMI, alarm, load balancing, and auto-scaling service. Finally, we discussed the principles for the coordination sustainability.

Chapter 7
SwinFlow-Cloud Prototype Implementation

This chapter presents the prototype implementation of SwinFlow-Cloud based on the client-cloud architecture proposed in Chapter 4. Firstly, Section 7.1 gives an overview of the prototype implementation and the related technology. Then Section 7.2 addresses the implementation of the build time functions. Finally, Section 7.3 addresses the implementation of the runtime functions.

7.1 Overview of the implementation

This section introduces development technology, system development, and system runtime environment of SwinFlow-Cloud.

7.1.1 Development technology

SwinFlow-Cloud prototype is developed in Java programming language on J2SE 1.6 platform based on the design of the build time and runtime functions described in Chapters 5 and 6. This section addresses the key parts of the critical Java based technologies for building this prototype.

The client side is composed of the desktop-based applications which are developed based on the technology of Eclipse Rich Client Platform (RCP). The Eclipse RCP is a framework for creating modular user interfaces based on Open Services Gateway Initiative

(OSGi). As a model to modularised Java applications, OSGi technology is a set of specifications that defines a dynamic component system for Java in 1998 and developed by the OSGi Alliance found in 1999[39]. The model provides a mature and modular architecture to reduce software complexity and maintenance costs and enables a remote service management on the fly. Utilising OSGi modules to build software systems is like an assembly using in-house and off-the-shelf spare parts. It significantly increases development productivity and makes them much easier to modify and evolve. The OSGi technology is adopted by many software vendors or open source organisations such as IBM[40], Oracle[41], JBoss[42], Eclipse[43], etc.

Eclipse foundation, recognising the advantages of OSGi, implements the specifications of OSGi core frameworks: Equinox[44] as an underlying infrastructure platform to support all of Eclipse. Eclipse is an open source and extensible platform based on Java. Eclipse provides a plug-in model based on Equinox OSGi core framework implementation. A plug-in is a structured package which embodies a certain type of service. It can define one or more extension points where other plug-ins can add new functionalities or can use the extensions provided by other plug-ins. It can be exported as a directory or as a jar which can be added to other applications, and be grouped into features which can be distributed and installed into other applications. An Eclipse based program or product contains one or more plug-ins, which can be added, replaced or removed to alter the functionality of the program.

Eclipse RCP technology is a subset of Eclipse. It contains the core framework and services for building an Eclipse based application. As the name "rich client" implies, Eclipse RCP is an excellent development framework for the applications that interacts with application servers, database servers, or other backend resources to deliver a rich user experience on the desktop. So far, Eclipse RCP has been applied to much domain-specific software development for such as banking, automotive, medical, space exploration, etc.

The workflow client based on Eclipse RCP is able to provide three excellent features:

[39] http://www.osgi.org/Main/HomePage
[40] http://www-01.ibm.com/software/websphere/
[41] http://www.oracle.com/us/corporate/Acquisitions/bea/index.html
[42] http://www.jboss.org/overview/
[43] https://www.eclipse.org/
[44] http://eclipse.org/equinox/

(1) Plug-and-play extensibility. Any new functionality can be capsulated as a plug-in which complies with RCP plug-ins specifications and plugged into the client at build time, deployment time or even runtime. The new plug-ins can utilise the extension points of the existing plug-ins of the client and provide new extension points to other plug-ins.

(2) Flexible user interface layout. The client provides the flexible drag-and-drop tab-based layout management to users and enables various user interface components, including property sheets, viewers, console windows, and outline windows and so on, to organise around the editing area in client.

(3) Dynamic remote maintenance. The Eclipse RCP based workflow client can implement dynamic remote maintenance on the fly, including upgrading, troubleshoot, updating and so on, without the needs of users learning more IT or workflow management knowledge.

The cloud side is Web service-based applications which are developed based on the technology of Apache Axis 2[45]. Axis 2 is the third generation Web Services/SOAP/WSDL engine. It provides a complete object model and a modular architecture that makes it easy to add functionality and support for new Web services related specifications and recommendations. The advantages of Axis 2 for software development include:

- Send SOAP messages;
- Receive and process SOAP messages;
- Create a Web service out of a plain Java class;
- Create implementation classes for both the server and client using WSDL;
- Easily retrieve the WSDL for a service;
- Send and receive SOAP messages with attachments;

The cloud side based on Web services is able to provide some significant features:

- Plug-and-play extensibility. Any new functionality of cloud workflow service can be capsulated as a normal Web service regardless in any programming language and

[45] http://axis.apache.org/axis2/java/core/

deployed on an application server on the fly. The new functionality can be recognised by Axis 2 automatically.

- Complex object transports. Any complex serialisable object communicated between the client side and the cloud side can be serialised to byte input or output stream and transported over the Internet.

- Crossing firewall. The SOAP is based on HTTP. The Web services can cross the firewall without particular specifications on firewall.

Some graphical representations in SwinFlow-Cloud, such as process modelling, organisational modelling and process simulation and so on, use Eclipse Graphical Editing Framework (GEF)[46]. Eclipse GEF project provides a technology to create rich graphical editors and views. It is composed of three principle frameworks: Draw2D, GEF, and Zest. Draw2D is a lightweight drawing framework for rendering, layout, or printing graphical information without interactive behaviours. Zest is built on top of Draw2D, providing a JFace-like interface for easily binding a Java model with a Draw2D diagram. GEF is also built on top of Draw2D, providing rich APIs for creating an interactive diagram with advanced features, including palettes, drag-and-drop supports, a command stack for undoing and redoing commands, support for painting, and so on.

The graphical representations based on Eclipse GEF have some excellent features:

- Simplified development for graphical editing. The mature capsulations of underlying drawing APIs of native methods simplify system development and facilitate graphical representation to run across multiple platforms.

- Extensible graphical modelling. This feature is inherited from RCP plug-in model. Any new graphical modelling element can be capsulated into a plug-in and integrated into graphical editor to enhance the modelling capacities.

For navigation rule editing, we introduce ANTLR (ANother Tool for Language Recognition) to recognise and parse the custom expressions of navigation rules. ANTLR is a language tool that provides a framework for constructing recognisers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target

[46] http://www.eclipse.org/gef/

languages. A navigation rule in SwinFlow-Cloud is a Boolean expression with a textual character string and consists of the data variables with Java objects. ANTLR parses the expression definitions and generates a compiler to recognise the textual string and further calculate the value of the expression. ANTLR is a simple and highly efficient compiler generator and appropriate to the calculation of the navigation rules of SwinFlow-Cloud.

7.1.2 System development

SwinFlow-Cloud is an open source cloud workflow prototype to support large-scale instance-intensive workflows. The whole system has over 170 thousand lines of code, over 1300 Java class files and over 180 functionality packages.

For the system development environment, the development of the workflow client is Eclipse IDE for RCP Developers, which is a complete set of tools for developers who want to create Eclipse plug-ins, Rich Client Applications; while the service development of the cloud side is Eclipse IDE for Java EE Developers, which is the tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.

The system framework of the client side is built as a plug-in application based on RCP. Each management tool of workflow client is built as a perspective extension point. Each runtime service on cloud side is built as a Web service and deployed on cloud infrastructure.

We design a fundamental Java class structure to depict and represent the core data structure of SwinFlow-Cloud, as shown in Fig. 7.1. The top class is an abstract workflow entity, named as WorkflowEntity, which means that any data structure can be considered as a basic workflow entity at an abstract level and is constructed from the abstract entity. The entity is simple and only contains an identity, a name and owner properties. The data type of the owner property is WorkflowEntity. These properties indicate that all workflow entities have an identity and a name and an abstract owner. Using these properties, we can further construct a tree data structure, named as TreeNode, which means that most of the workflow entities can be further considered as a tree node and organised in a tree or a forest structure. The tree or forest structure provides efficient searching, adding, removing, and updating operations. On the basis of the structures, we design the version class which inherits from TreeNode class, named as Version, to represent the tree nodes with version number, status,

author, and update timestamp properties. On the basis of the fundamental Java classes, we design the other Java classes at multiple perspectives of workflow management, including Folder, Process, AbstractTask, Transition, AbstractOrganisation, ToolAgent, Rule and so on. Furthermore, we design a workflow entity container to implement workflow-specific collection operations. The concentrate classes will be further addressed in this Chapter.

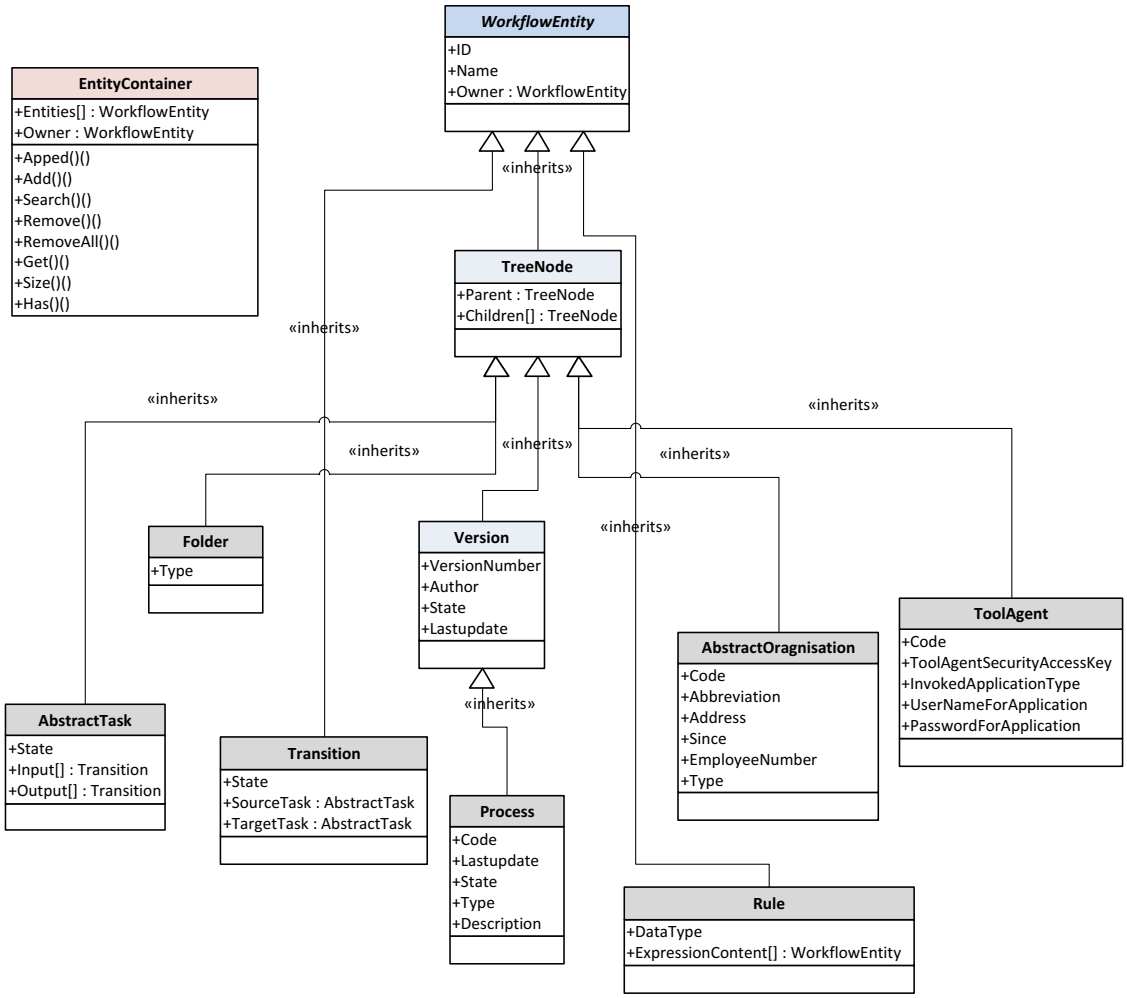

Figure 7.1 Fundamental Java class structure

7.1.3 System runtime environment

For the runtime environment of SwinFlow-Cloud, we choose a popular commercial cloud service infrastructure, Amazon AWS as runtime environment for deployment. Amazon AWS provides complete cloud services to support workflow enactment service performing, data persistency, image creation, system resource monitoring, billing, and network communication and so on.

Amazon Elastic Compute Cloud (Amazon EC2) offers various image configurations, named as Amazon Machine Image (AMI), to support workflow enactment service performing[47]. We choose Amazon Linux AMI with 64-bit, which can instantiate and launch an Amazon EC2 Linux instance in a short time period, as a target platform to migrate the workflow enactment service. Furthermore, we choose Amazon Linux AMI to create specific Amazon EC2 Linux instances to install and deploy the load balancing service, the alarm service, the WfSMI service, the billing service, and/or the auto-scaling service. In Amazon EC2, we can build various AMIs using the available resources to support image creation in the WfSMI service.

Apache Tomcat[48] is a popular Web application server for running Apache Axis 2 Web service engine[49]. We choose Tomcat as the application server to install and deploy a workflow enactment service, or other services mentioned above. Note that each service needs to be installed individually in a Tomcat of an Amazon EC2 Linux instance.

Amazon Simple Storage Service (Amazon S3) offers quick, easy and simple file storages over Internet like local disks[50]. We choose Amazon S3 to store the log files of the workflow enactment services and the library files or Java jar files for remotely accessing and invoking the external applications.

Amazon Relational Database Service (Amazon RDS) offers relational database service[51]. It supports the setup of various relational databases such as MySQL database, Oracle database, Microsoft SQLServer database, PostgreSQL database, etc. We choose MySQL as workflow storage management repository to implement workflow data persistency, including workflow relevant data, workflow control data, process definitions, organisation model, tool agents, etc.

Amazon CloudWatch offers monitoring services to watch the performance of workflow enactment services in Amazon EC2 Linux instances[52]. We choose the service to monitor the workflow enactment service components. For monitoring, we customise the resource

[47] http://aws.amazon.com/ec2/
[48] http://tomcat.apache.org/
[49] http://axis.apache.org/axis2/java/core/
[50] http://aws.amazon.com/s3/
[51] http://aws.amazon.com/rds/
[52] http://aws.amazon.com/cloudwatch/

utilisation metrics according to the specifications in the alarm service. The metrics can be used to estimate the system status.

Amazon AWS offers various pricing policies[53]. We obtain the policies from AWS to make the budget for planning the resource utilisation of SwinFlow-Cloud and monitor the budget on AWS.

7.2 Key functional component implementation

Using Eclipse RCP framework, we developed the build time functions in four perspectives: process, simulation, organisation, tool agent, etc. These perspectives enable workflow designers to focus on the management which they are responsible to.

7.2.1 Process manager

The process manager is used to query, create, modify, delete, and simulate a process definition.

Its fundamental Java class structure as depicted in Fig. 7.2 includes Process, AbstractTask, and Transition. The Process contains various tasks in its children array. The AbstractTask is the parent class of all concentrate Java classes for depicting tasks in a process which include StartPoint, EndPoint, ParallelJointPoint, ParallelSplitPoint, AssignTask, InvokeTask, ManualTask, DataVariable, and SubprocessPoint. The AssignTask class may contain one or more Assignment objects which implement assignment between data variables. The SubprocessPoint class can contain one or more SubprocessInput and SubprocessOutput objects to support sub-process invocation and access. The DataVariable class operates the workflow relevant data. The DataVariable objects use constructs expressions to calculate values for evaluation. The value can be used to make a decision such as navigating workflow execution.

The process manager uses “org.eclipse.ui.perspectives” extension point in Eclipse RCP to create a process perspective and consists of process viewer, property sheet (editor), outline

[53] http://aws.amazon.com/pricing/

viewer, process editor, process verifier, version controller, and simulator, etc. The manager implementation is shown in Fig. 7.3.

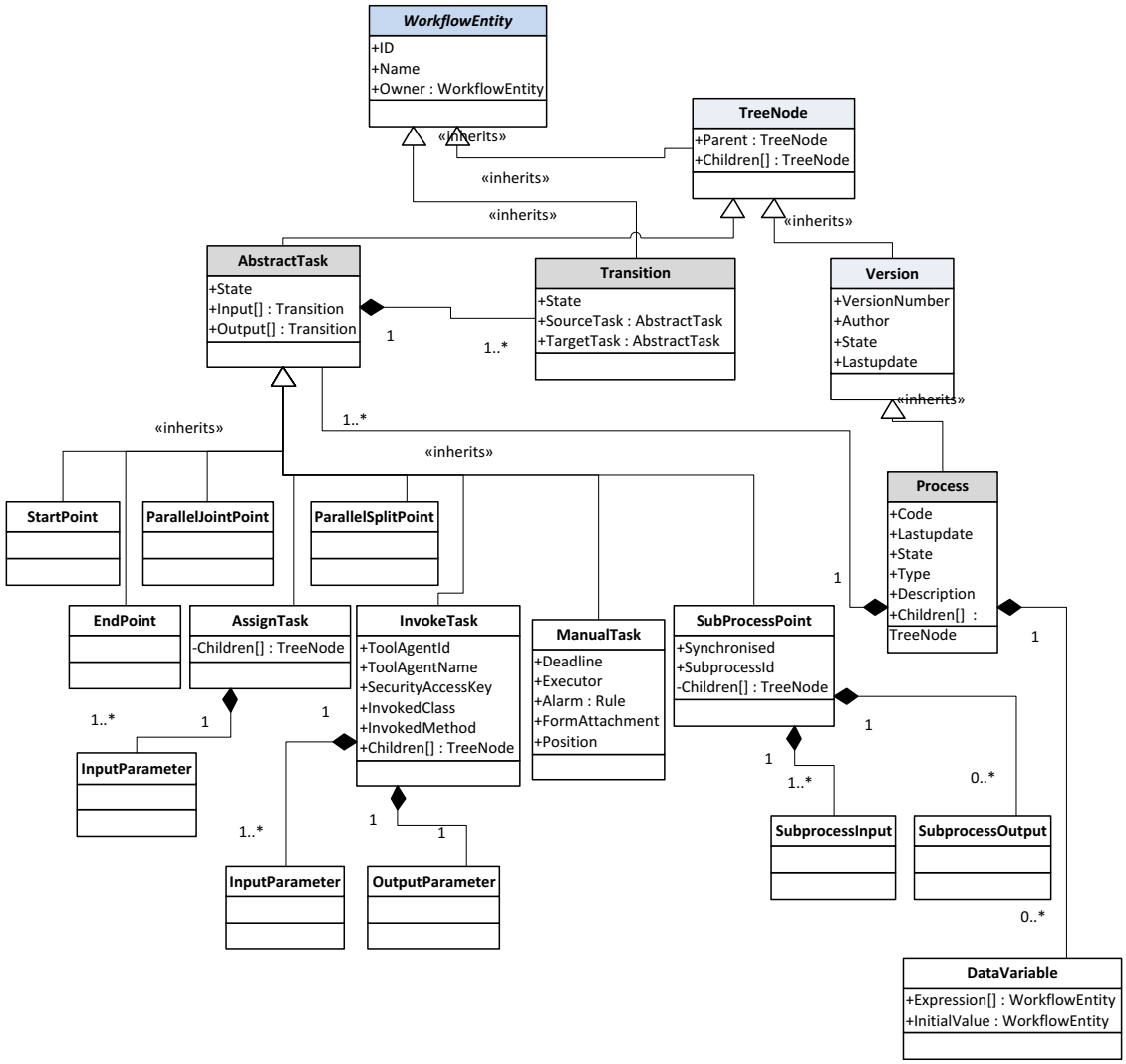

Figure 7.2 Java class structures from the process perspective

The process viewer can explore all processes owned by current organisation and organise the processes into folder tree structure on the left. Each folder can contain one or more processes. The process can be copied or moved between the folders and created, renamed, or deleted in the viewer. The property sheet can represent all properties of a task or a process and edit or modify the properties. The outline viewer can assist to select or locate a graphical element in a process and provide a snapshot of the process. The process editor may edit a process and changed any element and aligned to the left, right, centre, top or bottom. A workflow designer can drag and drop new graphical elements into the editor for modelling a

process, and define new data variables. The new data variable can be specified as multiple data types including BYTE, INTEGER, DOUBLE, STRING, etc. Each new data variable can be specified an initial value. The initial value can be a constant or an expression which needs the expression editor to edit. The version management modular can manage the status of process version. It can create, lock or unlock a draft version, and specify the version author and timestamp, and create a fixed version with a new version number, and release a versioned process to workflow enactment service for execution.

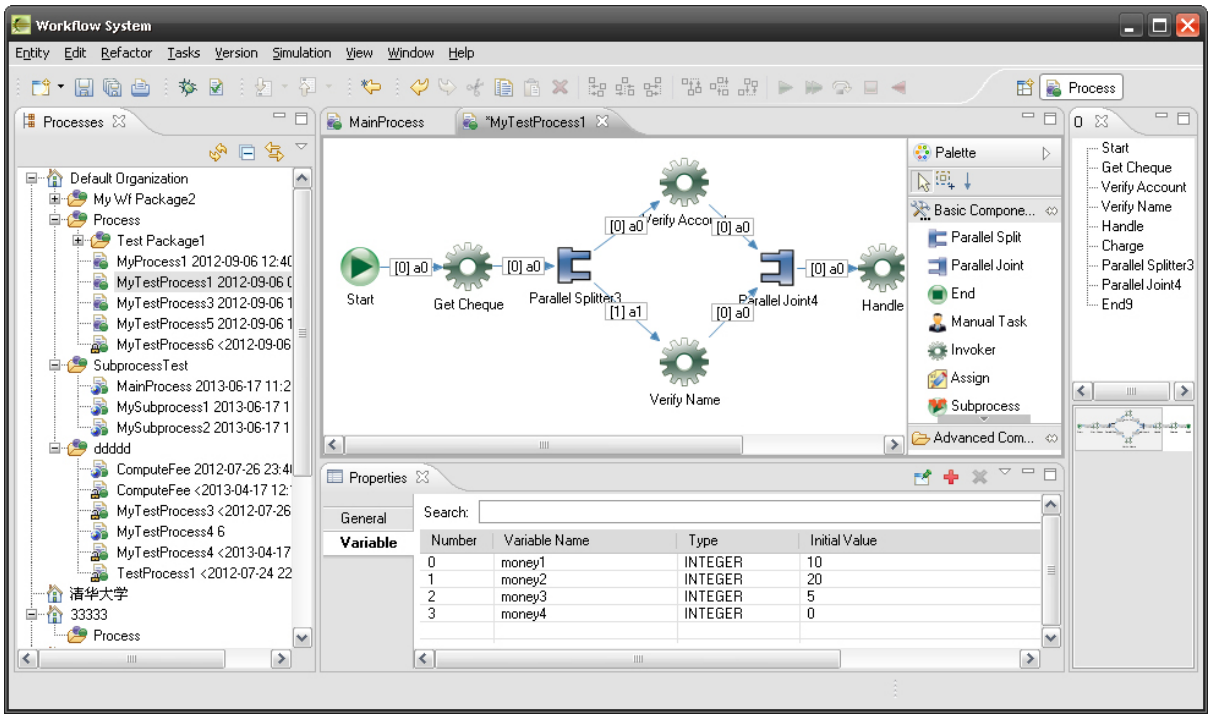

Figure 7.3 Process manager

The simulator can test a process through simulating the runtime configuration to verify the runtime correctness, as shown in Fig. 7.4. The simulator consists of process simulation editor, case viewer, variable watch viewer, console, etc. The process simulation editor can dynamically represent the process execution paths between the tasks with different colours. The case viewer can represent all the simulating process definitions including the sub-processes in these processes and can control the process to execute, pause, resume, terminate, or remove, etc. The variable watches viewer can represent the status of a process or a task or runtime values of data variables in the process. The console can print the execution logs or prompts of each step during execution of a process. If the execution has any errors or exceptions, the console can print the details of them like the console of Eclipse IDE.

The expression editor shown in Fig. 7.5 can define a new expression. The expression can be used in a navigation rule, an initial value, sub-process input, sub-process output or an assignment, etc. The editor utilises the compiler generated by ANTLR and all primary calculation operations of data variables. It can check the syntax errors before finishing the editing.

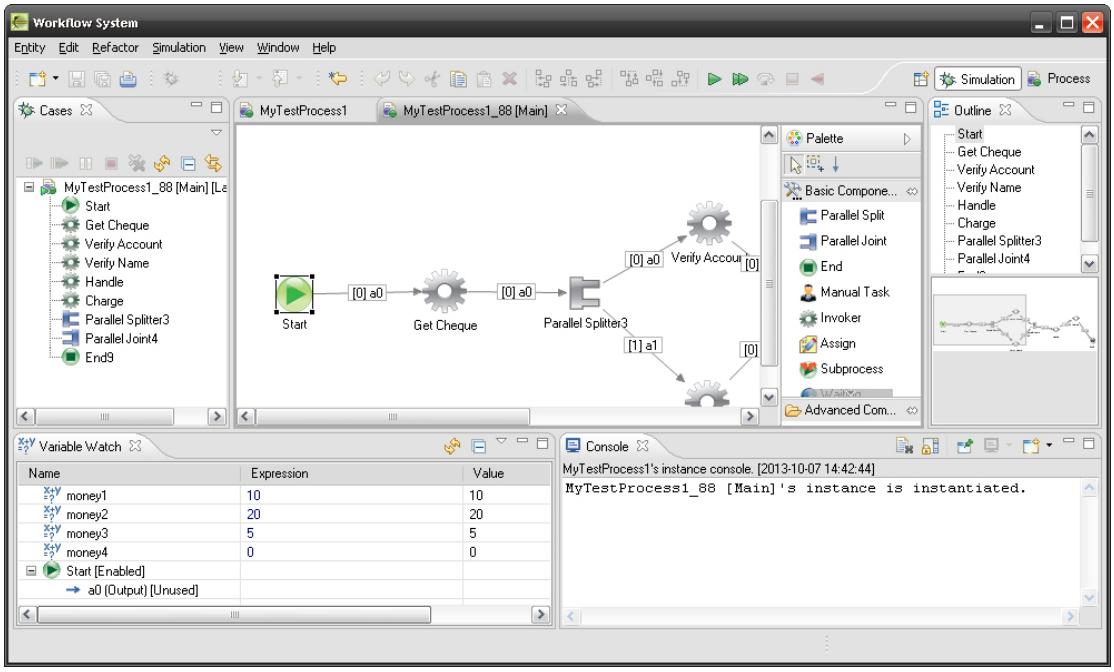

Figure 7.4 Process simulator

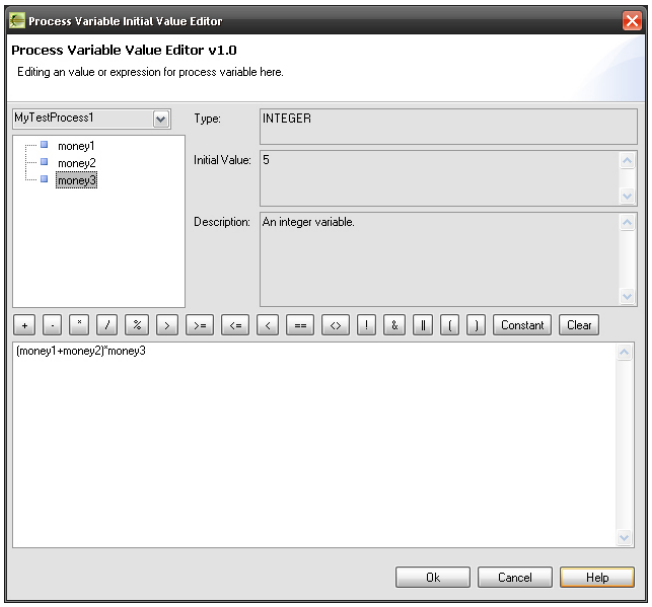

Figure 7.5 Expression editor

7.2.2 Organisation manager

The organisation manager is used to create, modify, delete an organisation and its structure, divisions, departments, project teams, positions, project roles, and users.

Its fundamental Java class structure, as shown in Fig. 7.6, includes AbstractOrganisation, AbstractPosition, Organisation, Department, Division, ProjectTeam, Position, ProjectRole, and User.

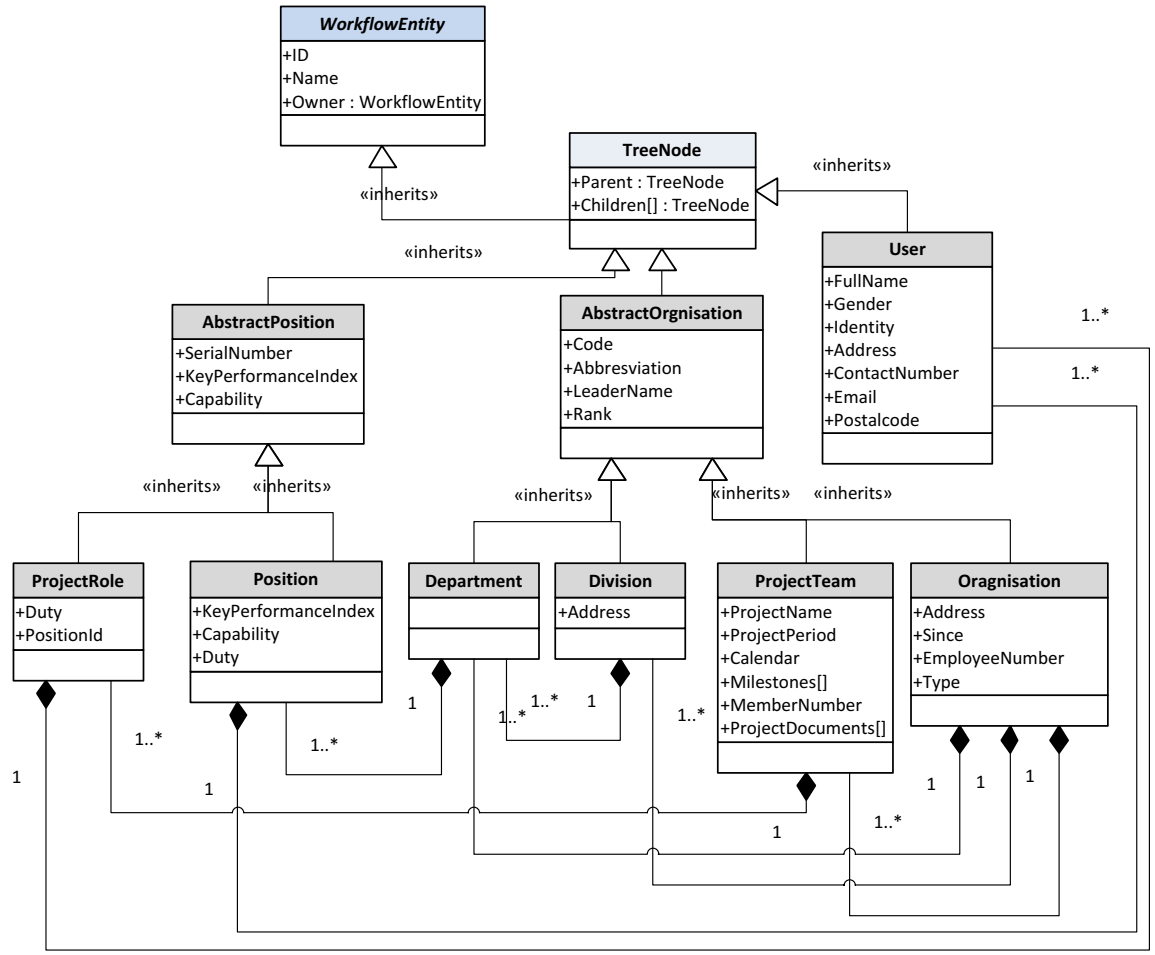

Figure 7.6 Java class structures from the organisation perspective

The AbstractOrganisation class is a parent class of Organisation class, ProjectTeam class, Division class, and Department class. The AbstractPosition class is a parent class of ProjectRole class and Position class. An Organisation object may contains one or more Division objects, Department objects, and ProjectTeam objects. A Division object may

contains one or more Division objects, Department objects and ProjectTeam objects. A Department object contains one or more Position objects but does not contain Division objects. A ProjectTeam object may contains one or more ProjectRole objects but does not contain Department objects and Position objects. A Position or ProjectRole object contains one or more User objects. But a Position object does not contain ProjectRole objects and a ProjectRole object does not contain Position objects.

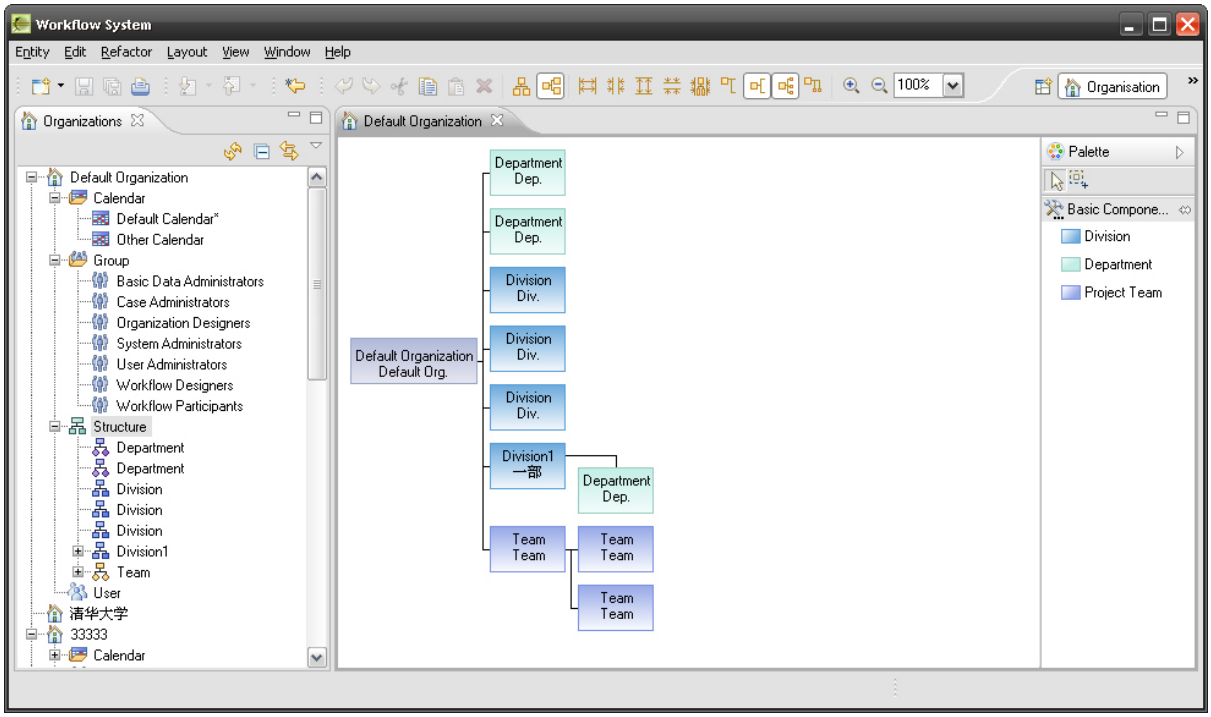

Figure 7.7 Organisation manager

The manager consists of organisation structure viewer and organisation structure editor as shown in Fig. 7.7. The viewer can explore all organisations, which are visible for the permitted users, and activate creation, modification, deletion, and renaming action to update the organisation properties. The organisation structures include division, department, project team, position, project role, and authority group which can authorise user operation authorities.

In the manager, an organisation structure consists of division structure, department structure, position structure, and project team structure. These structures can be edited in a structure editor. A designer can drag and drop add and delete structure components and align the components to the left, right, middle, top, and bottom, and transform the components structures, including rotate 90 degrees, flip vertical or horizontal and so on, and modify the

spaces between the components. The editor can zoom in or out for changing the size of graph of the structures at 400%, 200%, 100%, 50%, 20%, etc. The manager can shift to other managers through the swift button on the top-left corner. Like the process manager, the manager has also a property editor which is not shown in Fig. 7.7. The designer can modify or update the properties of any component in organisation structures.

7.2.3 Tool agent manager

The manager is to create, modify, or delete a tool agent. The fundamental Java class structure is shown in Fig. 7.8.

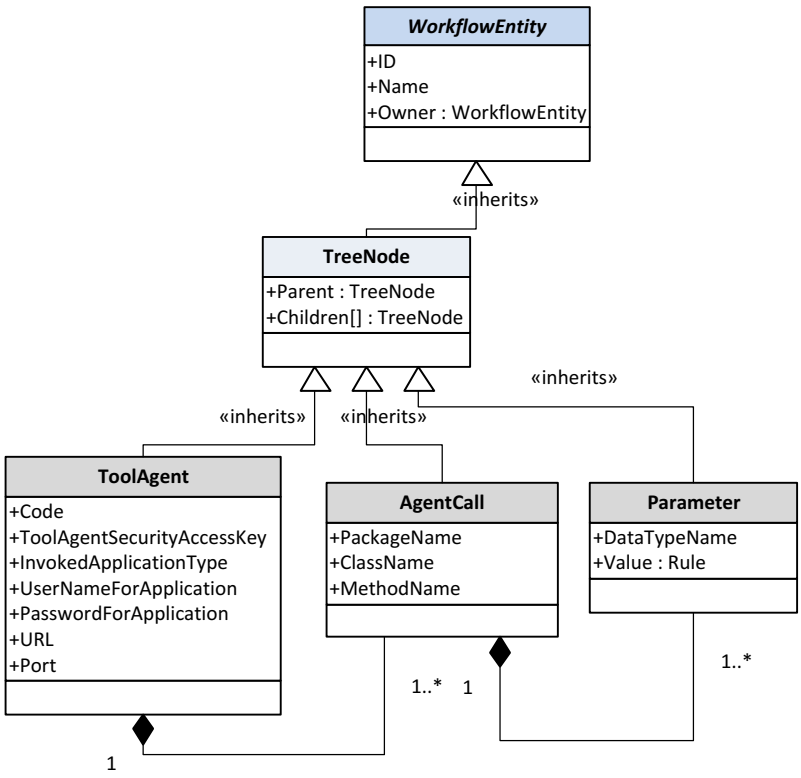

Figure 7.8 Java class structures from the application perspective

The ToolAgent class, AgentCall class and Parameter class inherit from the TreeNode class. ToolAgent stores the fundamental information for invoking an application, including Code, ToolAgentSecurityAccessKey, which controls the permissions to access the application, InvokedApplicationType, which indicates the application types such as Java applications, or WebService, or C++, etc., UserNameForApplication, which is provided by the applications to control accesses, PasswordForApplication, which is provided by the application to control

accesses, URL, which indicates the location of the invoked applications, Port, which indicates the port number of the invoked applications. One ToolAgent object may contain one or more AgentCall objects. The AgentCall class is used to describe the invoked methods in the invoked applications and contain PackageName, ClassTypeName, and MethodName, which can indicate the details of invocations. An AgentCall can only implement an invocation but there may be multiple invocations in one application. The Parameter class describes the input or output parameters in an AgentCall and contains DataTypeName, which indicates the data type of a parameter, and Value, which indicates the value of a parameter. The value can be a rule expression at build time. Therefore, the value is a rule object.

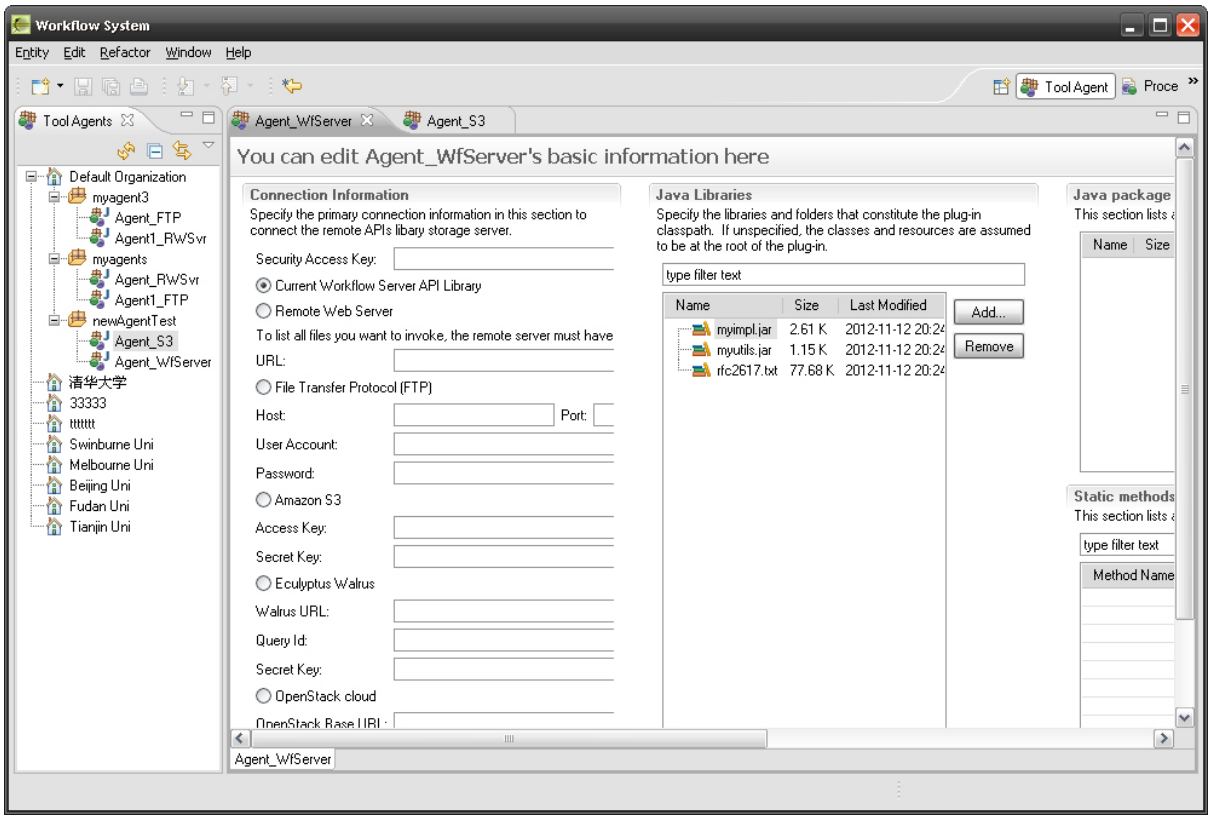

Figure 7.9 Tool agent manager

The manager consists of tool agent viewer on the left and tool agent editor on the right as shown in Fig. 7.9. The viewer can explore all tool agents in an organisation and organise these agents into the tree folder structure and can create, modify, delete, search copy, and rename a tool agent. The editor consists of three components: tool agent property detail editor, package or library explorer, class selector and method selector. The detail editor can represent all property details of a tool agent and is able to update the details including security access key, host, port, types of the invoked applications, etc. The package or library explorer

can list the library or package file details on the specified URL or host and select a file for specifying the class. The class selector can support a selection of Java class to list the static methods in the class. The method selector can select one or more static methods to create one or more AgentCalls. Each AgentCall can support invocation of Java static method.

The manager is developed and implemented through using advanced Java reflect technology. Hence, we can test the availability of Java methods through specifying an AgentCall.

7.2.4 Workflow enactment service

The workflow enactment is a fundamental functional service to support workflow management and consists of three core modular: workflow engine, task transaction engine, and navigation engine.

7.2.4.1 Workflow engine

The workflow engine is a suite of algorithms, including workflow engine algorithm (Fig. 7.10), initialisation algorithm (Fig. 7.11), task running algorithm (Fig. 7.12), and process suspending algorithm (Fig. 7.13).

Algorithm 1: **Workflow engine**
Input: *wfen*; // java object;
Output:

```
01. var: proc;          // a process object;
02. var: enabledTaskQueue; runningTaskQueue; exceptionTaskQueue; // queues
03.
04. if (wfen is a identity of a process) // this wfen is a main process
05.   { proc = createNewProcess(wfen); // create a new process instance
06.   { initialise(proc);                    // initialise the process instance
07. if (wfen is subprocessPoint of a process) // this wfen is a sub-process
08.     if (wfen.subprocess is null)
09.       { proc = createNewProcess(wfen);
10.       { initialise(proc);
11.       { addProcessIntoSubprocessPoint(wfen, proc); // add new process to parent
12.     else
13.         proc = wfen.subprocess;
14. if (proc.state == LAUNCHED)  // the process instance is launched
15.     proc.state = RUNNING;       // change the state to the running state
16. try
16.     while (true)
18.       { run(getEnabledTaskFromQueue(enabledTaskQueue); // get a task instance to run
19.       { if (proc is completed) // if the process instance completed
20.           { proc.state = COMPLETED; // change the state to the completed state
21.           { break;
22. catch
23.     if (proc has exception) // if the process instance has any exception
24.       { suspendProcess(proc); // suspend the process instance
25.       { proc.state == SUSPENDED; // change the state to the suspended state
26.       { rollbackAllActiveTransactions(); // roll back all started transactions
```

Figure 7.10 Algorithm for workflow engine

The workflow engine will invoke three critical methods, including initialisation algorithm (Fig. 7.11), task running algorithm (Fig. 7.12), and process suspending algorithm (Fig. 7.13), to complete the process instance. Algorithm 1 has a loop to get new enabled tasks to run. Once the EndPoint task is ended, the the process will complete.

Algorithm 2: **Initialise**
Input: *proc*; // a process object;
Output:

```
01. setStartTimestamp(proc); // set the start time stamp for the proc
02. proc.state = LAUNCHED; // change the state to the launched state
03. for (each child: proc.children) // if it has children, specify states of children
04.     if (child is a task)
05.         child.state = UNENABLED;
```

Figure 7.11 Algorithm for initialisation

```
Algorithm 3: Task_run
Input: task;      // a task object;
Output:
```

```
01. if (task is one of {StartPoint, EdnPoint, InvokeTask, WaitTask,
02.                     AssignTask, ManualTask, SubprocessPoint,
03.                     ParallelJointPoint, ParallelSplitPoint}) {
04.     // if task is one element class in the set above
05.     // crate a new transaction engine to execute a transaction
06.     createTaskTransactionEngine(createTransaction(task)).run();
07. }
```

Figure 7.12 Algorithm for task running

Algorithm 4 (Fig. 7.13) uses iteration to change the state of current process instance and its sub-processes.

```
Algorithm 4: Suspend_process
Input: proc;      // a process object;
Output:
```

```
01. proc.state = SUSPENDED; // change current state to the suspended state
02. for (each child: proc.children) // its all sub-processes will change state.
03.     if (child is SubprocessPoint && child.subprocess is not null &&
04.         (child.subprocess.state == LAUNCHED||child.subprocess.state == RUNNING))
05.         suspend_process(child.subprocess);   // iterate to call this algorithm.
```

Figure 7.13 Algorithm for process suspending

7.2.4.2 Task transaction engine

The task transaction engine (Fig. 7.14) can execute a task transaction and control the transaction rollback for any exception. The transaction engine is designed as a simple engine to meet requirement R2 addressed in Section 3.2.

Algorithm 5: **Transaction_running**
Input: *trans*; // a transaction object;
Output:

```
01. prepare(trans); // preparation can initialise transaction object;
02.                 // backup the data into cache for rollback;
03. trans.state = PREPARED; // change state of a transaction to the prepared state
04. var ne = createNavigationEngine(trans); // create a new navigation engine ne
05. try
06.     handleInputTransitions(ne); // ne is used to handle input transitions
07.     commit(trans); // commit the current transaction for completing it
08.     trans.state = COMMITED; // change state of transaction to commited state
09.     handleInputTransitions(ne); // ne is used to handle output transitions
10. catch
11.     rollback(trans); // roll back the transaction
12.     trans.state = EXCEPTION; // change the state to the exception state
13.     handleInputTransitions(ne); // rollback all input transitions
```

Figure 7.14 Algorithm for transaction running

As shown in Fig. 7.14, the system will create a transaction for each task instance to execute. The transaction engine can handle the input and output transitions and commit the executed results to the process instance for updating. The transactions are sequential on same execution path but concurrent on the parallel execution path.

7.2.4.3 Navigation engine

The navigation engine is to navigate the input and output transitions of the specified task. The engine needs to clear the status of the input transitions and activate the status of the output transitions of the specified task. Therefore the engine is composed of two algorithms: the navigation input transition algorithm (Fig. 7.15) and the navigation output transition algorithm (Fig. 7.16). The two algorithms are executed in the task transaction engine to support calculation of navigation rules for updating the states of the input or output transitions of the task instance.

Algorithm 6: **Navigate_input_transitions**
Input: *task*; // a task object;
Output:

```
01. if (task is ParallelJointPoint) // if task is parallel joint point
02.   var allCompleted = true;
03.   for (each input: task.Inputs) // all input transitions are completed
04.       if (input.state != ENABLED) // if not all completed
05.           allCompleted = false;   // it means one or more task instance before
06.           break;                  // the current parallel joint point does not complete
07.   if (allCompleted)  // if all completed, mark the completed state
08.       for (each input:task.Inputs)
09.           input.state = COMPLETED;
10. else
11.    for (each input: task.Inputs) // if task is not parallel joint point
12.        if (input.state == ENABLED) // find the enabled transition
13.          input.state = COMPLETED;
14.          if (input.sourceTask is not ParallelSplitPoint) // mark all target tasks on
15.              for (each output: input.sourceTask.Outputs) // the output transitions
16.                  if(input != output && output.state==ENABLED) // as unenabled
17.                    out.state = UNENABLED;
18.                    out.targetTask.state = UNENABLED;
```

Figure 7.15 Algorithm for navigating input transitions

Algorithm 7: **Navigate_output_transitions**
Input: *task*; // a task object;
enabledTaskQueue; // a queue contains enabled tasks;
Output:

```
01. if (task is ParallelSplitPoint) // if task is parallel split point
02.     for (each output: task.Outputs) // its all output transitions will be
03.         output.State = ENABLED; // enabled and all target task instances
04.         output.TargetTask = ENABLED; // on the output transitions will
05.         enqueue(enabledTaskQueue, output.TargetTask); // be enabled
06. else
07.   if (task.Outputs is null)
08.       return;
09.   for (each output: task.Outputs) // if task is not parallel split point, then
10.       output.State = ENABLED; // the first transition with navigation
11.       output.TargetTask = ENABLED; // rules will be enabled
```

Figure 7.16 Algorithm for navigating output transitions

7.3 Key non-functional component implementation

This section represents the implementation of the key non-functional components. In the non-functional components, the core is the alarm and scaling estimation models.

7.3.1 Alarm service

The implementation is the alarm estimation model which consists of four core algorithms: the alarm estimation algorithm (Fig. 7.17), the getting composite index algorithm (Fig. 7.18), the quadratic curve fitting algorithm (Fig. 7.19), and Markov predict algorithm (Fig. 7.20)

Algorithm 8: **Alarm_estimate**
Input: $u_{ij}(t), c_{ij}, i = 1..n, j = 1..m, t \in T$;
Output: *notification*; // the notification will be sent to the auto-scaling service

---

01.$\{r_{ij}(t), w_{ij}(t), \Delta w_{ij}(t), I_j(t), \bar{I}(t), \Delta\overline{w_i}(t)\}$ = Get_composite_index($u_{ij}(t), c_{ij}$);
02. **var**: *tendency* = Quadtractic_curve_fitting($\bar{I}(t)$); // use quadratic curve to fitting $\bar{I}(t)$
03. **var**: *state*;
04. **if** (*tendency* is **true**)
05. *state* = Makov_prediction($\bar{I}(t)$); // use Markov chain to predict
06. **if** (*state* is *EXBY* || *state* is *UNDL*) // if overloading or under loading, send notification
07. **Return** createNotification($\{r_{ij}(t), w_{ij}(t), \Delta w_{ij}(t), I_j(t), \bar{I}(t), \Delta\overline{w_i}(t)\}$);
08. **else**
09. **Return** null;

Figure 7.17 Algorithm for alarm estimation

Algorithm 9: **Get_composite_index**
Input: $u_{ij}(t), c_{ij}, i = 1..n, j = 1..m, t \in T$;
Output: $\{r_{ij}(t), w_{ij}(t), \Delta w_{ij}(t), I_j(t), \bar{I}(t), \Delta\overline{w_i}(t)\}$;

---

01. **var**: $w_0, r_{ij}(t), w_{ij}(t), \Delta w_{ij}(t), I_j(t), \bar{I}(t), \Delta\overline{w_i}(t)$;
02. $w_0 = 1/n$; // get the initial weight
03. **for** (**var** j = 1; j <= m; j++)
04. **for** (**var** i = 1; i <= n; i++)
05. $r_{ij}(t) = u_{ij}(t) / c_{ij}$; // the each resource utilisation in each workflow enactment service
06. **var**: $mean_j$;
07. **for** (**var** j = 1; j <=m; j++)
08. **for** (**var** i = 1; i <= n; i++)
09. $mean_j = mean_j + r_{ij}(t)$ // compute the sum of $r_{ij}(t)$
10. **for** (**var** j = 1; j <= m; j++)
11. $mean_j = mean_j / n$; // compute the mean of the sum
12. **for** (**var** j = 1; j <= m; j++)
13. **for** (**var** i = 1; i <= n; i++)
14. $\Delta w_{ij}(t) = r_{ij}(t) - mean_j$; // compute $\Delta w_{ij}(t)$
15. $w_{ij}(t) = w_0 + \Delta w_{ij}(t)$; // compute $w_{ij}(t)$
16. **for** (**var** j = 1; j <= m; j++)
17. **for** (**var** i = 1; i <= n; i++)
18. $I_j(t) = I_j(t) + w_{ij}(t) \cdot r_{ij}(t)$;// compute $I_j(t)$ of each enactment service
19. **for** (**var** j = 1; j <= m; j++)
20. $\bar{I}(t) = \bar{I}(t) + I_j(t)$;
21. $\bar{I}(t) = \bar{I}(t) / m$; // get $\bar{I}(t)$ of all enactment services
22. **for** (**var** i = 1; i <= m; i++)
23. **for** (**var** j = 1; j <= n; j++)
24. $\Delta\overline{w_i}(t) = \Delta\overline{w_i}(t) + \Delta w_{ij}(t)$;
25. $\Delta\overline{w_i}(t) = \Delta\overline{w_i}(t) / m$; // get $\Delta\overline{w_i}(t)$ of each resource utilisation
26. **Return** $\{r_{ij}(t), w_{ij}(t), \Delta w_{ij}(t), I_j(t), \bar{I}(t), \Delta\overline{w_i}(t)\}$;

Figure 7.18 Algorithm for getting composite index

Algorithm 10: **Quadratic_curve_fitting**
Input: $\overline{I}(t), t \in T$; // the mean of composite indexes of all workflow enactment service
Output: **true** or **false**; // if monotonically increasing or decreasing, true; otherwise, false

---

01. **var**: $t1$, $t2$, $t3$, $t4$, sum1, sum2, sum3; // temp variables
02. **for** (**var** i=0; i<T; i++) // compute the temp variables for constructing the equation group (8)
03. $t1 = t1 + i$; // $\sum_{i=0}^{k-1} t$ in (8)
04. $t2 = t2 + i \cdot i$; // $\sum_{i=0}^{k-1} t^2$ in (8)
05. $t3 = t3 + i \cdot i \cdot i$; // $\sum_{i=0}^{k-1} t^3$ in (8)
06. $t4 = t4 + i \cdot i \cdot i \cdot i$; // $\sum_{i=0}^{k-1} t^4$ in (8)
07. $sum1 = sum1 + \overline{I}(i)$; // $\sum_{i=0}^{k-1} \overline{I}(t)$ in (8)
08. $sum2 = sum2 + i \cdot \overline{I}(i)$; // $\sum_{i=0}^{k-1} t \cdot \overline{I}(t)$ in (8)
09. $sum3 = sum3 + i \cdot i \cdot \overline{I}(i)$; // $\sum_{i=0}^{k-1} t^2 \cdot \overline{I}(t)$ in (8)
10. // construct equation group (8) in Subsection 6.3.6
11. $$\begin{pmatrix} T & t1 & t2 \\ t1 & t2 & t3 \\ t2 & t3 & t4 \end{pmatrix} \begin{pmatrix} a0 \\ a1 \\ a2 \end{pmatrix} = \begin{pmatrix} sum1 \\ sum2 \\ sum3 \end{pmatrix};$$
15. // get $a0$, $a1$, $a2$, construct the first derivative
16. **var**: *fdu* = **false**;
17. **var**: *fdd* = **false**;
18. **for** (**var** i=0; i<T; i++)
19. **if** ($a1 + 2 \cdot a2 \cdot i > 0$) // monotonically increasing
20. *fdu* = **true**;
21. **if** ($a1 + 2 \cdot a2 \cdot i < 0$) // monotonically decreasing
22. *fdd* = **true**;
23. **if** (! *fdu*⊕! *fdd*)
24. **Return true**; // if true, do Markov prediction
25. **else**
26. **Return false**;

Figure 7.19 Algorithm for quadratic curve fitting

Algorithm 11: **Markov_predict**
Input: $\overline{I(t)}, t \in T$; // the mean of composite indexes of all workflow enactment service
S; // as defined in Subsection 6.3.6, S contains {*IDLE*,*UNDL*,*EXBY*, ...}
Output: $s, s \in S$; // return a state in S, *UNDL*, *EXBY*, ...

---

01. **var**: *trProbMatrix*[S][S], // transition probability matrix
02. *trCounMatrix*[S][S], // transition counting matrix
03. *initDistribution*[S]; // initial distribution probability vector
04. **for** (**var** i = 0; i < T; i++)
05. **if** (i < T - 1)
06. *trCounMatrix*[*getState*($\overline{I(i)}$)][*getState*($\overline{I(i+1)}$)]+ = 1;
07. **for** (each *state*1: S) // compute the transition matrix
08. **var**: *sum* = 0;
09. **for** (each *state*2: S)
10. *sum* += *trCounMatrix*[*state*1][*state*2];
11. **if** (*sum* > 0)
12. **for** (each *state*2: S)
13. *trProbMatrix*[*sate*1][*state*2] = *trCounMatrix*[*state*1][*state*2] / *sum*;
14. *initDistribution*[*state*1] = *sum* / T; // compute the initial distribution probability
15. **var**: m[S] = *initDistribution*[S]·*trProbMatrix*[S][S]; // one step;
16. **Return** MaxProbability(m[S]); // return the state in S with maximum probability.

Figure 7.20 Algorithm for Markov prediction

7.3.2 Auto-scaling service

The implementation is the auto-scaling estimation model which consists of three core algorithms: the scaling analysis algorithm (Fig. 7.21), the auto-scaling algorithm (Fig. 7.22), and the auto-scaling algorithm (Fig. 7.23).

Algorithm 12: **Scaling_analyse**
Input: $\{c_{ij}, r_{ij}(t), \Delta w_{ij}(t), TH_{ij}, i=1..n, j=1..m, t \in T\}$;
Output: []; // this is a set to store the analysed overloaded $r_{ij}^{'}$

---

01. $\{r_{ij}^{'}(t), i=1..k, j=1..l\}$ = getRc ($\{\Delta w_{ij}(t)>0, i=1..n, j=1..m\}$);
02. **var**: *RC*; // *RC* is a set to store the overloaded $r_{ij}^{'}$
03. **for** (each $r_{ij}^{'}$: $\{r_{ij}^{'}(t)\}$)
02. **var**: *t*1, *t*2, *t*3, *t*4, *sum*1, *sum*2, *sum*3; // the temp variables for computing the equation
03. **for** (**var** *i* = 0; *i* < *T*; *i*++) // group in (12)
04. $t1 = t1 + i$; // $\sum_{i=0}^{k-1} t$ in (12)
05. $t2 = t2 + i \cdot i$; // $\sum_{i=0}^{k-1} t^2$ in (12)
06. $t3 = t3 + i \cdot i \cdot i$; // $\sum_{i=0}^{k-1} t^3$ in (12)
07. $t4 = t4 + i \cdot i \cdot i \cdot i$; // $\sum_{i=0}^{k-1} t^4$ in (12)
08. $sum1 = sum1 + r_{ij}^{'}(t)$; // $\sum_{i=0}^{k-1} r_{ij}^{'}(t)$in (12)
09. $sum2 = sum2 + i \cdot r_{ij}^{'}(t)$; // $\sum_{i=0}^{k-1} t \cdot r_{ij}^{'}(t)$in (12)
10. $sum3 = sum3 + i \cdot i \cdot r_{ij}^{'}(t)$; // $\sum_{i=0}^{k-1} t^2 \cdot r_{ij}^{'}(t)$in (12)
11. // construct equation group (12) in Subsection 6.3.8
12. $\begin{cases} \begin{pmatrix} T & t1 & t2 \\ t1 & t2 & t3 \\ t2 & t3 & t4 \end{pmatrix} \begin{pmatrix} b0 \\ b1 \\ b2 \end{pmatrix} = \begin{pmatrix} sum1 \\ sum2 \\ sum3 \end{pmatrix}; \end{cases}$
15. // get *b*0, *b*1, *b*2, construct the first derivative
16. **var**: *fdu* = **false**;
17. **for** (**var** *i* = 0; *i* < *T*; *i*++)
18. **if** ($b1 + 2 \cdot b2 \cdot i > 0$) // monotonically increasing
19. *fdu* = **true**;
20. **if** (!*fdu*)
21. $k_o = (th_5 - th_4)/T$;
22. **for** (**var** *i* = 0; *i* < *T*; *i*++)
23. **if** ($b1 + 2 \cdot b2 \cdot i > k_o$)
24. putToSet(*RC*, r_{ij});
25. **Return** *RC*;

Figure 7.21 Algorithm for scaling analysis

Algorithm 13: **Scaling_out**
Input: $RC = \{r_{ij}^{'}(t), i=1..k^{'}, j=1..l^{'}\} \subset \{r_{ij}^{'}(t), i=1..k, j=1..l\}$;
$\{c_{ij}, i=1..k^{'}, j=1..l^{'}\}$;
$\{c_{\max i}, i=1..k^{'}\}$; // $c_{\max i}$ is the maximum available resource capacity
// in billing budget plans offered by the billing service
Output: []; // the quantity range needs to be scaled out by

---

01. **var**: *C*; // *C* is a set that contains the minimum quantity of
02. // each resource need to be scaled
03. **for** (**var** *i* = 1;*i* <= $k^{'}$; *i*++)
04. **var**: *sc*; // sum of $r_{ij} \cdot c_{ij}$
05. **for** (**var** *j* = 1;*j* <= $l^{'}$;*j*++) // $C = \{sc_i, i=1..k^{'}\}$
06. *sc* = *sc* + $r'_{ij} \cdot c_{ij}$;
07. *sc* = *sc* / ((*th*$_2$+*th*$_3$) / 2); // [*th*$_2$, *th*$_3$) is the interval of the *NLWK* state
08. *sc* = *sc* / $c_{\max i}$; //
09. putToSet(*C*, *sc*);
10. **var**: $sc_{\min}$ = MAX(*C*); // *scm* is the minimum quantity of workflow enactment
11. // service that needs to be scaled out
12. **Return** [$sc_{\min}$, *m*]; // *m* is obtained in Subsection 6.3.8

Figure 7.22 Algorithm for scaling out

Algorithm 14: **Scaling_in**
Input: $\{I_j(t), \bar{I}(t), TH_j, i = 1..n, j = 1..m\}$;
Z; // the minimum quantity of workflow enactment services
Output: SCL_d; // the quantity range needs to be scaled in by

---

01. **var**: *UndL*;
02. **if** ($\bar{I}(t) > th_2$) // if $\bar{I}(t)$ is more than the th_2 of the *UNDL* state
03. { **for** (**var** j = 1; j <= m; j++) // find the servers on *UNDL* state
04. **if** ($I_j(t) < \bar{I}(t)$)
05. putToSet(*UndL*, $I_j(t)$); // put the servers in a set for
06. SCL_d = *UndL*.size; // scaling in
07. **else**
08. $SCL_d = m - Z$; // if $\bar{I}(t)$ is less than the th_2 of the *UNDL* state
09. // then only reserve Z servers
10. **Return** SCL_d

Figure 7.23 Algorithm for scaling in

7.4 Summary

In this chapter, we firstly gave an overview of the technologies developing SwinFlow and the motivations of choosing the technologies. Secondly, we surveyed the system development. Thirdly, we introduced the development environment of SwinFlow-Cloud: Amazon AWS and its characteristics. Next, we represented the implementation of key fundamental functional and non-functional components and algorithms to support SwinFlow-Cloud. The functional components include process manager, organisation manager, tool agent manager, workflow engine, task transaction engine, and navigation engine. The non-functional components include the alarm service and the auto-scaling service.

Chapter 8
Case Study, Experiments and Evaluation

This chapter uses the motivating example in Section 3.1 to represent a case study and deploy SwinFlow-Cloud on Amazon AWS for demonstration and evaluation based on the case study. The case is discussed in Section 8.1. Section 8.2 represents the experiments based on the case. Section 8.3 evaluates the results of the experiments.

8.1 Case study: mobile charge

8.1.1 Case analysis

In Section 3.1, we introduced the mobile charge workflow and its task structure as motivating example. In this section, the case will be further discussed in detail and modelled in SwinFlow-Cloud. The aim of the workflow is to process the CDRs and complete the charge of an SMS. The workflow consists of automatic tasks without manual interactions and is able to launch and execute automatically through external events triggering.

The first task invokes a specific application, named as *Collect CDRs*, to collect CDRs. The application accesses the APIs of external telecommunication exchange equipment to import CDRs. As defined in Section 3.1, CDR is a character string with some specific delimiters and can be stored in a String variable. This workflow defines a String variable array, denoted as *cdrs*[], to store multiple CDRs.

The second task invokes a specific application, named as *Pre-process CDRs*, and sends all CDRs as parameters into the application for transferring various equipment-specific CDR formats into a unified format. There may be multiple CDRs for one SMS. This task merges the

CDRs into one CDR. That is, a CDR is only to charge one SMS. The format can be recognised by other tasks of this workflow.

To simplify workflow modelling and improve performance, we merge the third, fourth, and fifth tasks in Section 3.1 into one task. The task further checks the duplications for one SMS and removes the duplications to guarantee that an SMS only has one CDR. As the parameter, the CDR of an SMS will be returned by the application and stored into a String variable, denoted as *cdr*.

The third task invokes a specific application, named as *Price CDR*, and sends the CDR to the application for pricing the SMS. The application can calculate the usage amount and generate a consumption cost according to the standard policies of China Mobile Limited SMS charge. The cost is an amount and stored into a Real variable, denoted as *cost*.

The workflow checks a Boolean variable, denoted as *b*, to determine whether there is any offer such as promotion plans or bonus in this SMS. If the variable indicates that there is an offer in the SMS, the workflow will execute the fourth task to price the CDR again; otherwise, workflow will execute the fifth task.

The fourth task invokes *Price CDR* again and sends the CDR to the application for pricing the SMS. Most of SMSs charge according to the standard price policies. The application can calculate the usage amount again and generate a consumption cost according to the special policies of China Mobile Limited SMS charge. The cost is overwritten into *cost*.

The fifth task invokes a specific application, named as *Store into DB*, and sends the CDR and the details to the database. The application can access the database through APIs.

The sixth task transfers the CDR and the detail to a specific application, named as *Generate Bill*, to generate a bill. The application returns a bill which is in a character string format and stores the bill into a String variable, denoted as *bill*.

The seventh task sends the bill into a specific application, named as *Charge Bill*, to charge. The charge has two patterns: real time charging and regular charging. To simplify the complexity of the case, we define this last task as following: the task sends the generated bill to a charge service.

8.1.2 Process modelling and tool agent defining

As discussed above, the tasks need to access multiple external applications, i.e., *Collect CDRs*, *Pre-process CDRs*, *Price CDR*, *Store into DB*, *Generate Bill*, and *Charge Bill*. We need to create six tool agents to support the workflow invocations to the applications. To idealise the experiments, we suppose that all applications can offer adequate capacity and performance to support workflow invocations. Furthermore, we need to create one String variable array, i.e., *cdrs*[], two String variables, i.e., *cdr*, *bill*, one Real variable, i.e., *cost*, and one Boolean variable, i.e., *b*.

According to the analysis, we can design the process in process manager, as shown in Fig. 8.1, and define the tool agent in tool agent manager, as shown in Fig. 8.2.

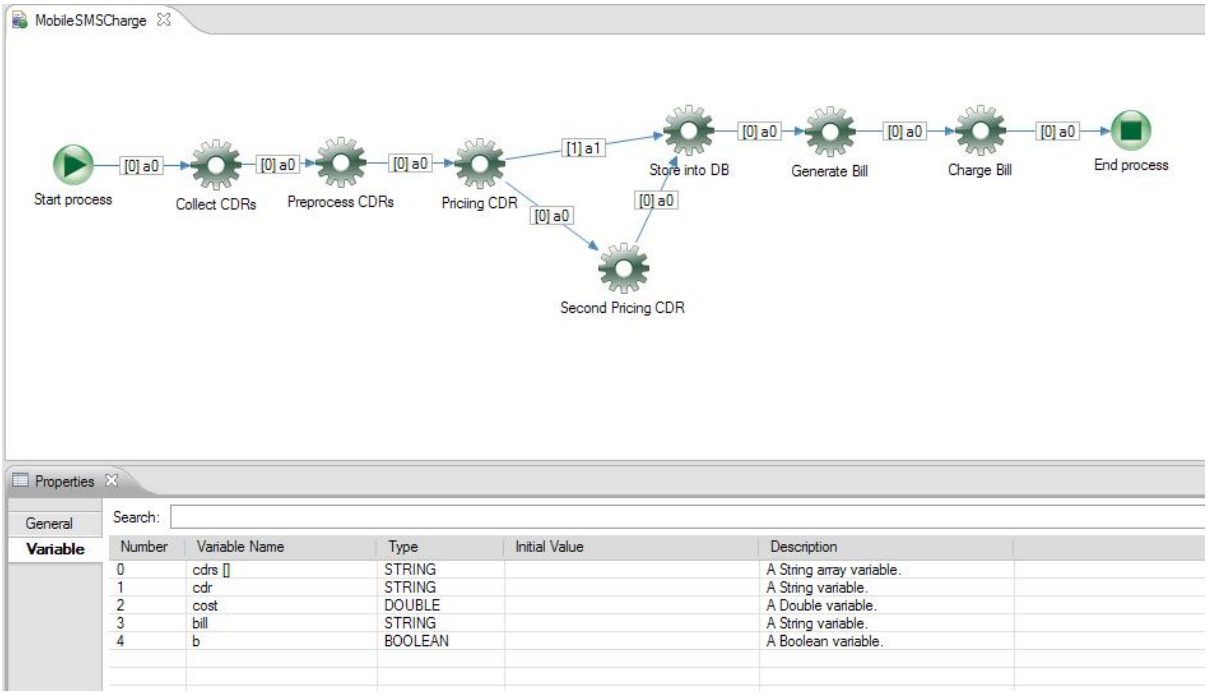

Figure 8.1 Mobile SMS charge process definition

We create a process, named as Mobile Charging, and add nine nodes and two navigation rules in it. Except the StartPoint and EndPoint, there are seven system invocation tasks for supporting workflow execution. Meanwhile, we define the data variables as listed above. The *b* variable can be used to support the navigation rule, e.g., *b* == TRUE means that the SMS needs to price again, while *b* == FALSE means that it does not need to do so.

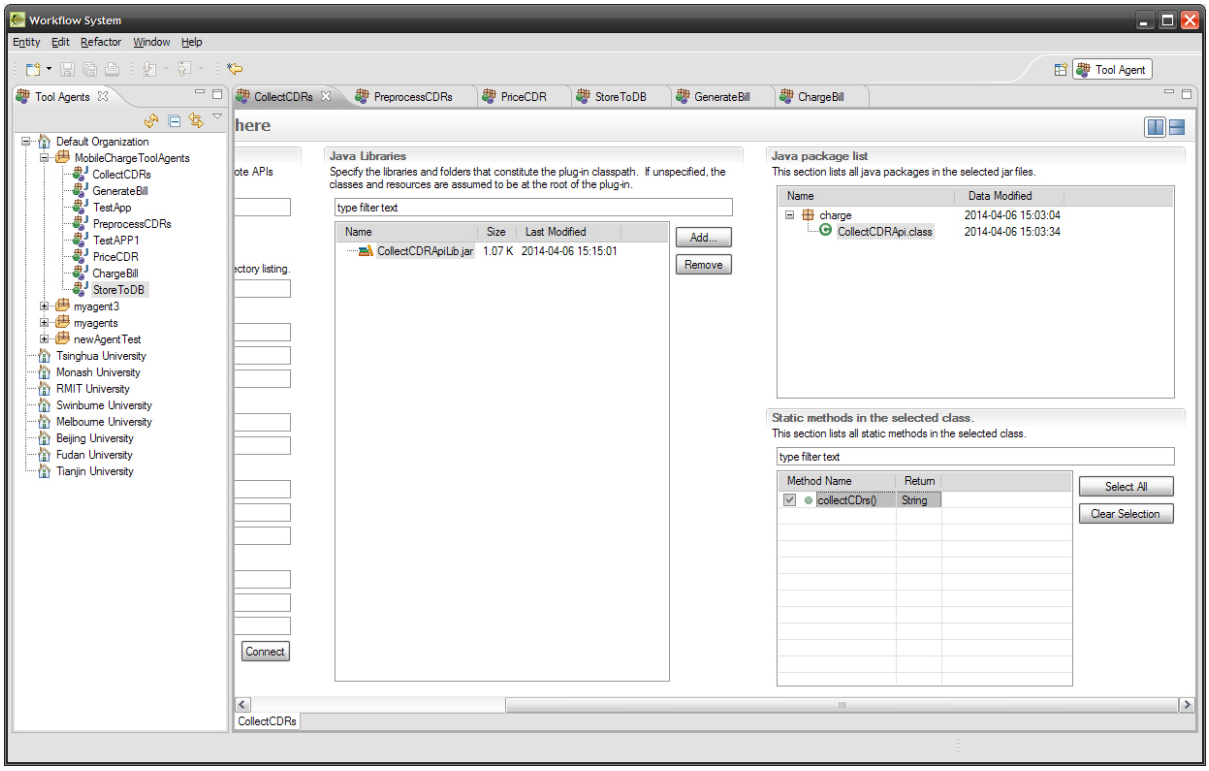

Figure 8.2 Tool agent definitions for mobile SMS charge

We define six tool agents to support the invocations of the workflow, including Collect CDRs, Pre-process CDRs, Price CDR, Store into DB, Generate Bill, and Charge Bill. The agents can transfer data in to or out from the external applications or services. The applications or services located in remote servers provide one or more static methods and specific parameters.

After defining the tool agents, we specify them in the invocation tasks in the Mobile SMS Charge, as shown in Fig. 8.3. In this figure, we only represent the specification of the first task, Collect CDRs. Other specifications are similar to that of the first task. The parameters of the tool agent can be specified using the expression editor. In this task, the obtained CDRs from the tool agent, *Collect CDRs*, is stored into the variable *cdrs*[].

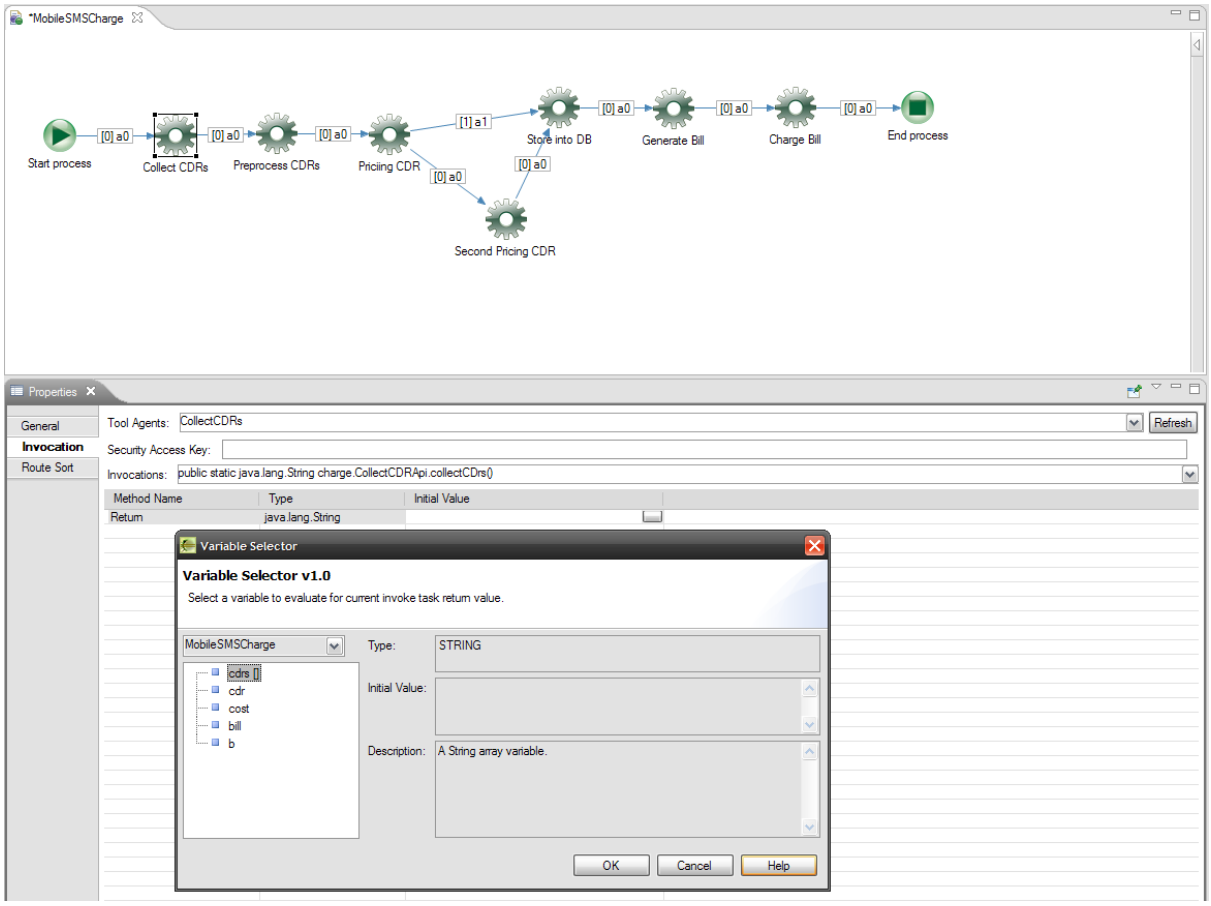

Figure 8.3 Specifying tool agents to mobile SMS charge

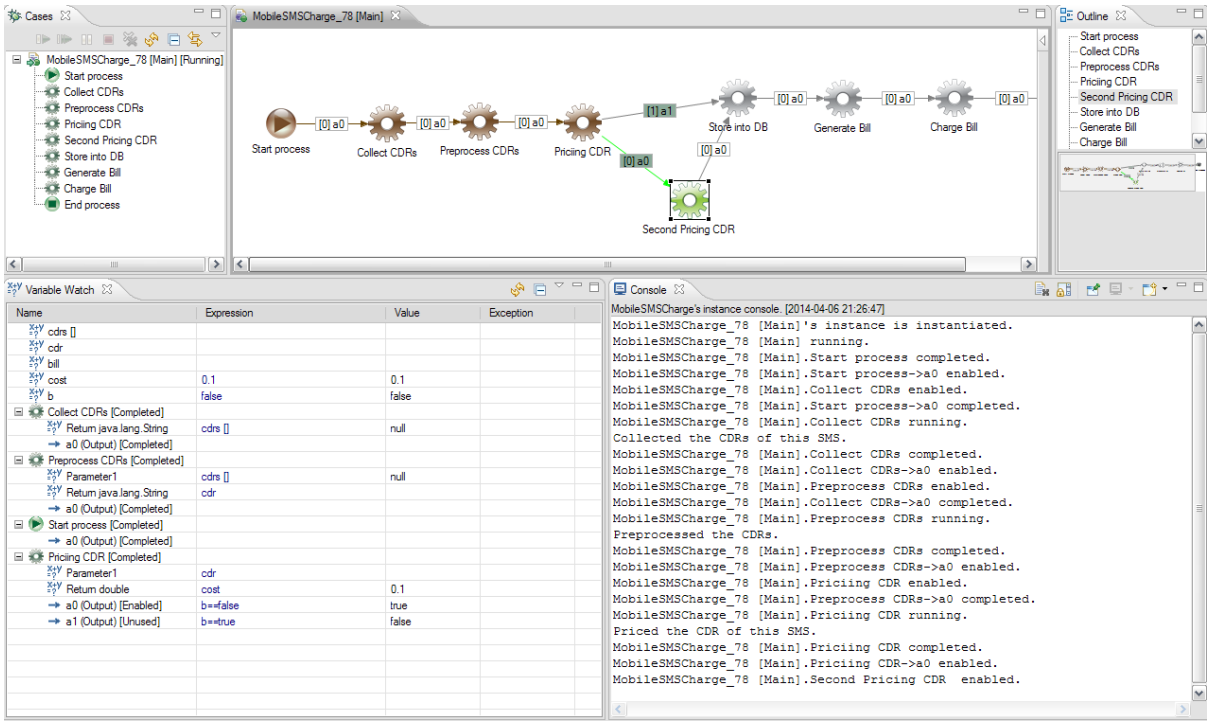

Figure 8.4 Simulation for mobile SMS charge

The Mobile Charging can be simulated in the simulation tool, as shown in Fig. 8.4. We can check the variable status and logs in the watch viewer and console. The simulation indicates that the Mobile Charging can execute successfully.

We create a new version of the Mobile Charging and release the process to the workflow enactment service for execution, as shown in Fig. 8.5 and Fig. 8.6.

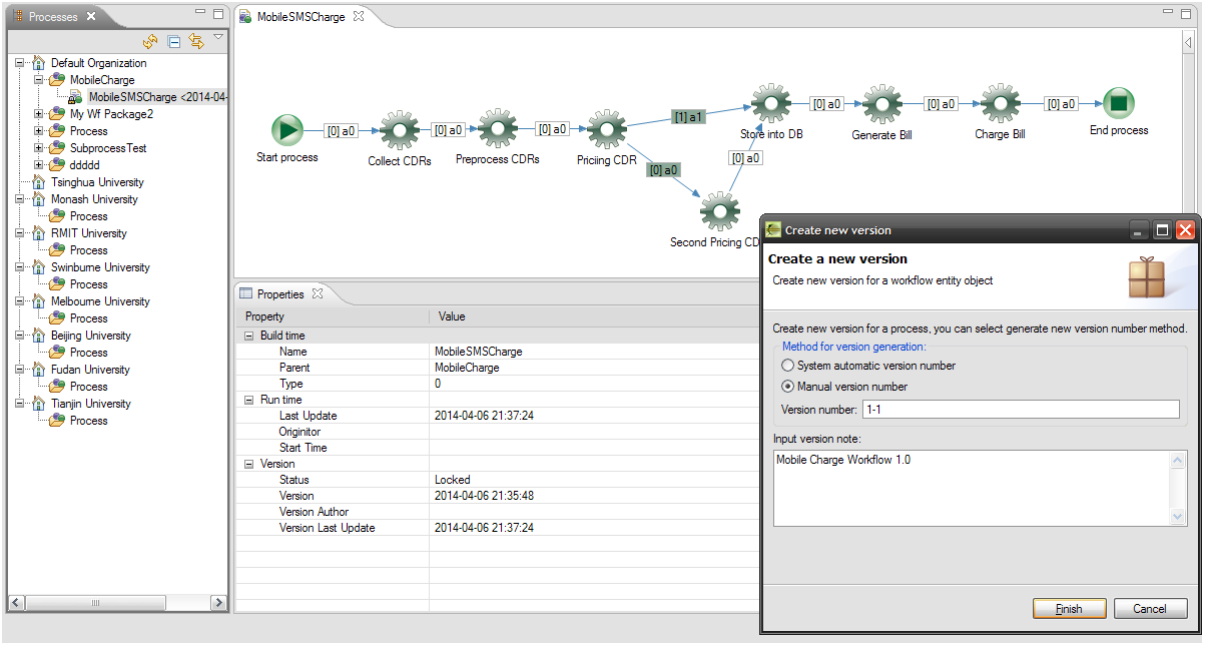

Figure 8.5 Creating a new version for mobile SMS charge

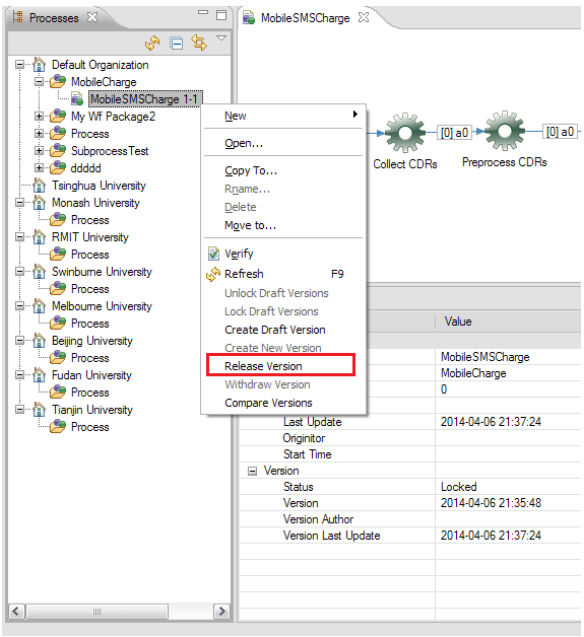

Figure 8.6 Releasing to workflow enactment service

8.2 Experiments

8.2.1 Experiment of functional components

The previous section modelled and defined the workflow, and released to the workflow enactment service. In this section, we will represent an experiment for the functional aspects addressed in Chapter 5. The experiment can be monitored by the workflow administrator and monitoring tool.

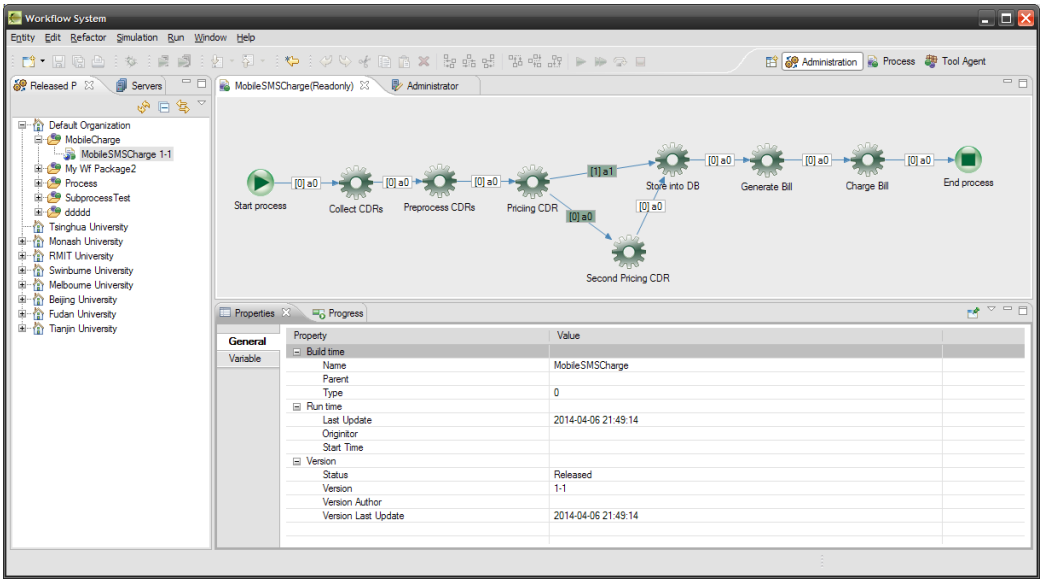

Figure 8.7 A released process definition

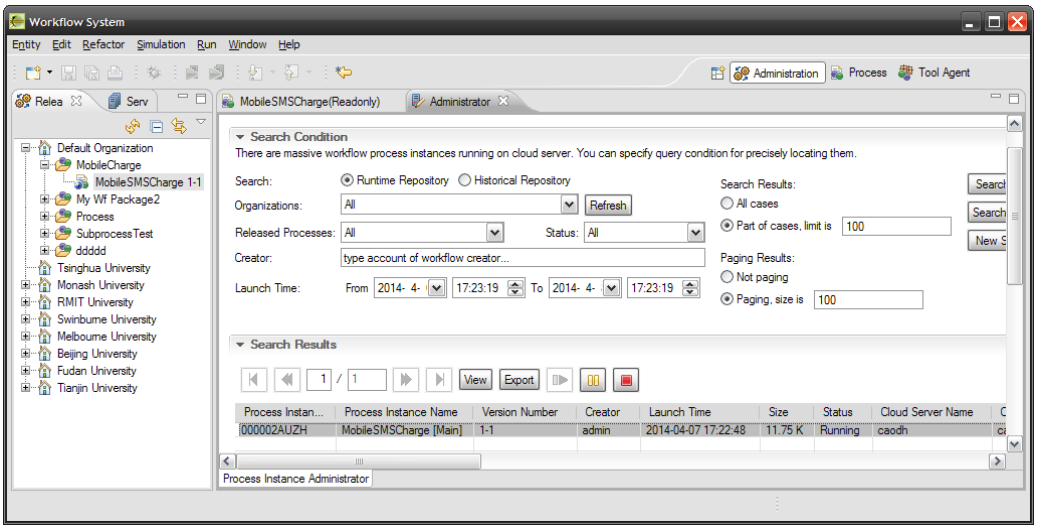

Figure 8.8 A running workflow instance in administration and monitoring tool

As shown in Fig. 8.7, the mobile SMS charge process definition is released to the workflow enactment service. We launch a workflow instance using this process definition and check the workflow instance in workflow administration and monitoring tool as shown in Fig. 8.8. According to our experiment, on average the workflow enactment service takes 2 seconds to execute a workflow instance.

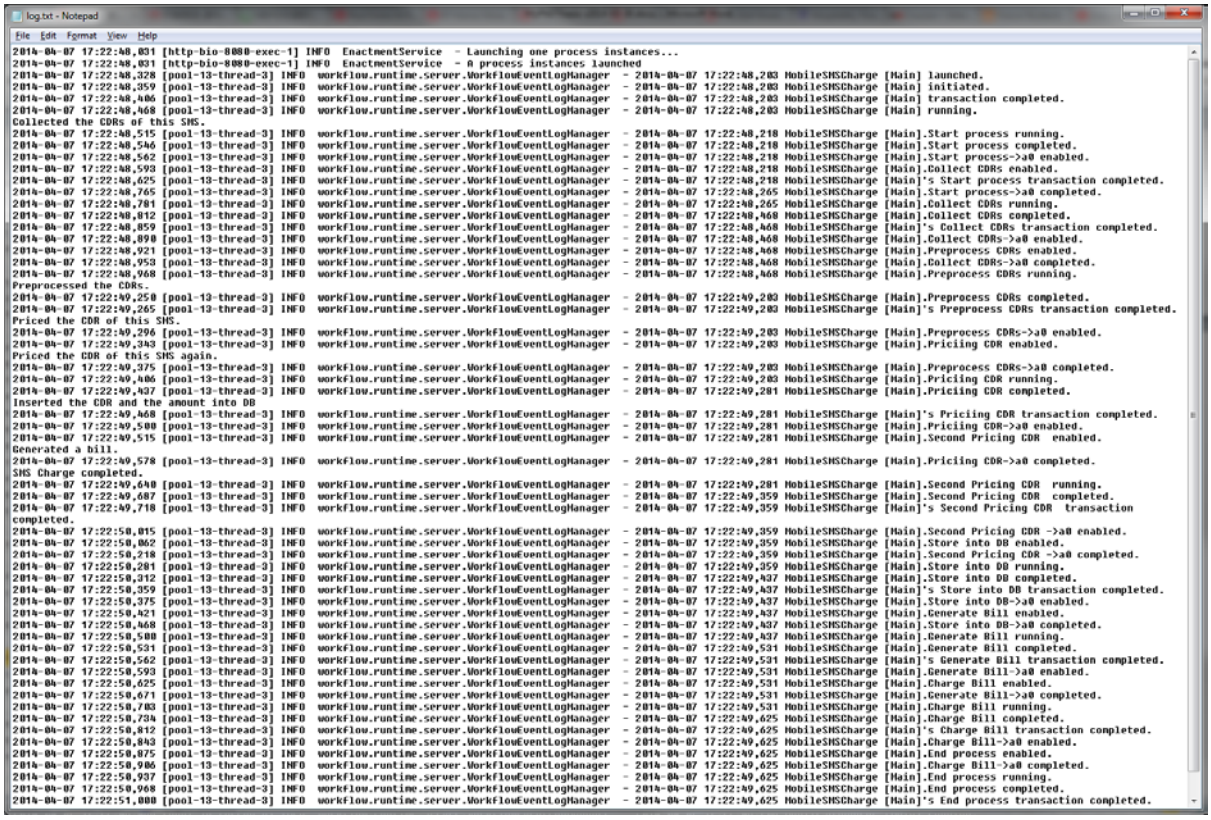

Figure 8.9 Logs of a workflow instance

After a workflow instance completed, the logs of the instance are represented in Fig. 8.9. The logs record the execution of a workflow instance in detail.

8.2.2 Experiment of non-functional components

In this section, we will represent an experiment for the non-functional aspects of SwinFlow-Cloud. The cloud side is deployed in Amazon AWS. We build a workflow client on client side emulator to simulate massive workflow requests sent to the cloud side.

8.2.2.1 Preparation

We addressed the runtime environment of SwinFlow-Cloud in Section 7.1.3. In this Section, we give a detailed preparation for the experiment, including both the client side and the cloud side.

For the client side, we design a client emulator based on Apache JMeter[54], which is open source Java application software that is designed to test functional behaviour and measure performance, to simulate massive workflow requests. The emulator is specified to send about 3,000 requests to the cloud side every five seconds on average.

For the cloud side, to reduce the communication delays between the services, we first deploy load balancing service, alarm service, and auto-scaling service onto one Amazon Linux EC2 instance in Amazon AWS, i.e., a virtual machine with Linux operation system, and configure the initial specifications as follows: for the composite index *I*, we consider the three typical resource utilisation rates to calculating the index: r_{cpu} (CPU resource utilisation), r_{proc} (Workflow engine thread pool utilisation) and r_{task} (Task transaction engine thread pool utilisation). We also deploy a workflow enactment service onto another Amazon Linux EC2 instance, denoted as workflow server. Then we measure the decaying time of a workflow server (i.e., t_{dc}) being about 130 seconds and the pending time (i.e., t_{pd}) of a workflow server being about 60 seconds. The time period for the alarm service is T_{al}. The time period for the load balancing service is T_{ld}. The initial quantity of available workflow server *Z* is 1.

8.2.2.2 Simulation experiments

According to the pre-configurations above, initially, there are *Z* workflow servers to handle the requests from the client side. Our experiments consist of two separate sets. The first set of experiment is to demonstrate the workflow instance throughputs of SwinFlow-Cloud on Amazon AWS with the pre-configurations of T_{ld} and T_{al} meeting the two principles described in Subsection 6.3.9, while the second set of experiment is to demonstrate the throughput with the pre-configurations not meeting the two principles.

For the first set of experiment, we set T_{ld} of the load balancing service as 40 seconds and T_{al} as 60 seconds. Thus, 60 (T_{al}) + 60 (t_{pd}) < 130 (T_{dc}) meets Principle 1 (if $T_{al} + t_{pd} < t_{dc}$, then $\bar{I}(t)$ is able to keep recoverable); while 40 (T_{ld}) < 60 (T_{al}) meets Principle 2 (when

[54] https://jmeter.apache.org/

the load balancing service needs to check $\bar{I}(t)$ before dispatching a request, if $T_{ld} < T_{al}$, then the load balancing service is able to monitor timely the state transitions of $\bar{I}(t)$). The experimental results are shown in Fig. 8.10. The experiment lasted for five hours. All workflow servers were kept recoverable throughout the experiment. After five hours, we terminated the client emulator. The cloud side automatically scaled in and released the workflow servers. *Note:* The Y-axis is the sum of input (the started workflow instances) and output (the completed workflow instances) records every ten minutes. The X-axis is time. Its interval is ten minutes. The line with dots demonstrates the changes of the input, while the line with crosses demonstrates the changes of the output.

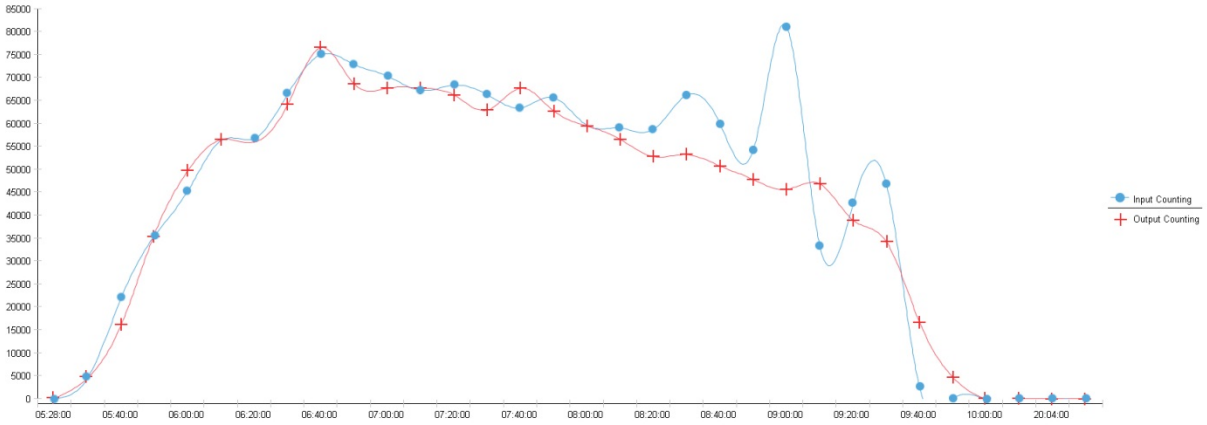

Figure 8.10 Throughputs of workflow servers with two principles

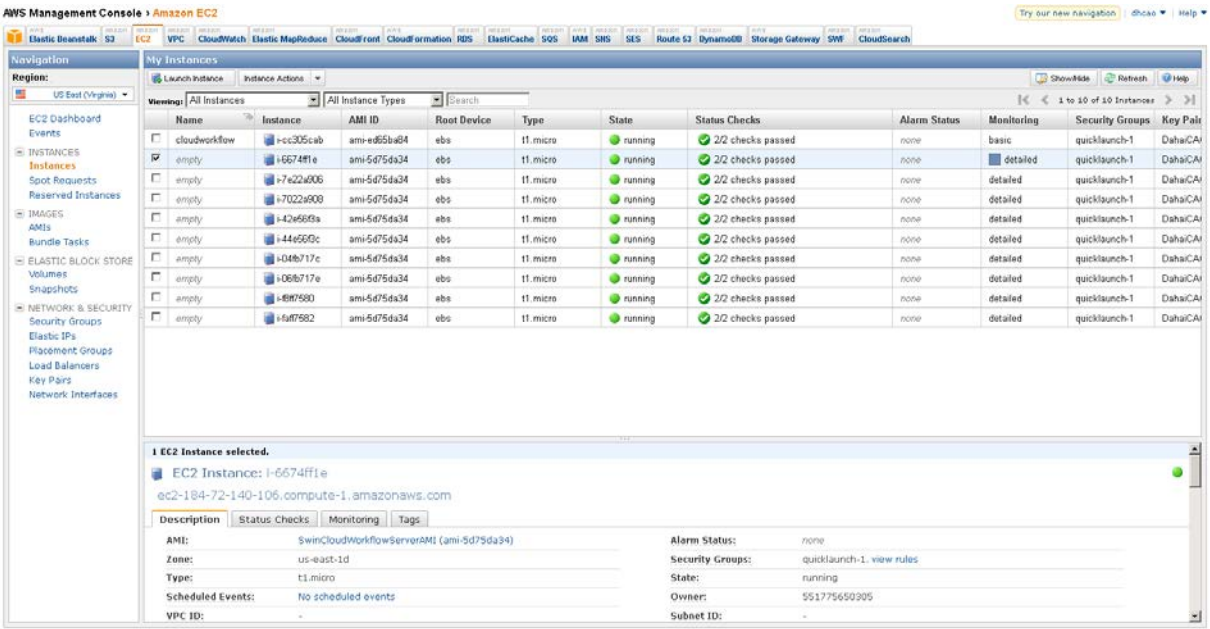

Figure 8.11 Workflow servers in the first set of experiment

Fig. 8.11 shows that SwinFlow-Cloud scaled out by ten workflow servers in total to handle the requests from the client emulator. The maximum input of these workflow servers is more than 81,000 workflow instances every ten minutes with an average of about 75,000, while the maximum output is about 80,000 workflow instances every ten minutes with an average of about 70,000. Therefore, the total throughput of the SwinFlow-Cloud is about 4,350,000 in five hours. The experimental results show that the coordination between the services is sustainable due to applying the two principles.

For the second set of experiment, we set T_{ld} as 120 seconds ($T_{ld} > t_{dc}$) and set T_{al} as 150 seconds ($T_{al} + t_{pd} > t_{dc}$), which do not obey the two principles. The experimental results are shown in Fig. 8.12. The figure shows that the throughput keeps increasing for the first one and half hours and reaches to the maximum point of over 14,000. It then decreases sharply and approaches to zero after one and half hours.

The reason for such a sharp decrease is due to $T_{ld} > t_{dc}$, which means that the moment when the load balancing service found out the workflow servers unrecoverable is later than the moment when the workflow servers became unrecoverable. When the massive number of requests is sent to the workflow servers on the cloud side, the servers run out quickly and become unrecoverable and unavailable before detected by the load balancing service. After the load balancing service has detected one or more workflow server instances that are unrecoverable and unavailable, it will stop dispatching new requests to them and dispatch the requests to the recoverable workflow servers. SwinFlow-Cloud will reduce the capacity due to the decreasing quantity of the recoverable and available servers and may be overloaded. On the other hand, $T_{al} + t_{pd} > t_{dc}$ results in that the alarm service cannot be aware of the status of the workflow servers timely and notify the auto-scaling service to create new workflow servers for balancing the incoming requests. Thus, it will result in that the more workflow servers are created and change onto the unrecoverable status. Fig. 8.12 demonstrates the analysis above. SwinFlow-Cloud has no capacity to accept more requests after the servers are unrecoverable, though it eventually created over 20 workflow server instances, as shown in Fig. 8.13. If the experiment is going on, more workflow servers will be created. According to our observation, the new workflow servers changed to the unrecoverable status. Therefore, the coordination between the services on cloud side will not be sustainable without the application of the two principles.

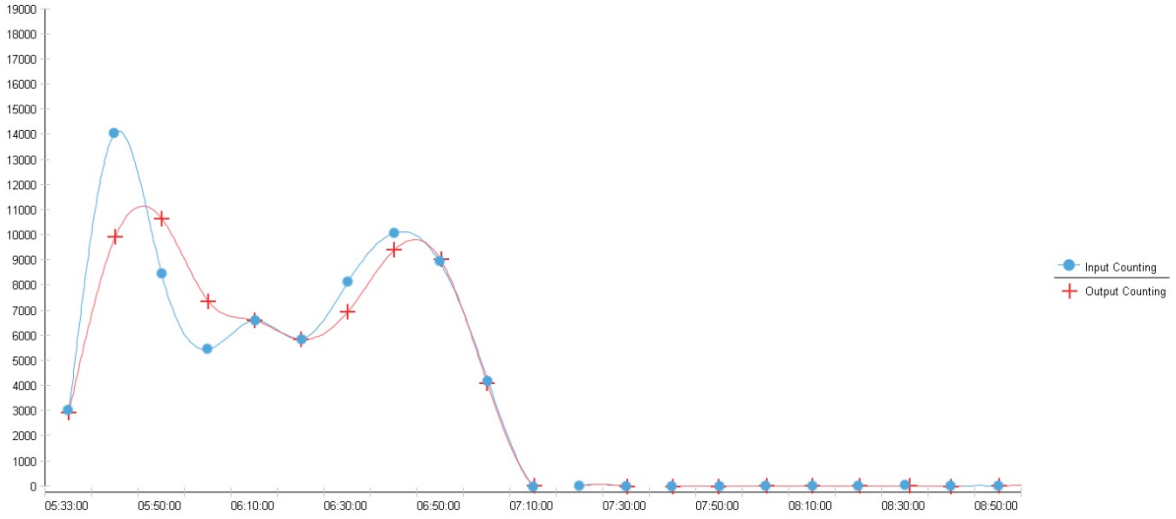

Figure 8.12 Throughputs of workflow servers without two principles

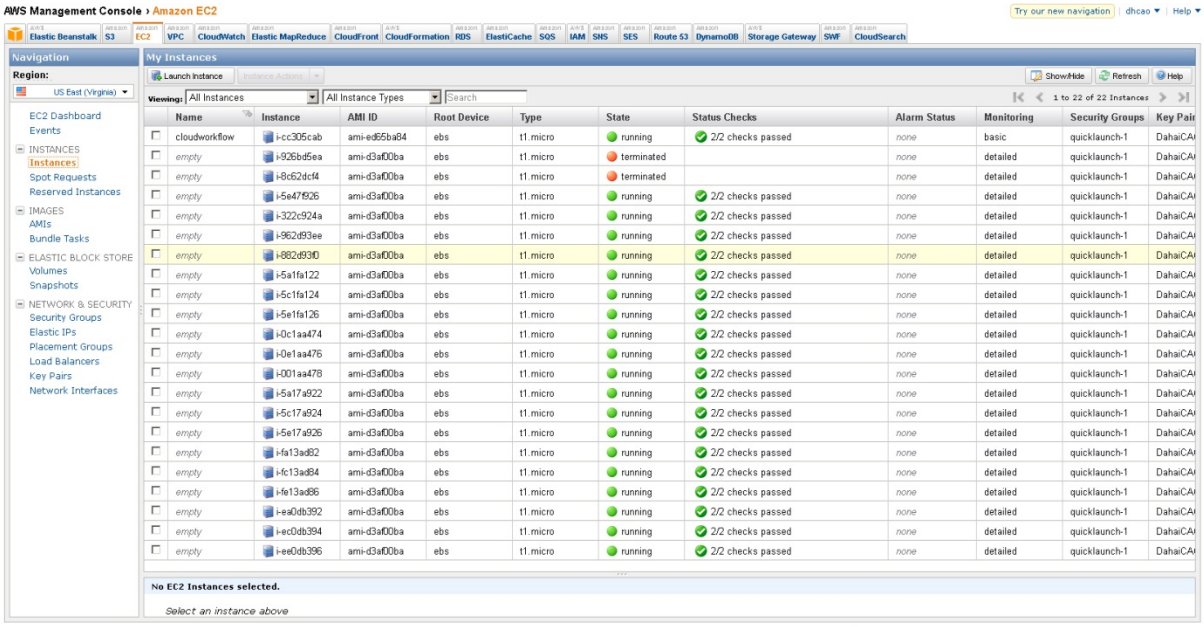

Figure 8.13 Workflow servers in the second set of experiment

8.3 Evaluation against requirements

The client-cloud SwinFlow-Cloud is a WfMS and supported the large-scale instance-intensive workflow in this experiment. The architecture meets requirement R1 (cloud workflow architecture).

In this experiment, SwinFlow-Cloud gained about 4,350,000 throughputs in five hours. The throughputs demonstrate that SwinFlow-Cloud has the capacity to support the workflow

applications with high throughputs. It meets requirement R2 (high throughputs) addressed in Section 3.2.

In this 5-hour experiment, the auto-scaling service scaled out by a total of additional ten workflow servers to meet the resource demand and scaled in for saving cost. It meets requirement R3 (Sustainable scalability) addressed in Section 3.2.

In the first set of the experiment, SwinFlow-Cloud keeps all workflow servers recoverable. The sustainable coordination guarantees the availability and reliability. Therefore, it meets requirements R4 (high availability) and R5 (high reliability) addressed in Section 3.2.

According to the Amazon pricing policies[55], the cost for our experiment which simulated the total throughput of 4,350,000 in 5 hours is $4.03 maximum. That is, the workflow system would only cost a maximum of $7,000 per year. Based on the performance in our experiment, we can extrapolate that the total cost is about $500,000 per year to handle 1.36 billion mobile SMS charge workflow instances on a daily basis which is marginal to the annual net profit of about $20 billion in China Mobile Limited. Hence, requirement R6 (cost-effectiveness) addressed in Section 3.2 is also met.

The experiment is our initial and basic experiment to demonstrate whether our cloud workflow is able to achieve the requirements we proposed in Section 3.2. We will further collect and research more experiment data in future.

8.4 Summary

In this chapter, we firstly gave an in-depth study and analysis to the motivating example in Section 3.2. Secondly, we deployed SwinFlow-Cloud on Amazon AWS and completed an experiment and further analysed the results. Finally, we evaluated the results of experiment and concluded the results met the requirements addressed in Section 3.2.

[55] http://aws.amazon.com/pricing/

Chapter 9
Conclusions and Future Work

This chapter addresses the conclusions and future work of this thesis. Section 9.1 summarises this thesis. Section 9.2 addresses the main contributions of this research. Section 9.3 gives an overview of our future research work.

9.1 Summary of this thesis

The objective of this thesis was to propose novel client-cloud architecture for cloud workflow to process large-scale instance-intensive workflows, and design and implement SwinFlow-Cloud to demonstrate the architecture. The thesis was organised as follows:

- Chapter 1 introduced workflow concepts which can be easily confused, clarified the relationship between the concepts, and further defined concepts and features of cloud workflow, and large-scale instance-intensive workflows. This chapter also described the key research issues and the structure of this thesis.

- Chapter 2 reviewed the WfMS architectures supporting large-scale workflow applications and the related literatures. In this chapter, we firstly surveyed the traditional workflow reference model proposed by WfMC in 1990s. Secondly, we gave an overview of the traditional WfMS architectures for large-scale workflows, including the centralised architectures and the decentralised architectures. Finally, we further discussed the start-of-the-art and state-of-the-practice cloud workflow research.

- Chapter 3 represented a motivating example of the representative large-scale instance-intensive workflow applications, refined the system requirements of designing cloud workflow and further analysed the research problems addressed in this thesis.

- Chapter 4 represented novel client-cloud architecture for cloud workflow. The client side consists of the Web-based or desktop-based applications including the functional workflow management tools and non-functional workflow accompaniment tools. The cloud side consists of the functional workflow enactment service and the non-functional workflow relevant services, including billing service, WfSMI service, load balancing service, alarm service, auto-scaling service, etc. We further discussed the elements which influence the sustainable coordination between the services.

- Chapter 5 addressed the functional design of SwinFlow-Cloud based on the client-cloud architecture. The functional design includes the build time and runtime functions. The former presented the process management, while the latter described the workflow instance, workflow enactment service which covers the workflow engine, the task transaction engine, and the navigation engine, and workflow administration and monitoring functionality.

- Chapter 6 addressed the non-functional design of SwinFlow-Cloud based on the client-cloud architecture. The non-functional design covers the organisation, version, and tool agent invocation management, the cloud workflow relevant service definition, administration and monitoring functionality, and the workflow relevant services. This chapter proposed two models: the alarm estimation model and the scaling estimation model. The former constructs a composite index to quantify the utilisation of various resources in workflow enactment services, uses the quadratic curve fitting methodology to monitor the tendency of the index, and further utilises the discrete-time Markov chain to predict future probability of the index. The latter estimates the overloaded resources, calculates the resource demands for balancing the remainder of workloads, and calculates the quantity of the workflow enactment services which needs to be scaled out or in. Finally, we discuss the principles for coordination sustainability.

- Chapter 7 represented the implementation of SwinFlow-Cloud prototype and the technology for developing the prototype. We firstly gave an overview of the

technologies developing SwinFlow-Cloud, discussed the motivations of choosing the technologies and surveyed the system development, and then introduced the development environment of SwinFlow-Cloud: Amazon AWS and its characteristics. Secondly, we represented the implementation of key fundamental functional and non-functional components and algorithms to support SwinFlow-Cloud. The functional components include process manager, organisation manager, tool agent manager, workflow engine, task transaction engine, and navigation engine. The non-functional components include the alarm service and the auto-scaling service.

- Chapter 8 represented the case study, experiments and evaluation. We deeply analysed the motivating example. The example is modelled in process manager and defined with multiple tool agents for invocation in the process. Then, we completed the two sets of experiment and demonstrated the principles which are able to facilitate the sustainability of the coordination between the services. Finally, we evaluated the results of the experiments and concluded that SwinFlow-Cloud based on the client-cloud architecture can meet all the requirements proposed in Section 3.2.

9.2 Contributions of this thesis

The significance of this research is that we have investigated comprehensively cloud workflow and further proposed novel cloud workflow architecture to support large-scale instance-intensive workflows. Because cloud computing is extending in broad areas, the architecture of the cloud workflow presented in this thesis can provide a reference for constructing complex applications in cloud.

In particular, the major contributions of this thesis are:

1. Novel client-cloud architecture for cloud workflow. This thesis utilises the cloud computing paradigm to innovate novel client-cloud model for architecting WfMS to support large-scale instance-intensive workflows. The client-cloud model offers more powerful scalability, availability, and reliability and provides a more cost-effective solution for WfMS.

2. The functional and non-functional components in the architecture. In the architecture, this thesis further inherits the functional component structure from the traditional

workflow reference model and proposes new non-functional components, workflow accompaniment tools and workflow relevant services, to enhance the traditional WfMS architecture.

3. The alarm estimation model. This estimation model can be sensitively aware of the tendency of the index and the status of cloud workflow for spikes of the workload of workflow emerging during processing large-scale instance-intensive workflows.

4. The scaling estimation model. This estimation model can quantitatively predict future resource demands for supporting automatic scaling up or down. The calculation for prediction promotes the cloud workflow to be a cost-effective workflow service provider.

5. Principles for coordination sustainability. The principles represent the constraints of the elements which influence the coordination between the services. The constraints are important in designing a cloud workflow based on the client-cloud architecture but often ignored by the designers. Thus, the principles improve the availability and reliability.

9.3 Future work

The current work in this thesis is that we have completed most of the fundamental designs of SwinFlow-Cloud including functional and non-functional aspects and deployed SwinFlow-Cloud in Amazon AWS. Based on the current work presented in this thesis, future work can be conducted from the following aspects:

1. For our system experiment, we will further conduct experiments to demonstrate and reveal more constraints we have discussed in Subsection 4.3.6 and Subsection 6.3.9. The constraints are critical to cloud workflow for guaranteeing the sustainability of the coordination between the services.

2. For system development, the current workflow client of SwinFlow-Cloud is a big standalone desktop-based plug-in application based on Eclipse RCP. It will be divided into multiple modular plug-ins and organised into a mainframe which is based on Eclipse RCP to offer more flexible extensibility.

3. In the client-cloud architecture, there are multiple elements which influence coordination sustainability between the non-functional services. Current work in this thesis has revealed some constraints among them, as addressed in the two principles. But there are still more constraints to be revealed in future practice. We will further our research to reveal other constraints.

4. Cloud workflow can rapidly accumulate massive data during processing instance-intensive workflow applications. Therefore, data management of cloud workflow, including data mining, data analysing, data security, data transferring and so on, should be further investigated in the future.

5. Current SwinFlow-Cloud is deployed in Amazon AWS for demonstration. In the future, we will abstract and capsulate the API level of SwinFlow-Cloud and migrate this system prototype onto other cloud infrastructures, such as Eucalyptus, OpenStack, Microsoft Azure[56], IBM SmartCloud, or Google Cloud[57], etc. We will compare the performances of which SwinFlow-Cloud running on these cloud platforms.

[56] https://www.windowsazure.com/en-us/
[57] https://cloud.google.com/

Bibliography

[1] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski and A. P. Barros, *Workflow Patterns*, Distributed and Parallel Databases, 14 (1): pp. 5-51, (2003)

[2] K. Aberer and M. Hauswirth, *Peer-to-Peer Information Systems: Concepts and Models, State-of-the-Art, and Future Systems*, in *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering(ESEC/FSE 2001)*, Vienna, Austria, pp. 326-327, (2001)

[3] G. Aceto, A. Botta, W. d. Donato and A. Pescapè, *Cloud Monitoring: Definitions, Issues and Future Directions*, in *Proceedings of the 1st IEEE International Conference on Cloud Networking (CLOUDNET)*, Paris, France, pp. 63-67, (2012)

[4] G. Aceto, A. Botta, W. d. Donato and A. Pescapè, *Cloud Monitoring: A Survey*, Computer Networks, 57 (9): pp. 2093-2115, (2013)

[5] Y. W. Ahn, A. M. K. Cheng, J. Baek, M. Jo and H.-H. Chen, *An Auto-Scaling Mechanism for Virtual Resources to Support Mobile, Pervasive, Real-time Healthcare Applications in Cloud Computing*, Network, IEEE, 27 (5): pp. 62-68, (2013)

[6] A. Aldeeb, K. Crockett and M. J. Stanton, *Multi-Agent Based Peer-to-Peer Workflow Management System*, in *Proceedings of the Scope of the Symposium*, Manchester, UK, pp. 1-8, (2008)

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica and M. Zaharia, Above the Clouds: A Berkeley View of Cloud Computing, Berkeley, CA, U.S.A., pp. 1-25, (2009)

[8] I. Astrova, A. Koschel and M. Schaaf, *Automatic Scaling of Complex-Event Processing Applications in Eucalyptus*, in *Proceddings of the 15th IEEE International Conference on Computational Science and Engineering (CSE)*, Nicosia, Cyprus, pp. 22-29, (2012)

[9] A. Bashar, *Autonomic Scaling of Cloud Computing Resources Using BN-based Prediction Models*, in *Proceedings of the 2nd IEEE International Conference on Cloud Networking (CloudNet)*, San Francisco, USA, pp. 200-204, (2013)

[10] M. Beltr´an and A. Guzm´an, *An Automatic Machine Scaling Solution for Cloud Systems*, in *Proceedings of the 19th IEEE International Conference on High Performance Computing (HiPC)*, Pune, India, pp. 1-10, (2012)

[11] G. Boss, P. Malladi, D. Quan, L. Legregni and H. Hall, *Cloud Computing*, IBM: pp. 17, (2007)

[12] J. M. Bradshaw, *An Introduction to Software Agents*, in *Software Agents*, MIT Press, Cambridge, MA, USA, pp. 3-46, (1997)

[13] N. M. Calcavecchia, B. A. Caprarescu, E. D. Nitto, D. J. Dubois and D. Petcu, *DEPAS: A Decentralized Probabilistic Algorithm for Auto-Scaling*, Computing, 94 (8-10): pp. 701-730, (2012)

[14] D. Cao, X. Liu and Y. Yang, *Novel Client-Cloud Architecture for Scalable Instance-Intensive Workflow Systems*, in *Proceedings of the 14th International Conference on Web Information Systems Engineering (WISE)*, Nanjing, China, pp. 270-284, (2013)

[15] E. Caron, F. e. e. Desprez and A. Muresan, *Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching*, in *the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, U.S.A., pp. 456 - 463, (2010)

[16] M. a. B. d. Carvalho, R. P. Esteves, G. d. C. Rodrigues, L. Z. Granville and L. M. R. Tarouco, *A Cloud Monitoring Framework for Self-Configured Monitoring Slices Based on Multiple Tools*, in *Proceedings of the 9th IEEE International Conference on Network and Service Management (CNSM)*, Zurich, Switzerland, pp. 180-184, (2013)

[17] S. Chaisiri, B.-S. Lee and D. Niyato, *Optimization of Resource Provisioning Cost in Cloud Computing*, Services Computing, IEEE Transactions on, 5 (2): pp. 164-177, (2012)

[18] S. Chaisiri, R. Kaewpuang, B.-S. Lee and D. Niyato, *Cost Minimization for Provisioning Virtual Servers in Amazon Elastic Compute Cloud*, in *Proceedings of the 19th IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, Singapore pp. 85-95, (2011)

[19] C. Chapman, W. Emmerich, F. G. Márquez, S. Clayman and A. Galis, *Software Architecture Definition for On-Demand Cloud Provisioning*, Cluster Computing, 15 (2): pp. 79-100, (2012)

[20] T. C. Chieu, A. Mohindra, A. A. Karve and A. Segal, *Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment*, in *2009 IEEE International Conference on e-Business Engineering (ICEBE)*, IEEE. pp. 281 - 286 (2009)

[21] W. M. Coalition, Workflow Process Definition Interface -- XML Process Definition Language, pp. 87, (2002)

[22] S. Costache, N. Parlavantzas, C. Morin and S. Kortas, *Themis: Economy-based Automatic Resource Scaling for Cloud Systems*, in *Proceedings of the 9th IEEE International Conference on High Performance Computing and Communication & the 14th IEEE International Conference on Embedded Software and Systems (HPCC-ICESS)*, Liverpool, UK, pp. 367-374, (2012)

[23] R. Cushing, S. Koulouzis, A. S. Z. Belloum and M. Bubak, *Prediction-Based Auto-Scaling of Scientific Workflows*, in *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, ACM, Lisbon, Portugal. pp. 1-6, (2011)

[24] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, Communications of the ACM - 50th Anniversary Issue: 1958 - 2008, 51 (1): pp. 107-113, (2008)

[25] E. Deelman, D. Gannonb, M. Shieldsc and I. Taylor, *Workflows and e-Science: An Overview of Workflow System Features and Capabilities*, Future Generation Computer Systems, 25 (5): pp. 528-540, (2009)

[26] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi and M. Livny, *Pegasus: Mapping Scientific Workflows onto the Grid*, in *Proceedings of the 2nd EuropeanAcrossGrids Conference (AxGrids 2004)*, Nicosia, Cyprus, pp. 11-20, (2004)

[27] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz, *Pegasus: A framework for mapping complex scientific workflows onto distributed systems*, Scientific Programming, 13 (3): pp. 219-237, (2005)

[28] B. Dougherty, J. White and D. C. Schmidt, *Model-driven auto-scaling of green cloud computing infrastructure*, Future Generation Computer Systems, 28 (2): pp. 371-378, (2012)

[29] S. Dutta, S. Gera, A. Verma and B. Viswanathan, *SmartScale: Automatic Application Scaling in Enterprise Clouds*, in *Proceddings of the 5th IEEE International Conference onCloud Computing (CLOUD)*, Honolulu, USA, pp. 221-228, (2012)

[30] L. Ehrler, M. Fleurke, M. Purvis and B. T. R. Savarimuthu, *Agent-based Workflow Management Systems (WfMSs)*, Information Systems and E-Business Management, 4 (1): pp. 5-23, (2006)

[31] G. J. Fakas and B. Karakostas, *A Peer to Peer (P2P) Architecture for Dynamic Workflow Management*, Information and Software Technology, 46 (6): pp. 423-431, (2004)

[32] F. L. Ferraris, D. Franceschelli, M. P. Gioiosa, D. Lucia, D. Ardagna, E. D. Nitto and T. Sharif, *Evaluating the Auto Scaling Performance of Flexiscale and Amazon EC2 Clouds*, in *Proceedings of the 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, Rumania, pp. 423-429, (2012)

[33] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*, Morgan Kaufmann, USA. (2003)

[34] I. Foster, Y. Zhao, I. Raicu and S. Lu, *Cloud Computing and Grid Computing 360-Degree Compared*, in *Proceedings of Grid Computing Environments Workshop, 2008 (GCE 2008)*, Austin, Texas, U.S.A., pp. 1-10, (2008)

[35] D. Franz, J. Tao, H. Marten and A. Streit, *A Workflow Engine for Computing Clouds*, in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING)*, Rome, Italy, pp. 1-6, (2011)

[36] G. Galante and L. C. E. d. Bona, *A Survey on Cloud Computing Elasticity*, in *Proceedings of the 5th IEEE International Conference on Utility and Cloud Computing (UCC)*, Chicago, USA, pp. 263-270, (2012)

[37] S. V. Gogouvitis, V. Alexandrou, N. Mavrogeorgi, S. Koutsoutos, D. Kyriazis and T. Varvarigou, *A Monitoring Mechanism for Storage Clouds*, in *Proceedings of the 2nd IEEE International Conference on Cloud and Green Computing (CGC)*, Xiangtan, China, pp. 153-159, (2012)

[38] Z. Gong, X. Gu and J. Wilkes, *PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems*, in *2010 International Conference on Network and Service Management (CNSM)*, Niagara Falls, USA, pp. 9 - 16, (2010)

[39] Z. Gong, P. Ramaswamy, X. Gu and X. Ma, *SigLM: Signature-driven load management for cloud computing infrastructures*, in *Proceedings of the 17th International Workshop on Quality of Service (IWQoS)*, Charleston, USA, pp. 1-9, (2009)

[40] T. J. Hacker and K. Mahadik, *Flexible resource allocation for reliable virtual cluster computing systems*, in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, Washington, USA. pp. 1-12, (2011)

[41] R. Han, L. Guo, Y. Guo and S. He, *A Deployment Platform for Dynamically Scaling Applications in the Cloud*, in *the 3rd IEEE International Conference on Coud Computing Technology and Science (CloudCom)*, Athens, Greece, pp. 506 - 510 (2011)

[42] R. Han, L. Guo, M. M. Ghanem and Y. Guo, *Lightweight Resource Scaling for Cloud Applications*, in *the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Ottawa, Canada, pp. 644 - 651, (2012)

[43] Y. He, X. Wang, Y. Chen, Z. Du, W. Huang and X. Chai, *A Simulation Cloud Monitoring Framework and Its Evaluation Model*, Simulation Modelling Practice and Theory, 38 (0): pp. 20-37, (2013)

[44] D. Hollingsworth, Workflow Management Coalition: The Workflow Reference Model, Winchester, Hampshire, UK, pp. 1-55, (1995)

[45] C.-J. Huang, A. J. C. Trappey and Y.-H. Yao, *Developing an Agent-based Workflow Management System for Collaborative Product Design*, Industrial Management & Data Systems, 106 (5): pp. 680-699, (2006)

[46] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li and T. Oinn, *Taverna: a tool for building and running workflows of services*, Nucleic Acids Research, 34 (suppl 2): pp. W729-W732, (2006)

[47] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer and D. H. J. Epema, *Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing*, IEEE Transactions on Parallel and Distributed Systems, 22 (6): pp. 931-945, (2011)

[48] J. Jiang, J. Lu, G. Zhang and G. Long, *Optimal Cloud Resource Auto-Scaling for Web Applications*, in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 58-65, (2013)

[49] A. Kamel, A. Al-Fuqaha, D. Kountanis and I. Khalil, *Towards a Client-side QoS Monitoring and Assessment Using Generalized Pareto Distribution in a Cloud-based Environment*, in *Proceedings of the IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, Shanghai, China, pp. 123-128, (2013)

[50] H. Kang, J.-i. Koh, Y. Kim and J. Hahm, *A SLA Driven VM Auto-Scaling Method in Hybrid Cloud Environment*, in *Proceddings of the 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Hiroshima, Japan, pp. 1-6, (2013)

[51] A. Khan, X. Yan, S. Tao and N. Anerousis, *Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach*, in *Proceddings of the IEEE Network Operations and Management Symposium (NOMS)*, Maui, USA, pp. 1287-1294, (2012)

[52] C.-C. Lin, J.-J. Wu, J.-A. Lin, L.-C. Song and P. Liu, *Automatic Resource Scaling Based on Application Service Requirements*, in *Proceeedings of the 5th IEEE International Conference on Cloud Computing (CLOUD)*, Honolulu, USA, pp. 941-942, (2012)

[53] C.-C. Lin, J.-J. Wu, P. Liu, J.-A. Lin and L.-C. Song, *Automatic Resource Scaling for Web Applications in the Cloud*, in *Grid and Pervasive Computing*, J. Park, et al., Springer-Verlag Berlin Heidelberg, pp. 81-90, (2013)

[54] X. Liu, Y. Yang, D. Cao and D. Yuan, *Selecting Checkpoints Along the Time Line: A Novel Temporal Checkpoint Selection Strategy for Monitoring a Batch of Parallel Business Processes*, in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, San Francisco, USA, pp. 1281-1284, (2013)

[55] X. Liu, D. Yuan, G. Zhang, J. Chen and Y. Yang, *SwinDeW-C: A Peer-to-Peer based Cloud Workflow System*, in *Handbook of Cloud Computing*, B. Furht and A. Escalante, Springer US, pp. 309-332, (2010)

[56] X. Liu, Y. Yang, D. Cao, D. Yuan and J. Chen, *Managing Large Numbers of Business Processes with Cloud Workflow Systems*, in *Proceedings of the 10th Australasian Symposium on Parallel and Distributed Computing*, Australian Computer Society, Inc.: Melbourne, Australia, pp. 33-42, (2012)

[57] X. Liu, Y. Yang, D. Yuan, G. Zhang, W. Li and D. Cao, *A Generic QoS Framework for Cloud Workflow Systems*, in *Proceedings of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, Sydney, Australia, pp. 713-720, (2011)

[58] X. Liu, D. Yuan, G. Zhang, W. Li, D. Cao, Q. He, J. Chen and Y. Yang, *The Design of Cloud Workflow Systems*, Springer, (2012)

[59] M. Mao and M. Humphrey, *Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows*, in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Seatle, USA, pp. 1 - 12, (2011)

[60] M. Mao and M. Humphrey, *Scaling and Scheduling to Maximize Application Performance within Budget Constraints in Cloud Workflows*, in *Proceedings of the 27th IEEE International Symposium on Parallel & Distributed Processing*, Boston, USA, pp. 67-78, (2013)

[61] M. Mao, J. Li and M. Humphrey, *Cloud Auto-scaling with Deadline and Budget Constraints*, in *the 11th IEEE/ACM International Conference on Grid Computing (GRID)*, Brussels, Belgium, pp. 41 - 48, (2010)

[62] G. A. McGilvary, J. Rius, I. n. Goiri, F. Solsona, A. Barker and M. Atkinson, *C2MS: Dynamic Monitoring and Management of Cloud Infrastructures*, in *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Bristol, UK, pp. 290-297, (2013)

[63] D. Moldovan, G. Copil, H.-L. Truong and S. Dustdar, *MELA: Monitoring and Analyzing Elasticity of Cloud Services*, in *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Bristol, UK, pp. 80-87, (2013)

[64] N. S. More and S. R. Hiray, *Load balancing and resource monitoring in cloud*, in *Proceedings of the CUBE International Information Technology Conference*, ACM: Pune, India. pp. 552-556, (2012)

[65] D. Niu, H. Xu, B. Li and S. Zhao, *Quality-Assured Cloud Bandwidth Auto-Scaling for Video-On-Demand Applications*, in *Proceedings of the IEEE INFOCOM*, Orlando, USA, pp. 460-468, (2012)

[66] H. S. Nwana and D. T. Ndumu, *An Introduction to Agent Technology*, in *Software Agents and Soft Computing Towards Enhancing Machine Intelligence*, H. Nwana and N. Azarmi, Springer Berlin / Heidelberg, Berlin/Heidelberg, pp. 1-26, (1997)

[67] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V. B. N. Rao, V. Sankarasubramanian, S. Seth, C. Tian, T. ZiCornell and X. Wang, *Nova: continuous Pig/Hadoop workflows*, in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, ACM: Athens, Greece. pp. 1081-1090, (2011)

[68] J. Park, H. Yu, K. Chung and E. Lee, *Markov Chain Based Monitoring Service for Fault Tolerance in Mobile Cloud Computing*, in *Proceedings of the IEEE Workshops*

of International Conference on Advanced Information Networking and Applications (WAINA), Biopolis, Singapore, pp. 520-525, (2011)

[69] N. Roy, A. Dubey and A. Gokhale, *Efficient Autoscaling in the Cloud using Predictive Models forWorkload Forecasting*, in *International Conference on Cloud Computing*, Washington, USA, pp. 500 - 507, (2011)

[70] N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst and N. Mulyar, Workflow ControlFlow Patterns: A Revised View, pp. 1-134, (2006)

[71] P. Saripalli, G. Kiran, R. S. R, H. Narware and N. Bindal, *Load Prediction and Hot Spot Detection Models for Autonomic Cloud Computing*, in *Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC)*, Melbourne, Australia, pp. 397-402, (2011)

[72] D. M. M. Schunselaar, T. F. van der Avoort, H. M. W. Verbeek and W. M. P. van der Aalst, *YAWL in the Cloud*, in *Proceedings of the 1st YAWL Symposium*, Sankt Augustin, Germany, pp. 41-48, (2013)

[73] J. Shao, H. Wei, Q. Wang and H. Mei, *A Runtime Model Based Monitoring Approach for Cloud*, in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD)*, Miami, USA, pp. 313-320, (2010)

[74] J. Shen, Y. Yang and Q. H. Huy, *SwinDeW-B: A P2P based Composite Service Execution System with BPEL*, in *Proceeding of the International Workshop on Dynamic Web Processes (DWP)*, Amsterdam, Netherlands, pp. 73-84, (2005)

[75] J. Shen, Y. Yang and J. Yan, *Adapting P2P based Decentralised Workflow System SwinDeW-S with Web Service Profile Support*, in *Proceedings of the 9th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, Coventry, UK, pp. 535-540, (2005)

[76] Z. Shen, S. Subbiah, X. Gu and J. Wilkes, *CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems*, in *the 2nd ACM Symposium on Cloud Computing (SoCC)*, Cascais, Portugal, pp. 1-14, (2011)

[77] W. Stallings, *Operating Systems: Internals and Design Principles*, 7 ed, New Jersey, U.S.A. Prentice Hall. 768. (2012)

[78] A. D. Stefano, G. Morana and D. Zito, *Scalable and Configurable Monitoring System for Cloud Environments*, in *Proceedings of the 22nd IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Hammamet, Tunisia, pp. 134-139, (2013)

[79] J. Styles and H. Hoos, *Ordered Racing Protocols for Automatically Configuring Algorithms for Scaling Performance*, in *Proceedings of the 15th ACM Annual Conference on Genetic and Evolutionary Computation*, Amsterdam, Netherlands, pp. 551-558, (2013)

[80] L. M. Vaquero, L. Rodero-Merino and R. Buyya, *Dynamically Scaling Applications in the Cloud*, ACM SIGCOMM Computer Communication Review, 41 (1): pp. 45-52, (2011)

[81] S. Venugopal, H. Li and P. Ray, *Auto-Scaling Emergency Call Centres Using Cloud Resources to Handle Disasters*, in *Proceddings of the 19th IEEE International Workshop on Quality of Service (IWQoS)*, San Jose, USA, pp. 1-9, (2011)

[82] C. Wang, J. Chen, B. B. Zhou and A. Y. Zomaya, *Just Satisfactory Resource Provisioning for Parallel Applications in the Cloud*, in *Proceddings of the 8th World Congress on Services (SERVICES)*, Honolulu, USA, pp. 285-292, (2012)

[83] W. Wang, H. Chen and X. Chen, *An Availability-Aware Approach to Resource Placement of Dynamic Scaling in Clouds*, in *Proceddings of the 5th IEEE International Conference on Cloud Computing (CLOUD)*, Honolulu, USA, pp. 930-931, (2012)

[84] WfMC, Workflow Management Coalition: Terminology & Glossary, Winchester, Hampshire, UK., pp. 1-65, (1999)

[85] J. Yan, Y. Yang and G. K. Raikundalia, *A Decentralised Architecture for Workflow Support*, in *Proceedings of the 7th International Symposium on Future Software Technology (ISFST 2002)*, Wuhan, China, pp. 23-25, (2002)

[86] J. Yan, Y. Yang and G. K. Raikundalia, *SwinDeW—A p2p-Based Decentralized Workflow Management System*, IEEE Transactions on Systems, Man, and Cybernrtics—Part A: Systems and Humans, 36 (5): pp. 922-935, (2006)

[87] J. Yang, C. Liu, Y. Shang, Z. Mao and J. Chen, *Workload Predicting-Based Automatic Scaling in Service Clouds*, in *Proceedings of the 6th IEEE International Conference on Cloud Computing*, Santa Clara, USA, pp. 810-815, (2013)

[88] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu and J. Chen, *A Cost-Aware Auto-Scaling Approach Using the Workload Prediction in Service Clouds*, Information Systems Frontiers, 16 (1): pp. 7-18, (2014)

[89] Y. Yang, K. Liu, J. Chen, L. J. and H. Jin, *Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G (e-Science07)*, in *Proceedings of the 3rd IEEE*

International Conference on e-Science and Grid Computing, Bangalore, India, pp. 51-58, (2007)

[90] L. Yazdanov and C. Fetzer, *VScaler: Autonomic Virtual Machine Scaling*, in *proceedings of the 6th IEEE International Conference on Cloud Computing (CLOUD)*, pp. 212-219, (2013)

[91] U. Yildiz, A. Guabtniy and A. H. H. Nguz, Business versus Scientific Workflow: A Comparative Study, pp. 20, (2009)

[92] J. Yu and R. Buyya, *A Taxonomy of Scientific Workflow Systems for Grid Computing*, ACM SIGMOD Record, 34 (3): pp. 44-49, (2005)

[93] D. Yuan, X. Liu, L. Cui, T. Zhang, W. Li, D. Cao and Y. Yang, *An Algorithm for Cost-Effectively Storing Scientific Datasets with Multiple Service Providers in the Cloud*, in *Proceedings of the 9th IEEE International Conference on eScience (eScience)*, Beijing, China, pp. 285-292, (2013)

[94] C. Zhang and H. D. Sterck, *CloudWF: A Computational Workflow System for Clouds Based on Hadoop*, in *Proceddings of The 1st International Conference on Cloud Computing (CloudCom 2009)*, Beijing, P.R. CHINA, pp. 393-404, (2009)

[95] Q. Zhang, Q. Zhu and R. Boutaba, *Dynamic Resource Allocation for Spot Markets in Cloud Computing Environments*, in *Proceddings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC)*, Melbourne, Australia, pp. 178-185, (2011)

[96] Y. Zhao, Y. Zhang, W. Tian, R. Xue and C. Lin, *Designing and Deploying a Scientific Computing Cloud Platform*, in *Proceedings in the 13th ACM/IEEE International Conference on Grid Computing (GRID)* pp. 104-113, (2012)

Appendix

Notation Index

Notation name	Definition	Page defined
c	capacity of a resource	82
c_{cpu}	total CPU time consumption in a workflow enactment service	82
c_{io}	maximum read/write bytes of an IO device	83
c_{mem}	maximum memory size in a workflow enactment service	83
c_{net_i}	maximum incoming bytes of the network device	83
c_{net_o}	maximum outgoing bytes of the network device	83
cp_i	total capacity of the i resource to keep the su_i in $[th_2, th_3)$	100
c_{proc}	length of the waiting queue of the workflow engine thread pool	83

$c_{session}$	maximum session number a workflow enactment service can offer	84
c_{task}	length of the waiting queue of the task transaction engine thread pool	84
$I_j(t)$	composite index to assess the overall status of j^{th} workflow enactment service at t, $I_j(t) \in [0, 1]$	91
$\overline{I}(t)$	mean of $I_j(t)$, i=1..m	91
$\overline{I'}(t)$	first derivative of $\overline{I}(t)$	92
k_o	slope which indicates the maximum speed at which $r'_{ij}(t)$ increases	99
m	number of workflow enactment services in SwinFlow-Cloud	82
n	number of the considered resources in each workflow enactment service	82
$\boldsymbol{P}_t$	transition matrix of the discrete-time Markov chain	94
$r(t)$	rate of $u(t)$ and c, with a range of values from 0 to 1, i.e., $r \in [0,1]$	82
$r_{cpu}(t)$	utilisation ratio of CPU in a workflow enactment service at t	82

$r_i\,(t)$	for n resources, the utilisation ratio of i^{th} resource, $i \in$ {*cpu*, *mem*, *net_i*, *net_o*, *io*, *proc*, *task*, *pool*, *session*, ⋯} or i=1..n, $r_i(t) \in [0,1]$	84
$r_{ij}(t)$	for m workflow enactment services, the utilisation ratio of i^{th} resource in j^{th} service, i=1..n, j=1..m, $r_{ij}(t) \in [0, 1]$	84
$r'_{ij}(t)$	$r_{ij}(t)$ with $\Delta w_{ij}(t) > 0$	99
$[r'_{ij}(t)]'$	first derivative of $r'_{ij}(t)$	99
$r_{io}(t)$	utilisation ratio of $u_{io}(t)$ to c_{io}	83
$r_{mem}(t)$	memory utilisation ratio of $u_{mem}(t)$ to c_{mem}	83
$r_{net\_i}(t)$	utilisation ratio of $u_{net\_i}(t)$ to c_{net_i}	83
$r_{net\_o}(t)$	utilisation ratio of $u_{net\_o}(t)$ to c_{net_o}	83
$r_{pool}(t)$	utilisation ratio of $u_{pool}(t)$ to c_{pool}	84
$r_{proc}(t)$	utilisation ratio of $u_{proc}(t)$ to c_{proc}	83
$r_{session}(t)$	utilisation ratio of $u_{session}(t)$ to $c_{session}$	84
$r_{task}(t)$	utilisation ratio of $u_{task}(t)$ to c_{task}	84
S	discrete state of $\bar{I}(t)$, S = {*IDLE*, *UNDL*, *NLLD*, *BUSY*, *EXBY*, *OVRL*}	93

SCL_d	scaled-in number	102
SCL_u	scaled-out number	101
su_i	total utilisation of the i resource in l' services	100
T	time period, $T = \{0,1,...\}$	82
t	any moment in T, $t \in T$	82
TH	threshold set, $TH = \{th_1,th_2,..,th_5\}$	93
$u(t)$	utilisation of a resource	82
$u_{cpu}(t)$	summary of the CPU time occupations of all workflow engines and task transaction engines at t	82
$u_i(t)$	for n resources, the utilisation of i^{th} resource, $i \in \{cpu, mem, net\_i, net\_o, io, proc, task, pool, session, \cdots\}$ or $i=1..n$	84
$u_{ij}(t)$	for m workflow enactment services, the utilisation of i^{th} resource in j^{th} workflow enactment service, $i=1..n, j=1..m$	84
$u_{io}(t)$	write/read bytes of an IO device at t	83
$u_{mem}(t)$	summary of the occupied memory size of all workflow engines and task transaction engines at t	83

$u_{net\_i}(t)$	summary of the incoming bytes of all the engines at t	83
$u_{net\_o}(t)$	summary of the outgoing bytes of all the engines at t	83
$u_{pool}(t)$	number of the waiting objects which needs to use the connections in the pool at t	84
$u_{proc}(t)$	number of the waiting workflow instances at t	83
$u_{session}(t)$	session number of a workflow enactment service at t	84
$u_{task}(t)$	number of the waiting task instances at t	84
w_0	initial $w_{ij}(t)$	90
$w_{ij}(t)$	coefficient of $r_{ij}(t)$ to represent the impact on $r_{ij}(t)$ at t	90
$\Delta\overline{w}_i(t)$	mean of $\Delta w_i(t)$, i=1..m	91
$\Delta w_{ij}(t)$	offset or changes of $w_{ij}(t)$ with $r_{ij}(t)$, $\Delta w_{ij}(t) \in [0, 1]$	90
Z	minimum number of workflow enactment services	103