

Georgia State University

ScholarWorks @ Georgia State University

Computer Science Dissertations

Department of Computer Science

Summer 7-28-2012

Simulation Software as a Service and Service-Oriented Simulation Experiment

Song Guo

Computer Sciences

Follow this and additional works at: https://scholarworks.gsu.edu/cs_diss

Recommended Citation

Guo, Song, "Simulation Software as a Service and Service-Oriented Simulation Experiment." Dissertation, Georgia State University, 2012.

doi: <https://doi.org/10.57709/3140905>

This Dissertation is brought to you for free and open access by the Department of Computer Science at ScholarWorks @ Georgia State University. It has been accepted for inclusion in Computer Science Dissertations by an authorized administrator of ScholarWorks @ Georgia State University. For more information, please contact scholarworks@gsu.edu.

SIMULATION SOFTWARE AS A SERVICE AND SERVICE-ORIENTED SIMULATION EXPERIMENT

by

SONG GUO

Under the Direction of Xiaolin Hu

ABSTRACT

Simulation software is being increasingly used in various domains for system analysis and/or behavior prediction. Traditionally, researchers and field experts need to have access to the computers that host the simulation software to do simulation experiments. With recent advances in cloud computing and Software as a Service (SaaS), a new paradigm is emerging where simulation software is used as services that are composed with others and dynamically influence each other for service-oriented simulation experiment on the Internet.

The new service-oriented paradigm brings new research challenges in composing multiple simulation services in a meaningful and correct way for simulation experiments. To systematically support simulation software as a service (SimSaaS) and service-oriented simulation experiment, we propose a layered framework that includes five layers: an infrastructure layer, a simulation execution engine layer, a simulation service layer, a simulation experiment layer and finally a graphical user interface layer. Within this layered framework, we provide a specification for both simulation experiment and the involved individual simulation services. Such a formal specification is useful in order to support systematic compositions of simulation services as well as automatic deployment of composed services for carrying out

simulation experiments. Built on this specification, we identify the issue of mismatch of time granularity and event granularity in composing simulation services at the pragmatic level, and develop four types of granularity handling agents to be associated with the couplings between services. The ultimate goal is to achieve standard and automated approaches for simulation service composition in the emerging service-oriented computing environment. Finally, to achieve more efficient service-oriented simulation, we develop a profile-based partitioning method that exploits a system's dynamic behavior and uses it as a profile to guide the spatial partitioning for more efficient parallel simulation. We develop the work in this dissertation within the application context of wildfire spread simulation, and demonstrate the effectiveness of our work based on this application.

INDEX WORDS: Web service, SOA, Workflow, Model composability and interoperability, DEVS, Modeling and simulation, Wildfire simulation

SIMULATION SOFTWARE AS A SERVICE AND SERVICE-ORIENTED SIMULATION EXPERIMENT

by

SONG GUO

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2012

Copyright by

Song Guo

2012

SIMULATION SOFTWARE AS A SERVICE AND SERVICE-ORIENTED SIMULATION EXPERIMENT

by

SONG GUO

Committee Chair: Xiaolin Hu

Committee: Yi Pan

Saeid Belkasim

Yichuan Zhao

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

2012

ACKNOWLEDGEMENTS

Firstly, I want to thank my advisor, Dr. Xiaolin Hu, for his generous advice and support for my Ph.D. study at Georgia State University. His board and deep insight in interdisciplinary area inspired and guided me in my research. I really appreciate what I learned from this great professor: how to perceive, think about, and analyze the physical world, which I will benefit from not only for my future research but also for the rest of my life.

Besides my advisor, I thank my committee members, Dr. Yi Pan, Dr. Raj Sunderraman, Dr. Saeid Belkasim, and Dr. Yichuan Zhao for their valuable suggestions in improving the quality of this work.

I would also like to thank my colleagues from Dr. Hu's research group: Yi Sun, Feng Gu, Fasheng Qiu, Fan Bai, Minghao Wang, and Haidong Xue for the useful discussion of research ideas. I would like to extend my gratitude to my parents Longyu Guo and Lianrong Wu for their strong, persistent encouragement, understanding and continuing support during my pursuit of my educational goals. Last but not least, special gratitude goes to my wife Dan Jiang for the encouragement and support for my research. Particularly, I would like to dedicate my work to the gift from heaven, my lovely boy Michael.

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	13
1.1	Problem statement.....	13
1.2	Service-oriented architecture	14
1.3	Why simulation software as a service.....	17
1.4	Main contributions.....	18
1.4	The organization of the work	21
CHAPTER 2	RELATED WORK	23
2.1	Cloud computing and software as a service.....	23
2.2	Simulation software as a service	25
2.3	Modeling and simulation frameworks.....	27
2.3.1	Discrete event system specification.....	28
2.3.2	Extensible modeling and simulation framework.....	32
2.3.3	High level architecture.....	32
2.4	Theories of integration of systems and models.....	34
2.4.1	Interoperability of systems	34
2.4.2	Composability of systems.....	37
2.5	Workflow tools and systems	39
2.5.1 Business workflow technology	
.....	40
2.5.2	Scientific workflow technology.....	40
CHAPTER 3	SPECIFICATION FOR SIMULATION SERVICE AND SERVICE-ORIENTED SIMULATION EXPERIMENT	43
3.1	Stateful vs. stateless	43
3.3	A specification for simulation software as a service and service-oriented simulation experiment.....	43
3.3.1	Specification for simulation software as a service	44
3.3.2	Specification for service-oriented simulation experiment.....	49
3.4	Service-oriented simulation experiment results collection	50
CHAPTER 4	COMPOSABILITY OF SIMUALTION SERVICES.....	52
4.1	Interaction patterns.....	52
4.1.1	Request-response	52

4.1.2	Publish-subscribe.....	53
4.1.3	Periodical synchronization	54
4.2	Coordinator service in simulation service composition	54
4.3	Maturity reference model for simulation service composability	57
4.4	Time granularity and event granularity in simulation service composition.....	59
4.4.1	Time granularity	63
4.4.2	Event granularity	65
4.4.3	Value message and delta message in granularity handling	66
4.4.4	Granularity handling agents.....	67
4.4.5	Granularity-enhanced coupling in simulation service composition	70
4.4.6	Experiment results	71
CHAPTER 5	SERVICE-ORIENTED SIMULATION EXPERIMENT	76
5.1	Existing workflow systems.....	76
5.2	The Proposed Service-Oriented Simulation Experiment	80
5.2.1	Composed simulation service and simulation experiment	80
5.2.2	Web browser-based execution approach for service-oriented simulation experiment	82
5.2.3	BPEL-based execution approach for service-oriented simulation experiment	87
5.3	An experiment example using the wildfire simulation service and the optimization service.....	89
CHAPTER 6	GRAPHICAL USER INTERFACE OF THE PROPOSED WORK	94
6.1	Graphical user interface for wildfire simulation service.....	94
6.2	Graphical user interface for simulation service composition	98
6.3	Graphical user interface for service-oriented simulation experiment	103
CHAPTER 7	SPATIAL PARTITIONING ALGORITHMS FOR PARALLEL WILDFIRE SIMULATION.....	110
7.1	DEVS-FIRE Overview	111
7.1.1	System architecture of DEVS-FIRE.....	111
7.2	Spatial uniform partition for parallel simulation of DEVS-FIRE.....	114
7.3	Profile-based spatial partition for parallel simulation of DEVS-FIRE.....	117
7.3.1	Architecture	118
7.3.2	Producing low resolution GIS data.....	119

7.3.3	Produce profile queue	124
7.3.4	Profile-based partition with spatial granularity	126
7.3.5	Parallel simulation based on the profile-based spatial partition	130
7.4	Experiment results	135
7.4.1	Simulations with different ignition locations and multiple ignitions points	136
7.4.2	Simulations with different granularities	144
7.4.3	Simulations with different number of PUs	147
7.4.4	Simulations with “watch dog” mechanism.....	149
CHAPTER 8	CONCLUSIONS AND FUTURE WORK	153
8.1	Conclusions	153
8.2	Future work	154
References	156

LIST OF TABLES

Table 2.1 Levels of Conceptual Interoperability	35
Table 7.1 The speed up of both partitioning methods	140
Table 7.2 Locations of the ignitions	142
Table 7.3 Speedup of different number of PUs	148

LIST OF FIGURES

Figure 1.1 Web service architecture	15
Figure 1.2 The proposed layered framework	21
Figure 2.1 DEVS/SOA architecture.....	31
Figure 2.2 Four model composability approaches	38
Figure 3.1 The meta-model of a simulation service	44
Figure 3.2 An example of composed simulation service.....	49
Figure 4.1 Request-response.....	53
Figure 4.2 Publish-subscribe.....	53
Figure 4.3 Periodical Synchronization.....	54
Figure 4.4 Federation of DEVS with Non-DEVS Simulation Service.....	57
Figure 4.5 Composition of the wildfire simulation service and the weather simulation service..	63
Figure 4.6 The state chart diagram for time granularity agent for value message.....	68
Figure 4.7 The state chart diagram for time granularity agent for delta message	68
Figure 4.8 The state chart diagram for event granularity agent for value message	69
Figure 4.9 The state chart diagram for event granularity agent for delta message.....	70
Figure 4.10 Granularity handling agents in the couplings of the composition.....	71
Figure 4.11 Simulation results of composition of the wildfire simulation service and the weather simulation service not using granularity handling agents.....	72
Figure 4.12 Simulation results of composition of the wildfire simulation service and the weather simulation service using granularity handling agents.....	74
Figure 5.1 Major scientific workflow systems	77
Figure 5.2 BPEL development environment	79
Figure 5.3 Architecture of the web browser-based execution approach.....	83
Figure 5.4 Invoke work unit.....	84
Figure 5.5 While work unit.....	85
Figure 5.6 Wait work unit.....	85
Figure 5.7 if-else work unit.....	86
Figure 5.8 Architecture of the BPEL-based execution approach	87
Figure 5.9 Experiment workflow using web-browser based approach for the wildfire simulation service and optimization service	90

Figure 5.10 Experiment workflow scenario uses the <i>invoke work unit</i>	91
Figure 5.11 Experiment workflow scenario uses the <i>assign work unit</i> and the <i>while work unit</i> ..	92
Figure 5.12 Experiment workflow scenario uses the <i>wait work unit</i>	93
Figure 6.1 GUI for wildfire simulation service	96
Figure 6.2 GUI for simulation service composition	100
Figure 6.3 GUI for service-oriented simulation experiment.....	105
Figure 7.1 Overall system architecture of DEVS- FIRE	112
Figure 7.2 Examples of 2D cell space	114
Figure 7.3 Three simple approaches to partition the cellular space.....	115
Figure 7.4 A simulation scenario in DEVS-FIRE	118
Figure 7.5 Architecture of the profile-based spatial partition method.....	119
Figure 7.6 Examples of “dominant fuel, averaged slope and aspect” rule	121
Figure 7.7 An example using our “dominant fuel, averaged slope and aspect” algorithm.....	123
Figure 7.8 Producing profile using low resolution GIS data: the sequence in the <i>PQ</i> represents the order this cell is ignited.....	126
Figure 7.9 Examples of partitioning with different granularities	128
Figure 7.10 Different locations of the ignition point in the cell space	138
Figure 7.11 Experiment result.....	140
Figure 7.12 Work load diagrams	141
Figure 7.13 Experiment results.....	143
Figure 7.14 Work load diagrams	146
Figure 7.15 Work load diagrams	148
Figure 7.16 Experiment scenario for “watch dog” mechanism	150
Figure 7.17 Experiment results.....	151

CHAPTER 1 INTRODUCTION

1.1 Problem statement

Simulation software, generally based on the process of imitating a real phenomenon with a set of mathematical formulas, is widely used in simulating wildfire spread process, weather conditions, electronic circuits, chemical reactions, and so on. Simulation software with real-time response has important industrial applications, especially for those where the penalty for improper operation is costly, such as wildfire fighting, nuclear power plant, airplane or chemical plant. Simulation software may manifest one or more of the following characteristics:

- 1) dynamic or mathematical models as core
- 2) real-time interactive
- 3) long running time
- 4) resource consuming
- 5) stateful
- 6) complicated inputs and/or outputs
- 7) requirement for cooperation with other simulation software

In the past, researchers and field experts had to either require the simulation software installed on their own machines or obtain the access to the machines that can host the simulation software to do their experiments or research. Sometimes, the experiments may be impeded by the fact that one simulation may depend on other simulations which may be not immediately available.

With the advancement of the web technology, new computing paradigm such as cloud computing [1] and service-oriented architecture (SOA) [2] provides new computing environments and methodologies for software system development via the Internet. Cloud

computing is a technology that uses the internet and central remote servers to maintain data and applications. Cloud computing allows consumers and businesses to use applications without installation and access their personal files at any computer with internet access. This technology allows for much more efficient computing by centralizing storage, memory, processing and bandwidth. Cloud computing can be segmented into three categories application, platforms, and infrastructure, each one of which serves a different purpose and offers different products for its end user all over the world.

1.2 Service-oriented architecture

SOA is a methodology with which a new application is created through integrating existing and independent business processes which are distributed over the networks. The business processes are called modules or services which communicate with each other, passing a message through the networks. This design concept requires interoperability between heterogeneous systems and languages and orchestration of services to meet the purpose of the creator. One of the implementations of the SOA concept is web service, which is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL [3]). Other systems interact with the Web service in a manner prescribed by its description using SOAP (Simple Object Access Protocol) [4] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. Publishing and discovering WSDLs is managed by Universal Description Discover and Integration (UDDI) [5], which is a platform-independent and XML style registry. Thus, three roles are classified in web service architecture: a service provider, a service discovery agency (UDDI), and a service requestor. The interaction of the roles involves publishing, finding, and binding operations. A

service provider defines a service description for a web service and publishes it to a service discovery agency. This operation publishes operations between the service provider and the service discovery agency. A service requestor uses a finding operation to retrieve a service description locally or from a discovery agency and uses the service description to bind it with a service provider and invoke or interact with the web service implementation. Figure 1.1 illustrates the basic web service architecture describing three roles and operations with WSDL and SOAP.

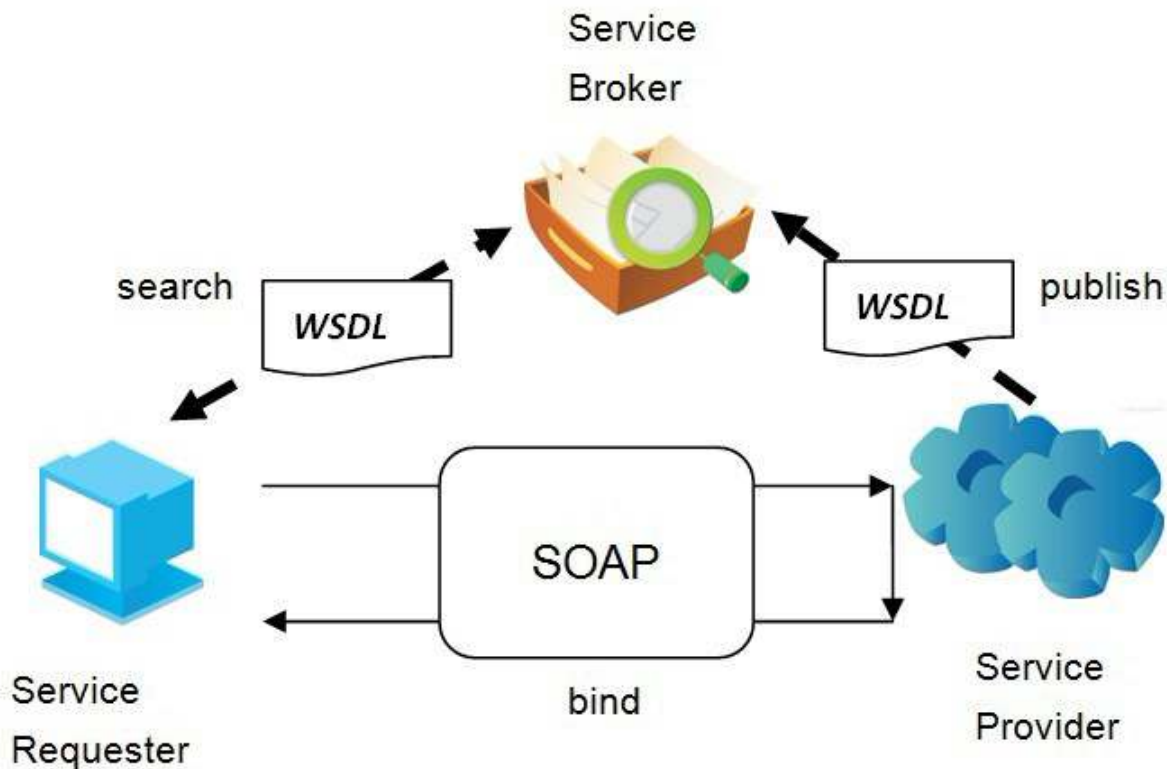


Figure 1.1 Web service architecture

As the web service gains its popularity and the technologies that enhance web service progresses, the notion “everything as a service” sprouts fast during recent years. This is also

applied to cloud computing and results in three service forms: software-as-a-service (SaaS) [6], platform-as-a-service (PaaS) [7], and infrastructure-as-a-service (IaaS) [8]. SaaS sometimes referred to as "on-demand software," is a software delivery model in which software and its associated data are hosted centrally (typically in the (Internet) cloud) and are typically accessed by users using a thin client, normally using a web browser over the Internet. PaaS and IaaS have the similar concept with SaaS. PaaS delivers a computing platform and/or solution stack as a service. IaaS delivers computer infrastructure – typically a platform virtualization environment – as a service.

A workflow [9] consists of a sequence of connected steps, which can model real work for further assessment, e.g., for describing a reliably repeatable sequence of operations. With the existence of web service technology, each operation in a workflow now can be represented by a web service. Designing a workflow can be viewed as making various web services to communicate and cooperate with each other in an organized fashion to achieve a certain purpose. In the business world, a business process, which can be viewed as a workflow, contains a set of interrelated tasks linked to an activity that spans functional boundaries. Business processes have starting points and ending points, and they are repeatable. Bearing the notion of “providing an environment where better business applications can be developed with less effort”, Business Process Execution Language for Web Service (BPEL4WS or BPEL) [10] is developed by combining the best of both WSFL [11] and XLANG [12] into one cohesive package. BPEL allows composition of web services and is thus the top-down approach to SOA – the process oriented approach to SOA.

In scientific experiments, the overall process, containing multiple steps, tasks and data flow, can be described as a Directed Acyclic Graph (DAG). This DAG is referred to as a workflow, e.g.

Brain Imaging workflows. Compared to the business world, there are similar needs for the sharable, reusable and composable scientific computing services and the automation of the composition of these services into a workflow in the scientific and research world. Thus, it is desirable to have:

- (1) simplified process for scientists and researchers to publish and share their scientific computing services within the community
- (2) an easy-to-use environment for scientists and researchers to compose different services according to a certain context and design the experiments (basically a workflow) they want
- (3) interactive tools for scientists and researchers to execute their experiments and view their results in real-time

1.3 Why simulation software as a service

Software as a Service (SaaS) has gained momentum in the past few years and the business world has been increasingly moving to SaaS model for their IT solutions. In the academic world, there is also a such need to provide the computing resources such as infrastructure, platforms, and software for researchers and scientists in a service fashion. In this work, we are fulfilling this need in the world of modeling and simulation. The reason we propose simulation software as a service is that it is beneficial for simulation software to exploit the tremendous resources, such as large number of processors, all kinds of platforms, and huge amount of storage, that cloud computing is able to provide. Another attraction to bring the SaaS light to modeling and simulation world is the multi-tenant architecture (MTA) that structures the software in the cloud computing.

However, bearing in mind the benefit brought by SaaS, this work focuses on some issues

that directly relate to modeling and simulation after it is led into this new paradigm. Although simulations can be wrapped into web services, there is still one intrinsic difference between regular software and simulation: the fundamental part of the simulation is a dynamic system where its elements change over time. Usually, this dynamic system is represented by a class of mathematical equations that are time-based with particular properties. Thus, the time factor is very crucial and critical to the correctness of the simulation in that the simulation must guarantee and maintain the correct temporal order of the events occurring in the simulation. One should note that the term “time” used here refers to the simulation time, not the actual time we live in. Another difference emerges when different simulations are composed together. Because of the temporal order, the interactions among simulations should be governed to retain the temporal order. This means that the requirement of composition of simulation services should incur extra work compared to that of composing regular web services which do not have such a tight coupling. Thus, this deserves effort to investigate.

1.4 Main contributions

A layered framework is proposed to support our research work as shown in Figure 1.2. There are five layers in the framework from the top to the bottom: graphical user interface layer, simulation experiment layer, simulation service layer, simulation execution engine layer and infrastructure layer.

Graphical user interface (GUI) layer is the representation layer of the simulation experiment and simulation service. It provides intuitive ways for user to interact with simulation services, and design simulation experiments. For this layer, we designed and implemented web-based GUIs for the wildfire simulation service, simulation service composition and service-oriented

simulation experiment, which aims to provide researchers and scientists an easy-to-use interface and real-time experiments and simulation results rendering.

At simulation experiment layer, simulation experiments are constructed by integrating simulation services, non simulation services and tools. In our case, simulation services include individual simulation services and composed simulation services. Non simulation services and tools are not simulation related and they have their own way of execution. The reason to include those non simulation services and tools is that they can provide some desirable features that will assist researchers and the scientists with their experiments. For this layer, we proposed a specification for the service-oriented simulation experiment.

Simulation service layer contains individual simulation services and composed simulation services. The individual simulation services are created by exposing the functionalities of the simulation as web service operations. Composed simulation services are constructed by governing multiple individual simulation services and/or composed simulation services within a common temporal factor. An important issue arises when composing different simulation services. That is there could be mismatches in the couplings among different simulation services, and these mismatches may degrade and handicap the composability of the simulation services. Bearing this in mind, we first proposed a specification for both individual simulation service and composed simulation service at this layer. Particularly, we paid special attention to the composability of the simulation service composition. In our work, we adopted a maturity reference model for simulation service composability, identified four types of mismatch and proposed four simulation agents to solve the mismatches in order to improve the quality of the composability of the simulation services.

Simulation execution engine layer provides simulation engine for the simulation services. In our work, we employ DEVS [13] modeling and simulation framework. One problem existing at this layer is that the performance of certain simulation, for example wildfire simulation, is not ideal especially when the simulation scale becomes extremely large. To improve the performance of the wildfire simulation, we proposed two spatial partition algorithms for the parallel execution of the wildfire simulation.

The infrastructure layer provides the executing environment for the simulation engine and other services and tools. Typically, cloud computing environment will be chosen as the underlying infrastructure for our work and we assume that everything such as multitenancy, management and security issues at this layer has been taken care of.

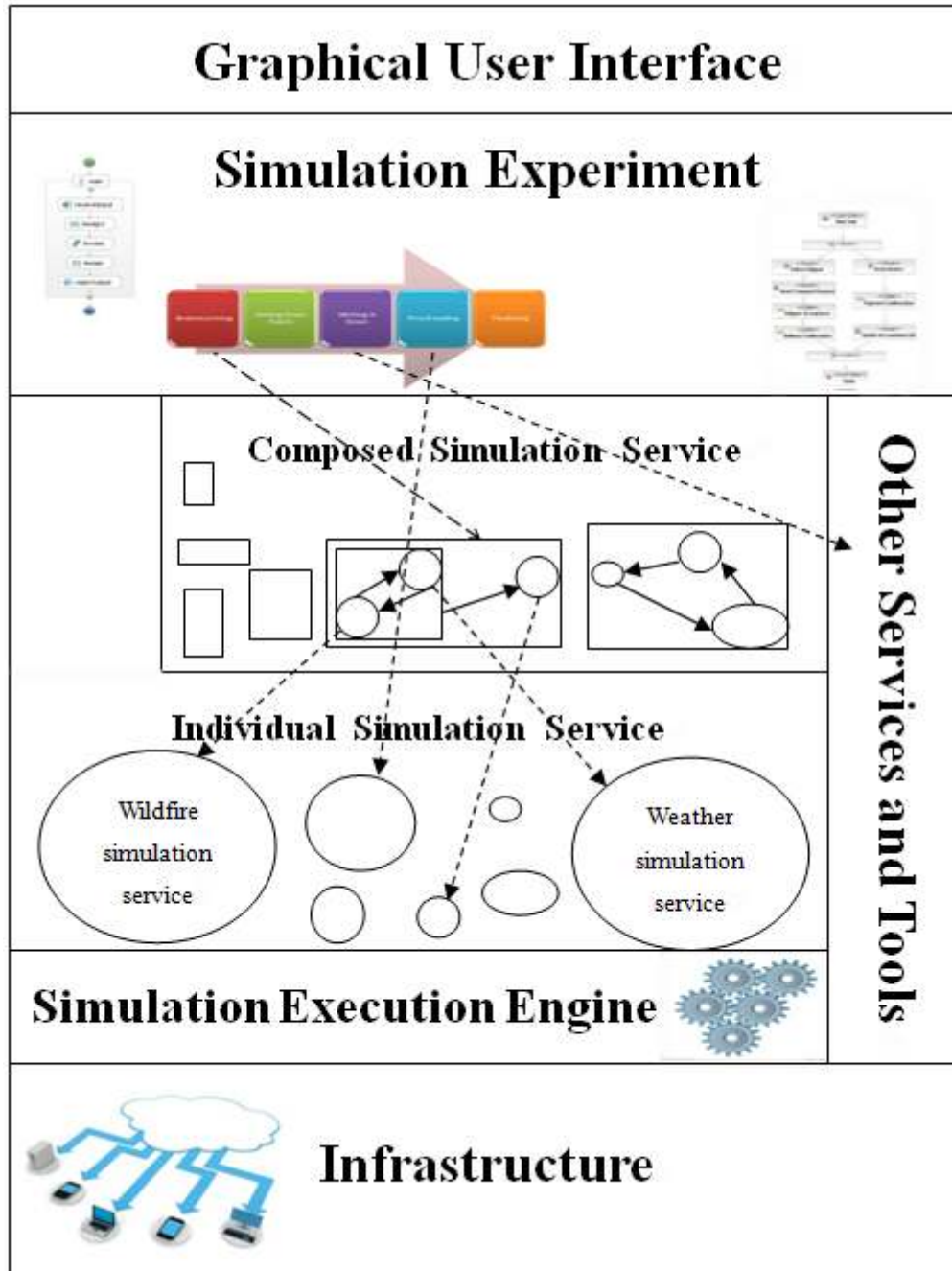


Figure 1.2 The proposed layered framework

1.4 The organization of the work

Based on the SOA methodology, the work will construct the entire system consisting of all the components, which will be explained later. Chapter 2 introduces the related research work of

simulation software as a service, modeling and simulation framework, theories of integration of systems and models, and workflow tools and systems. Chapter 3 describes the framework we proposed. Chapter 4 focuses on the composability of the simulation service. In Chapter 5, the service-oriented simulation experiment is presented. We design different user interfaces for different parts of the proposed framework in Chapter 6. Also, two partitioning algorithms which aim to improve the wildfire simulation performance are presented in Chapter 7. Finally, Chapter 8 gives conclusions and the future work.

CHAPTER 2 RELATED WORK

2.1 Cloud computing and software as a service

Cloud computing has drawn tremendous attentions with its ability to rapidly deliver computing resources, such as processing units, memory, storage etc, in a dynamic, scalable, and virtualized manner. As the cloud computing technology advances, there is a movement from computing as a product to computing as a service. The term “cloud computing” originates from computer network diagrams that represent the internet as a cloud. The US National Institute of Standards and Technology (NIST) has developed a working definition that covers that commonly agreed aspects of cloud computing:

a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can rapidly provisioned and released with minimal management effort or service provider interaction.[82]

In the above definition, it describes five essential characteristics, four deployment models and three service models. The essential characteristics are: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. The four deployment models are: private cloud, public cloud, community cloud, and hybrid cloud. The three service models are software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS).

Software as a service (SaaS), sometimes referred to as “on-demand software”, is a software delivery paradigm in which software and its associated data are hosted centrally in the cloud computing environment and are typically accessed by users using a thin client, normally using a web browser over the Internet. Software as a service gains success in business world due to its

multi-tenant architecture. Multitenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple users (tenants). It is regarded as one of the essential attributes of cloud computing. The work of [77] develops a multi-tenant placement model which decides the best server where a new tenant should be accommodated. The placement mainly considers the hardware resources including CPU and storage usage. In principle, a new tenant will be placed on the server with minimum remaining residual resource left that meets the resource requirement of the new tenant. In [78] authors found that the available multi-tenant database mapping techniques are not optimal. They proposed a novel customizable database design technique to create multi-tenant applications, by introducing an EET (Elastic Extension Tables) which consists of CTT (Common Tenant Tables) and VET (Virtual Extension Tables). EET enable tenants to create their own virtual database schema including: the required number of tables and columns, virtual database relationships with any of CTT or VET, and assigning a suitable data type and constraints of columns during multi-tenant application run-time execution. Oliver Schiller, al. et in [79] introduced the concept of a tenant context that logically assembles all information that describes the tenant's view of the database and cleanly isolates tenants among another. They also presented a schema inheritance concept tailored to multi-tenancy, which offers different schema types for different challenges. Their approach eases the development of multi-tenant SaaS applications that rely on a RDBMS, as it prevents the implementation of a complex query rewriter on top of the RDBMS. It also facilitates the central maintenance of the application core schema and the individual maintenance of tenants' schema. Configurability is one of the keystones to the success of any SaaS software. Configurability allows the single instance multiple tenant model which leads to many benefits both for the customers and the vendors which in turn has led to the acceptance and popularity of

SaaS. In [80], it addresses the issue of how to effectively and efficiently support configurability in SaaS software and proposes SaaS architecture to support configurability. The author aims to provide information on the nature of configurability in SaaS software, how it can be provided and what technologies should be employed in order to support it. He gave out a sample case of building SaaS application for University Grading Management System, analyzed in the configurable aspects of user interface, workflow, data, and access control, and then presented a SaaS enabled solution which supports those configurable aspects of SaaS software. There have also been studies on service performance issues in multi-tenant SaaS. Multi-tenancy's intention is to satisfy requests from different tenants concurrently by a single service instance over shared hosting re-sources. However, extensive resource sharing easily causes inter-tenant performance interference. [81] proposes a Service Performance Isolation Infrastructure (SPIN) which allows extensive resource sharing on hosting systems. By a detection independent of any threshold, SPIN gives anomaly report in advance of the instability of service system. Resources, like CPU, consumed during service access are accounted on behalf of each tenant. The SPIN prototype developed demonstrated its isolation efficiency on the Trade6 benchmark which is revised to support multi-tenancy. SPIN is shown to fit industry practice for a performance overhead less than 5%.

2.2 Simulation software as a service

Cloud computing has received significant attention recently as it is a new computing infrastructure to enable rapid delivery of computing resources as a utility in a dynamic, scalable, and virtualized manner. By taking advantage of various resources including physical resources such as processors, cache and logical resources such as in-memory data stores, distributed file system, and SaaS infrastructure in the cloud, we can put simulations into the cloud. Thus

simulations can be transformed in a SaaS fashion. Lanner group [83] examined the changing use of simulation as service to BPM (business process management) and developed a simulator for BPM systems in a cloud. T. Paviot [84] developed simulation CAD services in a cloud. W. Tsai et al [85] proposed SimSaaS framework that incorporates key elements of SaaS features, configuration, MTA (multi-tenancy architecture) and scalability, for simulation, designed a platform-independent configuration model to support MTA, and presented a simulation runtime infrastructure that supports MTA.

Developing, running, and analyzing a simulation model can be a very time-consuming and error-prone process. It is very desirable to develop highly modular modeling and simulation environments. Component-based modeling and simulation aims to provide such feature with the reusability of different simulation components. In addition, with the help of modern graphical user interface technology, it is possible to achieve construct new models by dragging and dropping existing components in a modeling process, which significantly relieves the burden in system modeling and reduces development time. Based on the component-based technology, different simulation environments can interact through standard interfaces and function together in the architecture such as HLA [28], which emphasizes on component-based modeling and simulation from the architecture point of view. Other works such as JSIM [86], SIMKIT [87], Skil [88], and VSE [89] focus on the implementation of component-based modeling and simulation environments. Y. Son et al. [90] presented the results of a collaborative effort among NIST, Penn State University, and University of Arizona to build the necessary component models and a template for a simple, discrete event simulation, which models the flow of jobs through a small job shop based on a pre-provided schedule. They derived a database structure from these formal models and discuss the population of that database with the data entries for the

sample job shop. They also examine the translators we developed to go from the neutral representation of the components to the representation required by two commercial simulation packages. Based on these work, they built a prototype of a simple service and preliminary experiments were conducted to investigate the performance of this prototype service for users requesting multiple services in parallel. J. Sommer and W. Franz [91] presented a component-based simulation model of a switched Ethernet network to evaluate enhancements to the Ethernet standards. In their work, they decomposed the network entities into a hierarchy of components interconnected in a plug-n-play manner. Those components form the basis of the simulation program. They also show the applicability of the simulation model and its implementation by using examples from the automotive domain. P. Cheung et al. [92] presented a novel approach to hardware/software co-simulation of component-based embedded systems. Their approach features a component model which unifies hardware and software component models with the concept of bridge component. Bridge components raise the level of abstraction for designing hardware/software interfaces. Their approach was applied to co-simulation of sensor system instances included in the TinyOS [93] distribution.

Our work differentiates from the above works by focusing on the composability aspect of the simulation services in the cloud. On one hand, we build our work on top of the work in [85] in order to take the advantage of the cloud computing environment; on the other hand, we also aim to provide reusability of the modularized simulation services in the distributed environment to ease the development of new simulations and experiments.

2.3 Modeling and simulation frameworks

In the modeling and simulation (M&S) community, Zeigler defines a modeling and simulation framework as [13]: “a framework that defines entities and their relationships that are

central to the M&S enterprise. The basic entities of the framework are source system, model, simulator and experimental frame. The basic interrelationships among entities are the modeling and the simulation relationships.” With the widespread application of modeling and simulation techniques in military, education, aeronautics, astronautics, commerce, communication, manufacture, and other communities, many discipline-specific M&S frameworks have been built up.

2.3.1 Discrete event system specification

DEVS [13] abbreviating Discrete Event System Specification is a modular and hierarchical formalism based modeling and simulation framework for modeling and analyzing general systems that can be discrete event systems which might be described by state transition tables and continuous state systems which might be described by differential equations and hybrid continuous state and discrete event systems. DEVS is a timed event system. The basic atomic DEVS model is defined as a 7-tuple

$$M = \langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$$

where

X is the set of input events,

Y is the set of output events,

S is the set of sequential states,

$ta: S \rightarrow T^\infty$ is the time advance function which is use to determine the lifespan of a state,

$\delta_{ext}: Q \times X \rightarrow S$ is the external transition function which defines how an input event

changes a state of the system,

$Q = \{(s, t_e) | s \in S, t_e \in (T \cap [0, ta(s)])\}$ is the set of total states, and t_e is the elapsed time since the last event,

$\delta_{int}: S \rightarrow S$ is the internal transition function which defines how a state of the system changes internally,

$\lambda: S \rightarrow Y^\emptyset$ is the output function where $Y^\emptyset = Y \cup \{\emptyset\}$ and $\emptyset \notin Y$ is a silent event or an unobserved event.

In DEVS, the atomic models can be coupled together to form a multi-component that is defined by the following structure:

$$N = \langle X, Y, D, \{M_i\}, C_{xx}, C_{yx}, C_{yy}, Select \rangle$$

where

X is the set of input events,

Y is the set of output events,

D is the name set of sub-components,

$\{M_i\}$ is the set of sub-components where for each $i \in D$, M_i can be either an atomic

DEVS model or a coupled DEVS model,

$C_{xx} \subseteq X \times \bigcup_{i \in D} X_i$ is the set of external input couplings,

$C_{yx} \subseteq \bigcup_{i \in D} Y_i \times \bigcup_{i \in D} X_i$ is the set of internal couplings,

$C_{yy}: \bigcup_{i \in D} Y_i \rightarrow Y^\emptyset$ is the external output coupling function,

$Select: 2^D \rightarrow D$ is the tie-breaking function which defines how to select the event from the set of simultaneous events.

In the service-oriented architecture, several frameworks, which are built upon DEVS formalism, emerge and gain more and more attentions in recent years. Mittal, Risco-Martín and Zeigler in [14] proposed DEVS Modeling Language (DEVSML), which is built on XML and provides model interoperability among DEVS models located at remote locations. The DEVSML environment is built on client-server paradigm and the simulation is executed at the server's end.

In their work, the proposed DEVS atomic and coupled DTDs are open to standardization from the community for successful model sharing and collaboration. The DEVSML framework not only provides the needed feature of run-time composability of coupled systems using the SOA framework, but also provides the capability to translate model to and from XML and JAVA programming language leading to model composability and validation. Mittal in [15] proposed DEVS Unified Process framework (DUNIP) for integrated development and testing of service-oriented architectures. DUNIP uses an XML based DEVS Modeling Language (DEVSML) framework that provides the capability to compose models that may be expressed in a variety of DEVS implementation languages. The models are deployable for remote and distributed real-time executing agents over the Service Oriented Architecture (SOA) middleware, which is developed as the infrastructure of DUNIP, called DEVS/SOA [16] DEVS/SOA depicted in Figure 3 was proposed as a simulation service platform to address simulator compatibility issues like DEVS/C++ [17], DEVSJAVA [18], DEVS/RMI [19] etc. The simulation processes are totally transparent to model execution over the net-centric infrastructure. Users can execute models over Internet by Web services and SOA protocols. The composition and execution of models conforms to System Entity Structure (SES), modular, hierarchical DEVS specification, and DEVS simulation protocols [13].

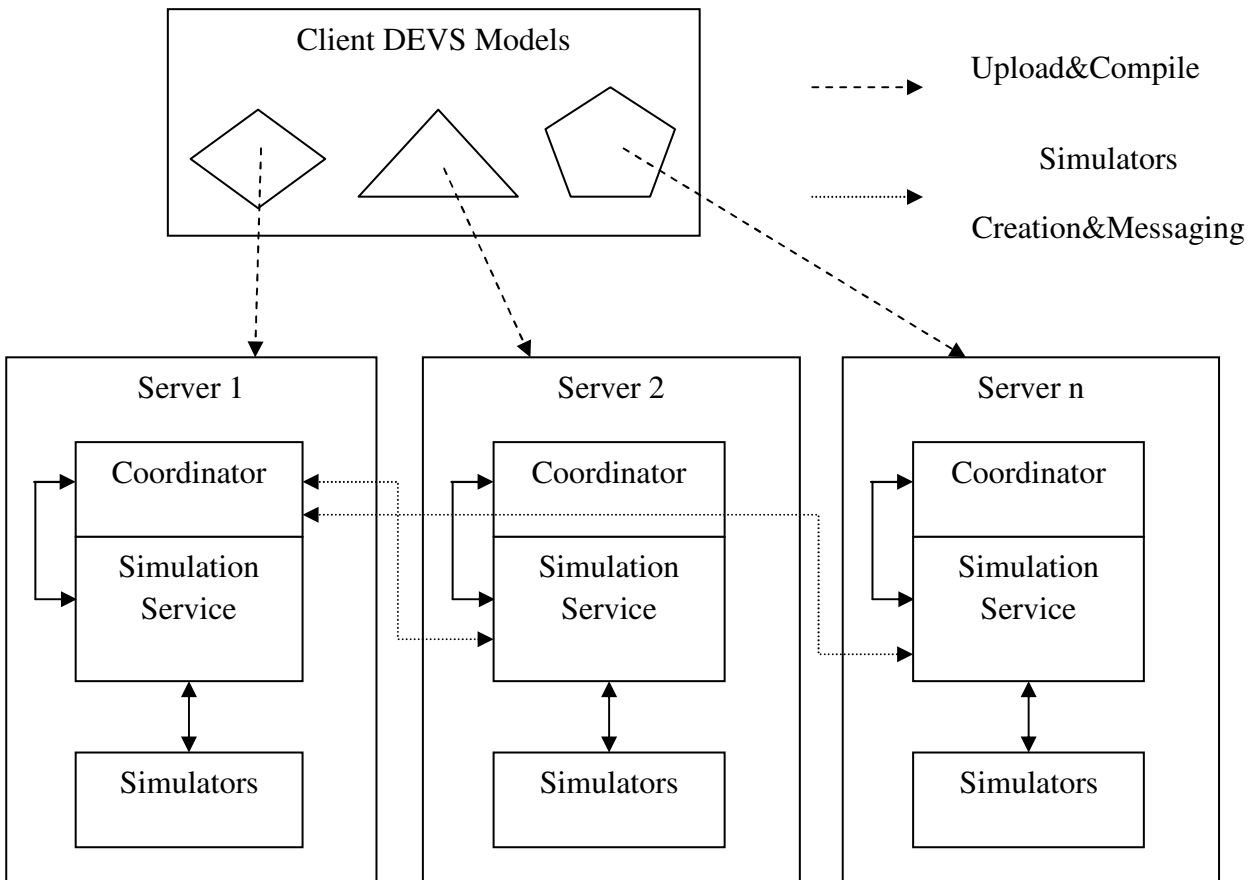


Figure 2.1 DEVS/SOA architecture

Because of the missing support for some basic SOA concepts in most M&S frameworks, there exist difficulties when modeling and simulating service-oriented computing systems. Hence, Sarjoughian et al. propose an SOA-compliant DEVS (SOAD) simulation framework [20] to address these issues. SOAD concerns the three roles in SOA, messaging patterns, primitive and composite service composition, and hardware model for router link. UNIP Web enables DEVS framework as service-oriented frameworks but the M&S objectives are not necessary service-oriented systems. While SOAD may not be service-orientation itself, however, the M&S objectives are service-oriented systems. Wainer et al. [21, 22] investigates the Web services based Cell-DEVS framework. Cell-DEVS is a DEVS-based formalism that defines spatial

models as cell spaces. Web enabling CD++, which is an M&S toolkit to execute Cell-DEVS models, can expose simulation functionalities as Web services to improve interoperability and reusability for users' convenience. The architecture of Web services based distributed simulation framework D-CD++ is shown in [21]. The set of service interfaces in D-CD++ includes session management, configuration, simulation modeling and control, and retrieving data interfaces. The execution of D-CD++ conforms to parallel DEVS simulation protocols and adopts global conservative time management strategy. The master and slave coordinators are used to reduce the number of exchanged messages among simulation services. The experiments and performance analysis are performed for D-CD++ both over the Internet and dedicated fiber optic link. It shows that the overhead of SOAP messaging is the major bottleneck.

2.3.2 Extensible modeling and simulation framework

XMSF [23, 24] is defined as a composable set of standards, profiles, and recommended practices for Web-based M&S. XMSF utilizes Web services and related techniques to build up a common M&S technique framework. The practice of XMSF includes the Web-Enabled RTI [25] and the project using XMSF to connect Navy Simulation System, Simkit, and CombatXXI for joint modeling and analysis sponsored by SAIC [26]. The Armed Forces of Korea also investigates the intelligent-XMSF approach base on autonomous Web Services [27]. The common technique framework of XMSF provides conceptual and technical support for service-oriented simulation. The related profiles of XMSF also provide experience in practice and implementation. The limitation of XMSF is its lack of concrete standards and implementation for service description, composition, and integration. It also lacks the support of software/systems engineering.

2.3.3 High level architecture

A high-level architecture (HLA) is general purpose architecture for distributed computer simulation systems. The High Level Architecture (HLA) [28] provides the specification of a common technical architecture for use across all classes of simulations in the US Department of Defense. It provides the structural basis for simulation interoperability. The baseline definition of the HLA includes (1) the HLA Rules, (2) the HLA Interface Specification, and (3) the HLA Object Model Template (OMT). Computer simulations can interact, including data communication and actions synchronization, with other computer simulations regardless of the computing platforms when using HLA. The interaction between simulations is managed by a Run-Time Infrastructure (RTI). In HLA, a federate is an HLA compliant simulation entity, which can be connected via the RTI using a common OMT to form a federation. The collection of related data sent between simulations is called an object. Events sent between federates are called interactions.

In the service-oriented architecture, Service-oriented HLA (SOHLA) [29] refers to the architecture enabled by SOA and Web Services etc. techniques which supports distributed interoperating services. According to the layers of HLA, Web enabling HLA can be implemented at four layers: at the communication layer (such as Web-Enabled RTI [30, 31]), at the interface specification layer (e.g., HLA Evolved Web Service API [32] and Unified Architecture [33]), at the federate interface layer (such as HLA Connector [33]) and at the application layer (e.g., HLA Island [32]). In the Swedish “HLA and SOA integration” in support for the network-based defense, the prototypical architecture has been implemented and tested. This project integrates four federates using the native API, WS API and HLA Connector respectively, which shows the feasibility of those approaches. At present, HLA Evolved Web

Service API is the latest progress using SOA and Web services technique to extend the HLA at the interface specification.

2.4 Theories of integration of systems and models

Systems built from sub-systems have interesting behavior and characteristics stemming from diverse capabilities of individual sub-systems as well as interactions among them. Theories that support the integration of heterogeneous systems and models developed by different simulation frameworks includes two important aspects: composability and interoperability. In M&S theory, current literature distinguishes between Composability of Models and Interoperability of Simulation. The former addresses the model challenges on higher levels, while the latter deals with simulation implementation issues and Integratability with network questions.

2.4.1 Interoperability of systems

It is not trivial to create interoperability of systems in that there are various levels of interoperability between two systems ranging from no interoperability to full interoperability. However, it is not uniquely new to employ multiple different levels or layers to classify interoperability. Within the technical domain, various models of interoperability exist. One of the matured models in this context is the “Levels of Information Systems’ Interoperability (LISI)” model [34]. LISI has the following layers:

Isolated Systems: No physical connection exists

Connected Systems: Homogeneous product exchange is possible

Distributed Systems: Heterogeneous product exchange is possible

Integrated Systems: Shared applications and shared data

Universal Systems: Enterprise wide shared systems

Within the context of the NATO C3 Technical Architecture (NC3TA) [35], NC3TA Reference Model for Interoperability (NMI) is used. NMI has the following degrees:

No Data Exchange: No physical connection exists

Unstructured Data Exchange: Exchange of human-interpretable, unstructured data

Structured Data Exchange: Exchange of human-interpretable structured data

Seamless Sharing of Data: Automated data sharing within systems based on a common exchange model

Seamless Sharing of Information: Universal interpretation of information through cooperative data processing

Although LISI models are used successfully to determine the degree of interoperability between information technology systems, they do not provide a systematic formulation of the underlying properties of information exchange. To remedy this situation, Tolk and Muguira [36] proposed a model name Levels of Conceptual Interoperability Model (LCIM), which focuses on the data to be interchanged and the interface documentation. Table 1 lists the outline of LCIM.

Table 2.1 Levels of Conceptual Interoperability

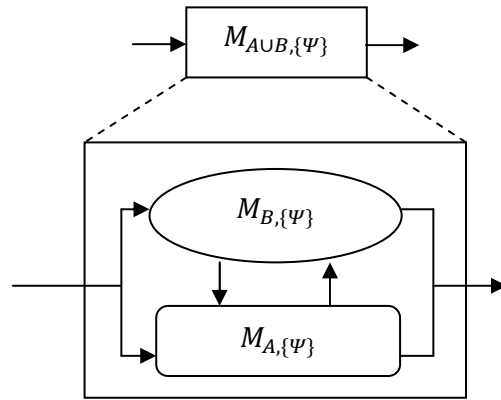
Level of Conceptual Interoperability	Characteristic	Key Condition
Conceptual	The assumptions and constraints underlying the meaningful abstraction of reality are aligned.	Requires that conceptual models be documented based on engineering methods enabling their interpretation and

		evaluation by other engineers
Dynamic	Participants are able to comprehend changes in system state and assumptions and constraints that each is making over time, and are able to take advantage of those changes.	Requires common understanding of system dynamics
Pragmatic	Participants are aware of the methods and procedures that each is employing.	Requires that the use of the data – or the context of their application – is understood by the participating systems.
Semantic	The meaning of the data is share	Requires a common information exchange reference mode.
Syntactic	Introduces a common structure to exchange information.	Requires that a common data format is use.
Technical	Data can be exchanged between participants.	Requires that a communication protocol

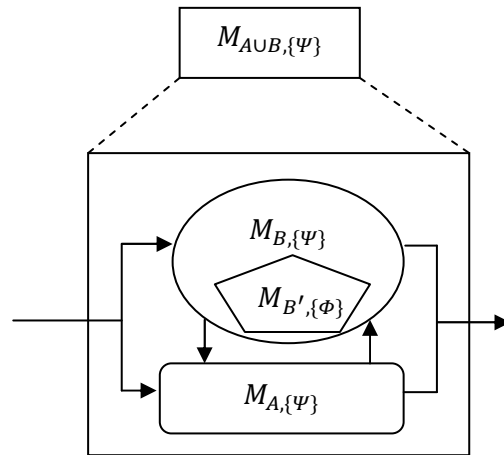
		exists.
Stand alone	No interoperability	

2.4.2 Composability of systems

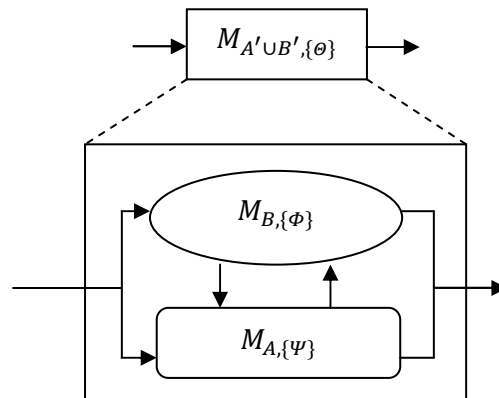
Composition of models is considered essential in developing heterogeneous complex systems and in particular simulation models capable of expressing a system's structure and behavior. Generally, a modeling formalism can be defined to consist of two parts: model specification and execution algorithm [13]. Model specification is a mathematical theory that describes structures and behavior of the modeling targets. Execution algorithm specifies an algorithm to correctly execute any model that is described according to the model specification. In [37], a model A that can be specified in a modeling formalism Ψ is denoted as $M_{\cup A, \{\Psi\}}$. A model composed of a finite number of disparate models (e.g., A, B, \dots, K) specified in a finite number of distinct modeling formalisms (e.g., Ψ, Φ, Ω) is denoted as $M_{\cup A, B, \dots, K, \{\Psi, \Phi, \Omega\}}$. The author presented a classification of four model composability approaches, which are Mono Model Composability, Super Model Composability, Meta Model Composability and Ploy Model Composability. Formulation for each of the four model composability approaches is described in terms of modeling formalisms and simply illustrated using the following four figures in Figure 2.2.



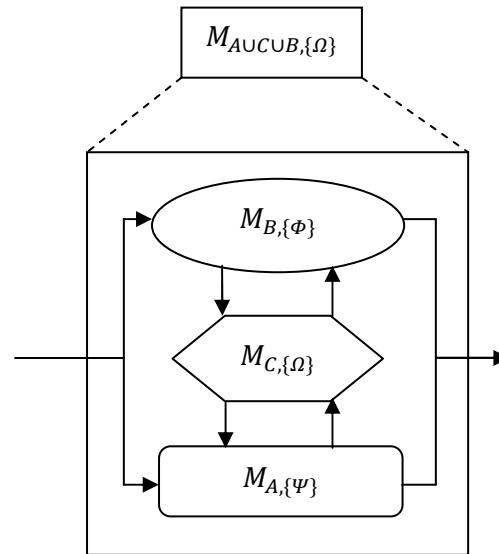
(a) mono composability approach



(b) super composability approach



(c) meta composability approach



(d) ploy composability approach

Figure 2.2 Four model composability approaches.

2.5 Workflow tools and systems

An atomic web service is a basic unit of operation in a Service-oriented Architecture (SOA). However, in general, an atomic web service is not able to satisfy the functional requirements of complex tasks. Therefore, it is desirable to logically connect several atomic web services to satisfy complex functional requirements, leveraging the loose coupling characteristics of SOA. In general, we would not expect to find an atomic web service that would take these exact inputs, produce the desired output, and satisfy any user constraints outlined in the task description. Therefore, we would need to take individual web services and put them together in a way that the goal of the landing plan is met. This requires decomposing an abstract specification into an abstract (composite) workflow composed of subtasks that eventually correspond to web services, with the subtasks connected through some process logic. There are several tools and systems that are designed and developed for this purpose

2.5.1 Business workflow technology

As workflow technology was first adopted by the business community, this has led to a space crowded with competing specifications, from opposing companies, some of which have risen to the top, superseding others. The Business Process Execution Language (BPEL) [38] is an executable business process modeling language and currently the current de-facto standard way of orchestrating web services. It has broad industrial support from companies such as IBM, Microsoft and Oracle. Industrial support brings concrete implementations, tools and training. Around BPEL, recent efforts from different sources have resulted in several BPEL designers with user-friendly GUI, such as BPEL Designer from Open Middleware Infrastructure Institute UK (OMII-UK) [39], eclipse BPEL Designer [40], ActiveBPEL Designer from active endpoints [41] and Oracle BPEL Process Manager [42]. Yet Another Workflow Language (YAWL) [43] is based on the rigorous analysis of workflow patterns, a particular type of design pattern. YAWL aims to support all (or most) of the workflow patterns and has a formal underpinning based on Petri-nets. The language is supported by an Open Source implementation and has some industrial support. XML Process Definition Language (XPDL) [44] is a format standardized by the Workflow Management Coalition (WfMC) to interchange Business Process definitions between different workflow products like modeling tools and workflow engines. WfMOpen [45] is an open-source J2EE based implementation of a workflow engine as proposed by the Workflow Management Coalition (WfMC) and the Object Management Group (OMG), WfMOpen uses XPDL as input.

2.5.2 Scientific workflow technology

The concepts of workflow have recently been applied to automating large-scale science (or e-Science), coining the term scientific workflow [46]. A scientific workflow attempts to capture

a series of analytical steps, which describe the design process of computational experiments. Scientific workflow systems provide an environment to aid the scientific discovery process through the combination of scientific data management, analysis, simulation, and visualization.

Taverna [47] is an open-source, Grid-aware workflow management system; it provides a set of transparent, loosely-coupled, semantically-enabled middle-ware to support scientists that perform data-intensive in-silico [48] experiments on distributed resources. Taverna is implemented as a service-oriented architecture, based on Web service standards. Provenance [49] plays an integral part in Taverna, allowing users to capture and inspect details such as who conducted the experiment, what services were used, and what the results of services provided. Taverna uses a proprietary language, the Simple Conceptual Unified Flow Language or SCUFL [50] for short. The SCUFL language is a high level XML-based conceptual language allowing a user to define workflow through groups of local or remote services which are connected with data links (providing data flow) and control links (allowing coordination of services not connected through data flow). The Taverna workbench depends on the FreeFluo engine [48].

Kepler [51] is an open-source scientific workflow engine with contributors from a range of application-oriented research projects. Kepler is built upon the Ptolemy II system [52] based at the University of California at Berkeley, which is a mature dataflow-oriented workflow architecture. In Kepler, the focus is on actor-oriented design. Actors are re-usable independent blocks of computation, such as: web services, database calls etc. They consume data from a set of inports and write data to a set of outports. A group of actors can then be wired together by introducing a mapping from outports to inports. A novel feature in Kepler allows the actor communication (dataflow) concerns to be separated from the overall workflow coordination, which is defined in a separate component called a director. This separation allows a workflow

model to be run with different execution semantics, such as synchronous dataflow and process networks. Kepler provides a large variety of computational models inherited from the Ptolemy II system and uses the proprietary Modeling Markup Language.

CHAPTER 3 SPECIFICATION FOR SIMULATION SERVICE AND SERVICE-ORIENTED SIMULATION EXPERIMENT

3.1 Stateful vs. stateless

Simulation service is different from a regular web service in that a simulation model is a dynamic model where the concepts of time and state are essential. The model of a simulation can be abstracted as a timed Finite State Machine (FSM), which goes through many state transitions, from the initial state to the final state in a temporal order. Thus, we differentiate simulation service from regular web service and view simulation service as stateful service, which treats each service request as a series of dependent transaction that is related to the previous requests and the model's current state. In the example of wildfire spread simulation, the wildfire simulation service receives a message that adds a new ignition to the instantiated simulation, and the service should "know" which simulation instance this message corresponds to. Once this message is "routed" to the simulation instance in question, that simulation instance will be responsible for this message and makes response based on the content of the message and its current state. To achieve this, the factory pattern can be used, as described in [53]. Whenever the user wants to use the simulation service, an instance of the simulation this simulation service represents should be created and a session-ID-like key is returned for the future interaction with that simulation instance within the simulation service. When the simulation finishes, the instance should be destroyed and resources should be released.

3.3 A specification for simulation software as a service and service-oriented simulation experiment

When the scientists and researchers search for the simulation services they need, they expect the services have some unified interfaces which are easy to use. From system composition point of view, this is also desired. However, different simulation services may come from different communities and the interfaces of different services may vary greatly. This poses an important issue when they are coupled together. To solve this problem, we propose a specification for simulation software as a service and service-oriented simulation experiment to provide unified interface.

3.3.1 Specification for simulation software as a service

As for a simulation, it can be perceived as a black box, which is shown in Figure 3.1, associated with its inputs, outputs, initial state, resource data, control and configurable model parameters. To make a simulation as a service, these six aspects should be well defined and modeled in order to be used by the service consumers.

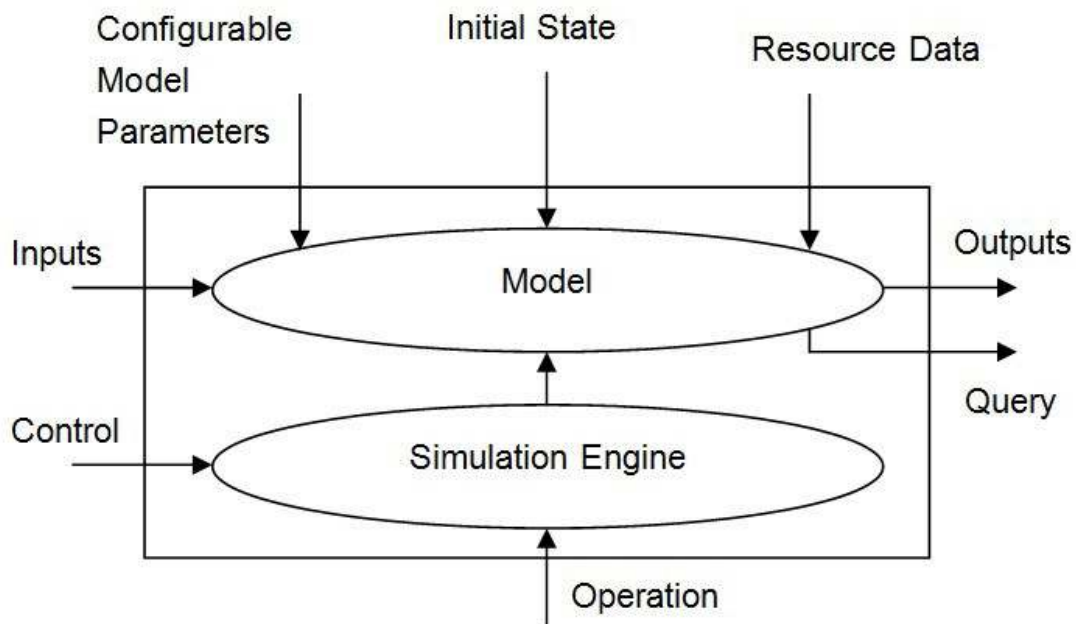


Figure 3.1 The meta-model of a simulation service

A single simulation service is a ten-tuple:

$$SimSaaS = \langle Namespace, ID, [I], [O], initialS, [RD], Control, [CMP], [Operation], [Query] \rangle,$$

where

Namespace is the identifier that uniquely distinguishes one simulation service from another.

ID is the instance identifier of the simulation service. It is used to distinguish multiple instances of the same simulation services.

[*I*] represents a set of inputs associated with the simulation service. For example, the ignition point location is an input of the wildfire simulation service.

[*O*] represents a set of outputs associated with the simulation service. For example, the burned area and perimeter of the fire at certain simulation timestamp are two outputs of the wildfire simulation service.

initialS represents the simulation initial state. It is the starting state of a simulation instance. In wildfire simulation, the initial state is the initial fire shape.

[*RD*] represents a set of the resource data, which are used by the simulation as resources. For example, fuel data, aspect data, and slope data are the resource data that are used in wildfire simulation. These three data lay the ground where the fire propagates.

Control represents *Start*, *Stop*, *Pause*, *Continue* commands.

[*CMP*] represents a set of the configurable model parameters, which are used to configure the simulation model. For example, the burning threshold is a model parameter in the wildfire simulation that determines at what level a model can be ignited.

[*Operation*] represents a set of simulation related operations. These operations will be used by the coordinator service when composed with other simulation services.

[*Query*] represents a set of interfaces that can be used by the user to obtain model related outputs.

Sometimes, some simulation services need to be composed together in order to achieve accuracy or these simulation services depend on each other to work properly. Thus, a specification for composed simulation service is also needed, which can be represented by an eleven-tuple:

Composed SimSaaS = \langle [*SimSaaS*], *Coordinator Service*, [*Composition Coupling*], [*I*], [*O*], [*initialS*], [*RD*], *Control*, [*CMP*], [*Operation*], [*Query*] \rangle ,

where

[*SimSaaS*] represents a set of simulation services used for composition.

[*Composition Coupling*] represents a set of connections that define the data flow among the simulation services. The composition coupling can be represented as *Composition Coupling* = \langle *SimSaaS_i.NameSpace*, *SimSaaS_i.ID*, *SimSaaS_i.O*, *SimSaaS_j.NameSpace*, *SimSaaS_j.ID*, *SimSaaS_j.I*, [*Property*] \rangle . It specifies one output of *SimSaaS_i* should be fed to one input of *SimSaaS_j* in the composed simulation service. [*Property*] is a set of properties that associates with this coupling. It defines pragmatic level interoperability between two simulation services, which will be detailed in the next section.

[*I*] represents a set of inputs associated with this composed simulation service. It contains a union set or a subset of the union set of all the inputs of all the simulation services employed in this composed simulation service.

[*O*] represents a set of outputs associated with this composed simulation service. It contains a union set or a subset of the union set of all the outputs of all the simulation services employed in this composed simulation service.

[*initialS*] represents a set of initial states for each simulation service employed in this composed simulation service.

[*RD*] represents a set of the resource data associated with this composed simulation service. It contains a union set of all the resource data of all the simulation services employed in this composed simulation service.

Control is represented by the *Control* of *Coordinator Service*. In other words, a composed simulation service is controlled via its *Coordinator Service*.

[*CMP*] represents a set of the configurable model parameters associated with this composed simulation service. It contains a union set of all the configurable model parameters of all the simulation services employed in this composed simulation service.

[*Operation*] represents a set of simulation related operations, which will be used during the execution of the composed simulation service. It is provided by the [*Operation*] of *Coordinator Service*, in order to be composed in a larger context with other simulation services.

[*Query*] represents a set of interfaces that can be used by the user to get model related outputs. It contains a union set of the [*Query*] of all the simulation services employed in this composed simulation service.

Coordinator Service is a special service that acts as the “glue” that actually connects different simulation services, coordinates communications among simulation services and

controls the execution phase of the composed simulation service. It is represented by a five-tuple:

$$\text{Coordinator Service} = \langle \text{Namespace}, \text{ID}, \text{Execution Protocol}, [\text{Operations}], \text{Control} \rangle,$$

where

Namespace is the identifier that uniquely distinguishes one coordinator service from another.

ID is the identifier that of the coordinator service. It is used to distinguish multiple instances of the same coordinator services.

Execution Protocol specifies how the simulation services in a composed simulation service should be executed. It guarantees the temporal order of the events occur in each participant simulation service in order to assure the correctness of the composed simulation service.

[*Operations*] represents a set of operations employed during the execution of the composed simulation service.

Control represents *Start*, *Stop*, *Pause*, *Continue* commands.

A composed simulation service is still a simulation service, and it can be further composed with other simulation services. Figure 3.2 shows an example of composed simulation services. The solid line with arrow illustrates the actual data flow and the dashed line with arrow represents the composed coupling. From the figure we can see that a composed simulation service retain the same interface as an individual simulation service, which means that it can be further composed in a larger context with other simulation services.

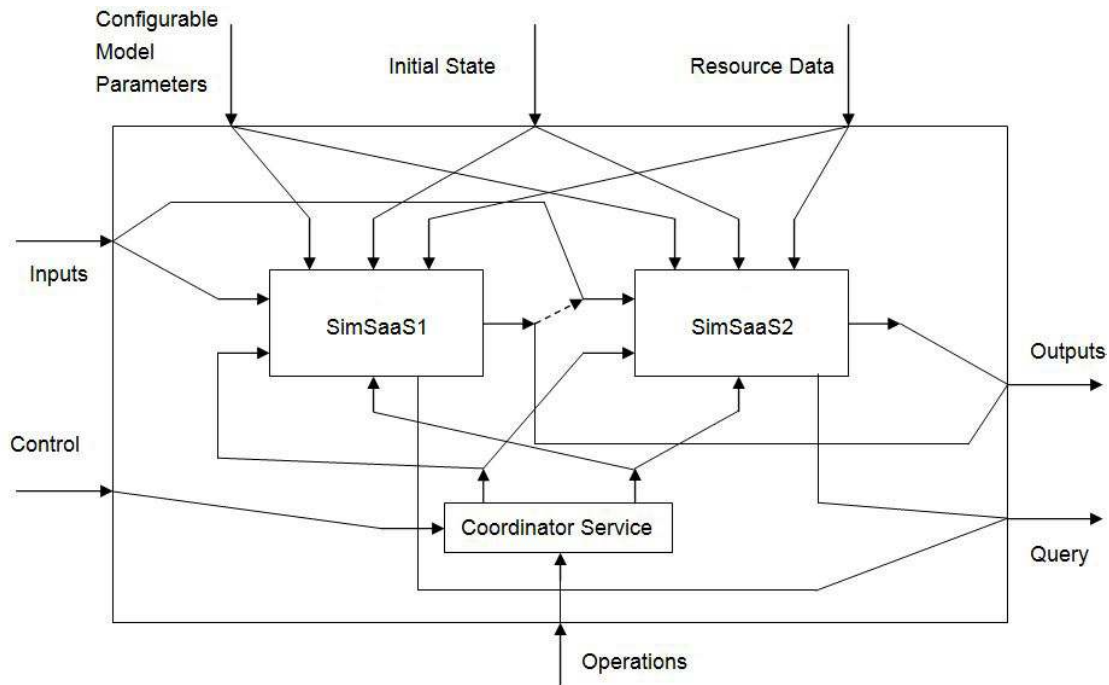


Figure 3.2 An example of composed simulation service

3.3.2 Specification for service-oriented simulation experiment

Usually, scientists and researchers want to do experiments which require several simulation components to work together to achieve a certain goal with the minimal human supervision. In the past, it requires scientists and researchers to have all the simulations on one machine. Extra programming duty on making different simulations working together is needed in order to design an experiment. Under the service-oriented environment, we define the experiment in SOA as follows.

$Experiment = \langle [SimSaaS], [Composed SimSaaS], [Other Service\&Tool], Experiment Workflow, Experiment Control \rangle,$

where

$[SimSaaS]$ is a set of simulation services that are involved in the experiment.

[*Composed SimSaaS*] is a set of composed simulation services that are involved in the experiment.

[*Other Service&Tool*] represents non simulation services and tools that are integrated into the experiment.

Experiment Workflow = <[*Workflow Coupling*], [*Input Trajectory*], [*Output*], *Workflow Specification*>. It defines the body of the experiment. [*Workflow Coupling*] is a set of couplings describing how the simulation services and/or composed simulation services are coupled in an experiment workflow. For a workflow coupling, it does not need a coordinator service since there is a workflow engine which will do the orchestration. [*Input trajectory*] is a set of input trajectories that are fed to the experiment; [*Output*] is a set of outputs that produced by the simulation services; *Workflow Specification* is the execution logic of the experiment. For example, Business Process Execution Language (BPEL), which is an OASIS standard executable language for specifying actions within business processes with web services, can be employed.

Experiment Control represents *Start*, *Stop*, *Pause*, *Continue* commands. It defines a group of commands that control the experiment. *Start* enables the experiment to begin; *Stop* enables the experiment to be terminated; *Pause* enables the experiment to be frozen; *Continue* enables the experiment to be resumed.

3.4 Service-oriented simulation experiment results collection

Experiment results collection is an important part in simulation software as a service (SimSaaS) and service-oriented simulation experiment (SOSE). Usually, experiment results are collected at certain phases of the experiment. The following lists three typical phases that an experiment results collection may be needed:

- ✧ Report all the results at the end of the entire experiment.
- ✧ Report results on a periodical basis.
- ✧ Report results if certain events happen.

Obtaining all the results at the end of the experiment is normal. However, there are times when the intermediate results need to be obtained as the experiment proceeds. If there are intermediate results produced or need to be collected, it is possible but inconvenient for the users to pause the simulation or the experiment and manually make the query. If the intermediate results are produced when certain events occur, it is impossible for the users to detect and make the query. To support both these two situations, we propose to implement an agent service which is responsible for the collection of the intermediate results based on different requirements. The detail of the agent service will be introduced in the next chapter.

CHAPTER 4 COMPOSABILITY OF SIMULATION SERVICES

To compose multiple simulation services is to make them work together and function as a single simulation service, which is called composed simulation service. The composed simulation services basically function with no difference as that of other single simulation services. The single simulation services being composed will communicate with each other in the realm of the software implementation of the models they represent, which is attributed to the issues of the interoperability; and behave in a composable way in the realm of the models themselves, which is attributed to the issues of composability. Although work has been done in both of these two aspects, there are still issues and challenges which are discussed in this chapter.

4.1 Interaction patterns

In a distributed computing environment, interactions are achieved via messages exchanges. In this section, some typical interaction patterns are summarized.

4.1.1 Request-response

The request-response pattern, illustrated in Figure 4.1, is the most common interaction pattern between two entities in the distributed computing environment. In this pattern, a requester sends a request message to a replier system which receives and processes the request, and returns the requestor a message in response. This pattern can be implemented either in a synchronous fashion, where the connection is held open until either the response is delivered or the timeout period expires; or in an asynchronous fashion, where the response will be returned at some unknown timer later.

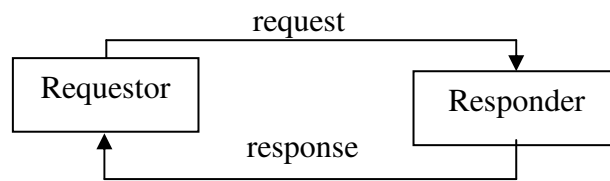


Figure 4.1 Request-response

4.1.2 Publish-subscribe

In this pattern as illustrated in Figure 4.2, the senders of the messages, called publisher, do not make the messages sent directly to any specific receivers, called subscribers. Published messages are categorized into topics (or classes). Publishers have no knowledge of what, if any, subscribers there may be. Subscribers attach their interests in one or more topics, and only receive messages that are of interest. Subscribers have no knowledge of what, if any, publishers there may be either.

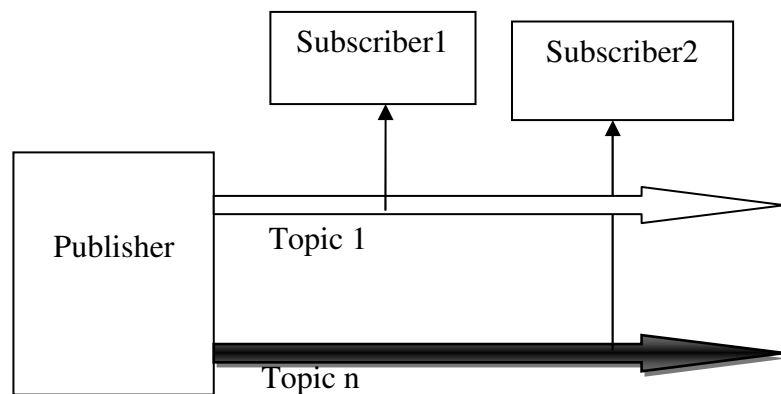


Figure 4.2 Publish-subscribe

4.1.3 Periodical synchronization

In this pattern illustrated in Figure 4.3, all of the participants need to be synchronized by exchanging outputs before proceeding to the next step of computation at a predefined time period, for example every 300 seconds.

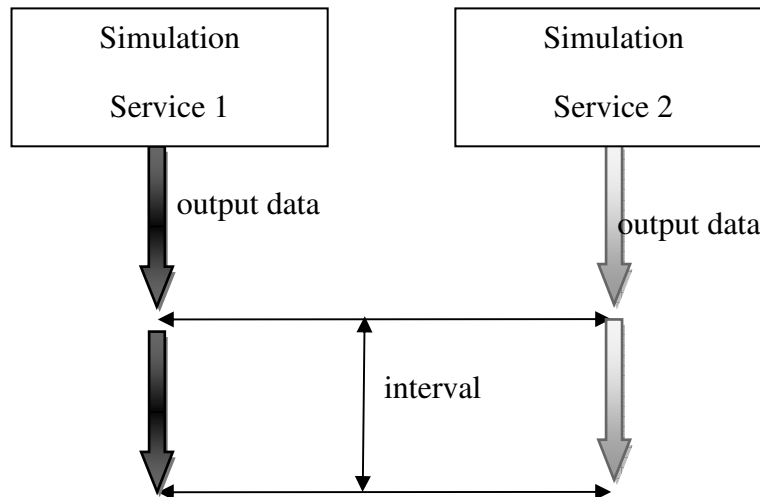


Figure 4.3 Periodical Synchronization

4.2 Coordinator service in simulation service composition

Simulation service is a dynamic system that evolves based on the time factor. Different simulation services in a composed simulation service need to conform to a common time system to guarantee the causality and correctness of the composed simulation service. That is events emerged in different simulation services in a composed simulation service should be prioritized and processed based on the temporal order. Another issue existing in composing simulation services is the complex interactions required by different simulation services to work together. Suppose we have two simulation services, each model of which has impact on the other. To achieve the

composition of these two simulation services and realize the mutual impact effect means that each of the participant simulation services needs to pass on certain produced data to the other simulation service from time to time as the simulation proceeds; and each simulation service will continue to execute based on the data received. However, it is unknown that when the data will be produced by each simulation service until runtime. Thus, to achieve simulation service composition, it is important and necessary to satisfy the temporal order requirement as well as to provide the ability to handle different interaction demands. From this point of view, none of the interaction patterns in section 4.1 can provide such properties and be naively employed to compose simulation services.

In section 3.3.1, a coordinator service is proposed in the composed simulation service specification in order to fulfill the above requirement on the temporal order in the distributed computing environment and the ability to handle different interaction demands. Based on the execution protocol, the coordinator service governs the time advancement of the all the involved simulation services on one hand; and on the other hand it routes the inside messages originated from the simulation services in this composed simulation service and the outside messages destined to this composed simulation service it is in charge of to the participant simulation services that construct this composed simulation service. Next, DEVS simulation protocol [54] is provided as an example coordinator execution protocol. The coordinator service would synchronize the activities of the simulation services to guide them through a cycle of method applications as follows:

1. `simulation_services.tellAll("setSimulationService", simulation_services);`
2. `simulation_services.tellAll("initialize");`

3. $N = \text{Min}(\text{simulation_services.AskALL}(\text{"nextTN"}));$
4. $\text{while}((tN < \text{INFINITY})){$
5. $\text{simulation_services.tellAll}(\text{"computeInputOutput"}, tN);$
6. $\text{simulation_services.tellAll}(\text{"sendMessages"});$
7. $\text{simulation_services.tellAll}(\text{"applyDeltFunc"}, tN);$
8. $tN = \text{Min}(\text{simulation_services.AskAll}(\text{"nextTN"}));$
- $}$

Each line is given the following explanation:

1. the coordinator service tells each of the simulation services in the collection the others' addresses.
2. the coordinator service asks each simulation service to perform the initialization function.
3. the coordinator service requests that each simulation service sends its time of next event and takes the minimum of the returned values to obtain the global time of next event.
4. the coordinator service enters the following cycle until activity ceases:
5. each of the simulation services applies its computeInputOutput method to produce an output that consists of a collection of contents(port/value) pairs – for DEVS simulation services this is a composite message computed according to the DEVS formalism based on its model's current state.
6. each of the simulation services partitions its output into messages intended for recipient simulation services and sends these messages to these recipient simulation services – for DEVS simulation services these recipients are

determined from the output ports in the message and the coupling information that will have previously been received from the coordinator service.

7. each of the simulators executes its ApplyDeltFunc method which computes the combined effect of the received messages and internal scheduling on its state, a side effect of which is produce of time of next event, tN – for DEVS simulators this state change is computed according to the DEVS formalism and the tN is updated using its model's time advance.
8. the coordinator obtains the next global time of next event and the cycle repeats.

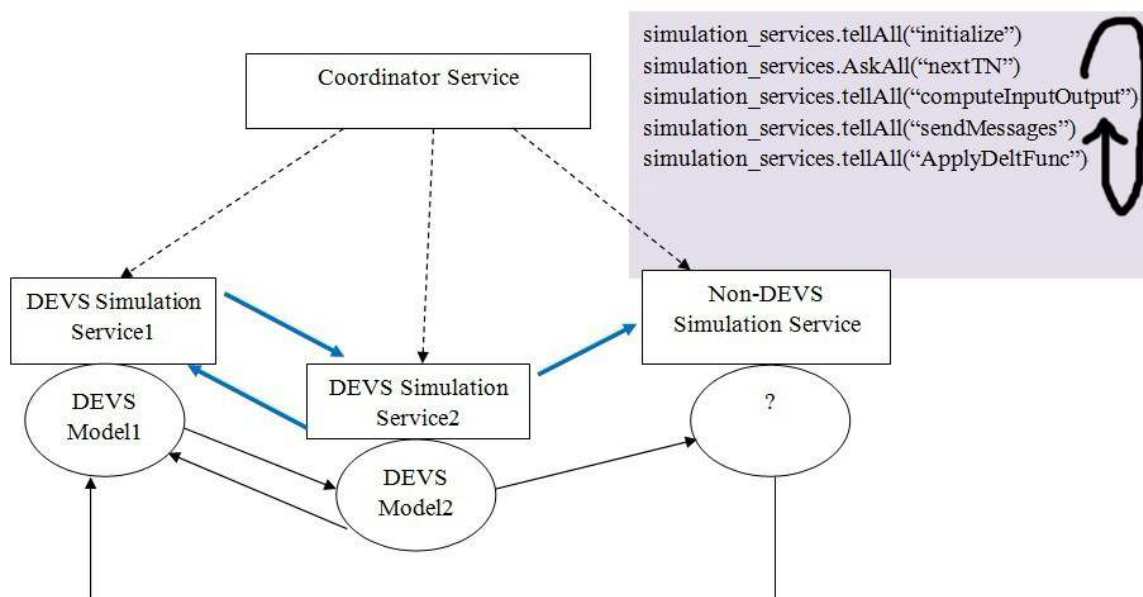


Figure 4.4 Federation of DEVS with Non-DEVS Simulation Service

4.3 Maturity reference model for simulation service composability

Simulation service composition should be considered one step further, beyond connections among different simulation services to just make them work together. Simulation service composability is about the degree of how simulation services can be

composed and how well they function as a whole to serve a common purpose. In other words, the composability should represent the maturity of the composed simulation services working together. In this paper, we adopt a maturity reference model for the simulation service composition that is adapted from the work in [5], [7], [59] and [61].

Zero composability: this level means the involved two simulation services cannot be composed at all. At this level, one can physically compose these two simulation services. However, the composition can crash the involved simulation services or runtime errors are reported.

Syntactic composability: this level means the involved two simulation services can be composed under the governing of common rules and simulation services can exchange common structured data. For example, the data that the weather simulation service produces contains two numbers: one is wind speed, and the other is wind direction. On the other hand, the wildfire simulation service needs data that consists of two numbers as its weather update input data.

Semantic composability: this level means that the involved two simulation services can be composed in a way that each party knows the meaning of the exchanged data. For example, the composition should guarantee that the exchanged data for the wildfire simulation service are two numbers which represent the value of wind speed and the value of wind direction in the correct order with correct range.

Pragmatic composability: this level means that the involved two simulation services can be composed in a way that each party behaves in a shared context. In other words, each party behaves the way as others expect. Building upon the previous level, this level

emphasizes on the rationality of the composition and the context of the simulation service behavior.

While there are research issues at each level, this section focuses on the issue of mismatch of time/event granularity at the pragmatic composability level.

4.4 Time granularity and event granularity in simulation service composition

Like building a complex system from multiple sub-systems, it is common practice to build a simulation experiment from multiple disparate individual simulations or simulation software. As the service-oriented architecture gains its broad acceptance and is made as a standard, the simulation experiment can now be built by employing a wide range of the available simulation services. When composing heterogeneous simulation services, issues arise such as whether the two simulation services can be composed together, or whether the composition is addressed in a meaningful and rational way. It is pointed in [55] that some composite models involve components that are naturally expressed in very different representations and formalisms because the phenomena are different in character. For example, missile trajectories are best represented by continuous differential equations, whereas force-on-force ground battles lend themselves well to discrete-event simulation or time-stepped simulation with large time steps. There are also differences in granularity, i.e., differences in number and kinds of aspects. This need for heterogeneity is not merely an artifact of the mathematics or programming. In [55], the authors also demonstrated several examples which indicate that the heterogeneity among the composites is a universal issue which poses great impacts on the compositions. Another obstacle standing in the way of composition of simulation services is that the simulation services themselves are not created and owned by the users in

the service-oriented architecture, which makes it impossible for the users to modify the simulation services according to their specific needs. As Splash (the Smarter Planet Platform for Analysis and Simulation of Health) from IBM [56] states: “experts in different organizations drawing on knowledge and techniques of such complex systems. It is a significant challenge to pull these sorts of models together into an integrated picture.” Thus, the compositions of the simulation services can be handicapped by the above facts. Authors in [57] and [58] proposed frameworks that employ agents as mediators, brokers and matchmakers between disparate composites, to tackle the heterogeneity existing in the compositions.

In a service-oriented computing environment, there is a need to compose several simulation services in a way that the multiple simulation services are coupled together and dynamically influence each other. While simulation service composition shares many common properties of general web service composition, it has some unique features due to the fact that simulation services rely on dynamic models where time is important. This means individual simulation services in the composition must behave according to a common time base in order to ensure the correct temporal order of events. Research of simulation composition has often been studied together with the topic of modeling and simulation interoperability. In particular, to help studying different issues arising in simulation interoperability, the Levels of Conceptual Interoperability Model (LCIM) were introduced, which identified seven levels of interoperability among participating systems [5]. A reformulation of LCIM was presented in [59, 7], which summarizes three linguistic levels of interoperability: Syntactic level, Semantic level, and Pragmatic level. In this chapter, a maturity reference model is adapted for simulation service composition

based on [59, 7], and focus on the pragmatic level which emphasizes on the rationality of the composition and the context of the simulation service behavior.

At the pragmatic level, an important issue that hinders the composition of simulation services is the mismatch of granularity when coupling two services together. In this chapter, two types of granularity, named as time granularity and event granularity are differentiated. The issue of time granularity occurs when there is mismatch between the sender and receiver regarding how often the information is generated and consumed. For example, a weather simulation model may generate weather information in hourly or daily basis. A climate simulation model, however, typically consumes weather change information in seasonal or yearly basis. This mismatch of time granularity makes it ineffective to directly couple the wildfire simulation with the climate simulation. The issue of event granularity occurs when there is mismatch between the sender and receiver regarding the significance of events contained in the output/input message. For example, while a factory simulation may send out a message whenever a new product item is finished, a supply chain management simulation may be only interested in situations when the product quantity changes significantly (e.g., increase or decrease by hundreds). This type of mismatch of event granularity makes it ineffective to directly couple the two simulations together. It is noted that the issues of time granularity and event granularity are independent of the simulation mechanisms (i.e., discrete time simulation or discrete event simulation) of the simulation services.

This section focuses on time granularity and event granularity in composing simulation services at the pragmatic level. The goal is to develop a systematic way to support simulation service composition with mismatch of time/event granularity.

To achieve this goal, the semantics of couplings between simulation services are enhanced with the capability of handling time/event granularity. Formal models for handling time/event granularity are defined and associated with the couplings in simulation service composition. From the operational point of view, this work is similar to previous work on agent-based framework for supporting model composition [57, 58]. However, this work focuses on the specific issue of time/event granularity and “agents” with well-defined behavior for handling time/event granularity are developed. The developed agents can be reused by others in a “plug-and-play” fashion based on users’ needs. This work is considered as an important step towards achieving standard and automated approaches for simulation service composition in the emerging service-oriented computing environment. Note that in this work, we make the assumption that all the involved simulations services are created and owned by different parties, and it is difficult or inconvenient to make direct changes to the simulation service themselves.

This work is motivated by our research on coupled weather-wildfire simulation, where a weather simulation and a wildfire simulation are coupled together in order to achieve more accurate wildfire spread simulation results. The wildfire simulation service needs weather data, such as wind speed and wind direction, as one of its inputs, and it produces heat as one of its outputs. The weather simulation service produces weather data such as wind speed and wind direction, to be fed into the wildfire simulation, and it takes heat generated from the fire simulation as an input. Figure 4.5 illustrates the composition.

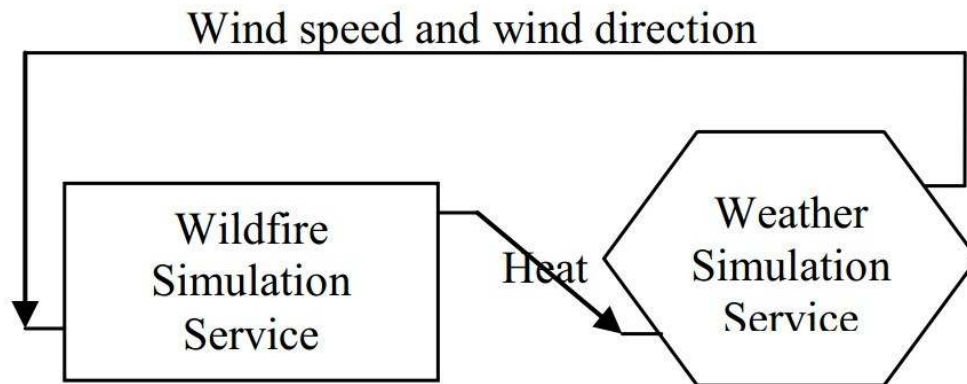


Figure 4.5 Composition of the wildfire simulation service and the weather simulation service

The composition as shown in Figure 1 works well at the syntactic level and semantic level, that is, the data format of the exchanged data is commonly supported and the meaning of the exchanged data is understood by both simulation services. However, at the pragmatic level there is mismatch of time/event granularity. Specifically, in our research the weather model is the Advanced Regional Prediction System (ARPS) model [60, 61], which is a three-dimensional, nonhydrostatic numerical weather prediction model. The ARPS model generates weather output typically at the time scale of second. Such frequent weather updates, however, do not make much sense for the wildfire model (the DEVS-FIRE model [62]), which typically works with weather data at time scale of tens of minutes (e.g., 10 minutes, 30 minutes), due to the nature of the wildfire spread behavior. The DEVS-FIRE model will be introduced in the later chapter.

4.4.1 Time granularity

Simulation services are supported by simulation systems which are dynamic systems whose evolvment is based on the time which has units also called granularities. Both the

outputs and the inputs of the dynamic system relate to the time. The system may generate outputs at certain time steps; and in turn the system may be given inputs feed at certain time steps. The term time granularity is not new and has been studied in [63] and [64] in that system support and reasoning different granularities has been recognized to be an import issue. The time granularity of the dynamic system can either be fixed or it can be driven by events, where a time step exists when something interesting occurs. For example, the weather simulation service is a dynamic system and it generates wind speed and wind direction outputs every 10 seconds. Then the time granularity for the wind speed and wind direction output of the weather simulation service is 10 seconds. When two disparate systems from different backgrounds are composed together by connecting the outputs of one system to the inputs of the other system, the issue of time granularity may occur when there is mismatch between the sender and receiver regarding how often the information is generated and consumed. The mismatch can happen in both directions, including from the output in a coarse-time-granularity to an input requiring a fine-time-granularity, and from the output in a fine-time-granularity to an input requiring a coarse-time-granularity. For example, the wildfire simulation service needs wind speed and wind direction as one of its input every 200 seconds. If we couple the previous weather simulation service with this wildfire simulation service, a mismatch from fine-time-granularity to coarse-time-granularity occurs.

To enable the compositions that involve simulation services with different time granularities, a time granularity handling agent is proposed to be added to act as a mediator between the two services. The time granularity agents will adjust the time

granularity from the source simulation services and generate outputs to satisfy the input time granularity requirement of the destination simulation services.

4.4.2 Event granularity

Compared with time granularity which deals with time, event granularity deals with the effect of the events, which can be represented by the degree of the changes in the observed subjects. In the composition of simulation services, the issue of event granularity may occur when there is mismatch between the sender and receiver regarding the significance of events contained in the output/input message. Similar as in time granularity, the mismatch can happen in two directions, including from coarse-event-granularity to fine-event-granularity and from fine-event-granularity to coarse-event-granularity. In the factory-supply chain example mentioned above, the coupling from the factory simulation service to the product item yield input of the supply chain simulation service is an example of fine-event-granularity to coarse-event-granularity mismatch. For the other situation, from coarse-event-granularity to fine-event-granularity, it might have the potential to cause problems if the event is directly conveyed. However, that is the best we can do in the scope of this work.

To enable compositions that involve simulation services with different event granularities, an event granularity handling agent is proposed to be added to act as a mediator between the two services. The proposed event granularity handling agents employ quantization, which is an approach in which attribute updates are sent only when threshold boundaries are crossed, rather than continuously as they are generated [65].

4.4.3 Value message and delta message in granularity handling

The introduction of granularity handling agents in simulation service composition makes it necessary to differentiate two types messages passed between services. In this paper, we differentiate two types of messages: delta message and value message. A delta message means the data contained in the message represents a change; a value message means the data contained in the message represents an “absolute” value. The reason that we need to differentiate two types of messages is because after adding the granularity handling agents, multiple input messages may need to be combined by the agent and sent out as a single output message (in the fine granularity to coarse granularity case), or a previous input message needs to be reused by the agent to process the messages in a correct way, there is a need to look at the types of the data contained in the messages and act accordingly.

The granularity handling agents will act in different ways when processing the two types of the messages. In general, when handling delta messages, the agent needs to add up all the input messages and sends out the sum as the output. This is because each input message represents a change from the previous value, and thus multiple inputs need to be summed together to represent the total change. If there is no input message received, the output message should contain the value of 0. When handling value message, the agent needs to send out the most recent input message as the output. This is because the most recent input message contains the most “current” value. When there is no recent input, the output message should contain the input received before. For example, a simulation service A produces an output in delta message, and another simulation service B accepts A’s output as input. A produces the output every 10 seconds, and B requires to receive

the input every 100 seconds. We can add a time granularity agent in the composition of the two simulation services with time granularity set to 100s. Thus, the output from A will be given to the time granularity agent first and only when the simulation time is the multiple of the time granularity 100s, such as 100s, 200s, 300s..., can A's the output be fed to B through the time granularity agent. To ensure the correctness after we add the time granularity handling agent, all the data received from A within each time granularity duration (in this example 100 seconds) should be summed by the agent before fed to B, since the data in the delta message are the change part of the value. If the messages exchanged between A and B are value messages, the granularity handling agent behaves differently: the latest data contained in the value message received from A within each time granularity duration should be used to feed B. The problems described above also happen where an event granularity handling agent is needed. In order to process the messages in correct ways, the agents will need to behave differently for the four cases, which will be explained further in the next section.

4.4.4 Granularity handling agents

Based on the above discussions, we identify and develop four granularity handling agents to solve the mismatch problem in the output-input coupling pair. They are time granularity agent for value message, time granularity agent for delta message, event granularity agent for value message, and event granularity agent for delta message.

Time granularity agent for value message will update its current value to the data value contained in the message each time it receives. When it is time for this agent to generate outputs at the time steps $T, 2T, 3T, \dots, nT$ (T represents time granularity), the current value will be sent out. Figure 4.6 displays the state chart diagram of this agent.

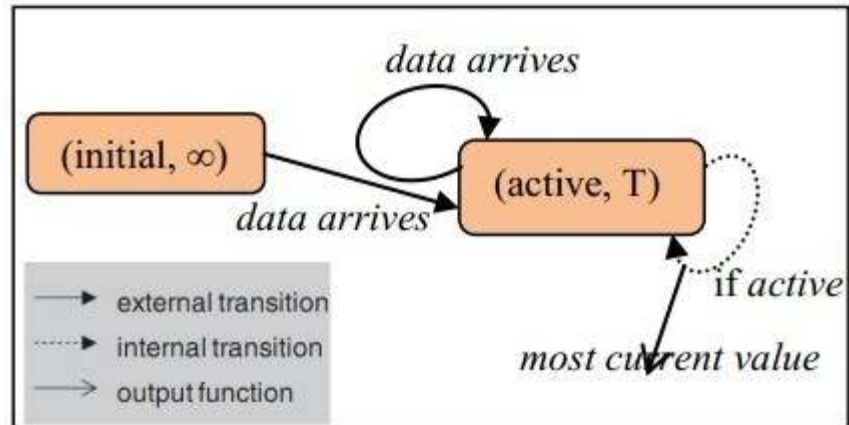


Figure 4.6 The state chart diagram for time granularity agent for value message

Time granularity agent for delta message will initially set its current value to zero. It accumulates the received data value to its current value. When it is time for this agent to generate output at the time steps $T, 2T, 3T, \dots, nT$ (T represents time granularity), the current value will be sent out and then the current value will be set to zero again.

Figure 4.7 displays the state chart diagram of this agent.

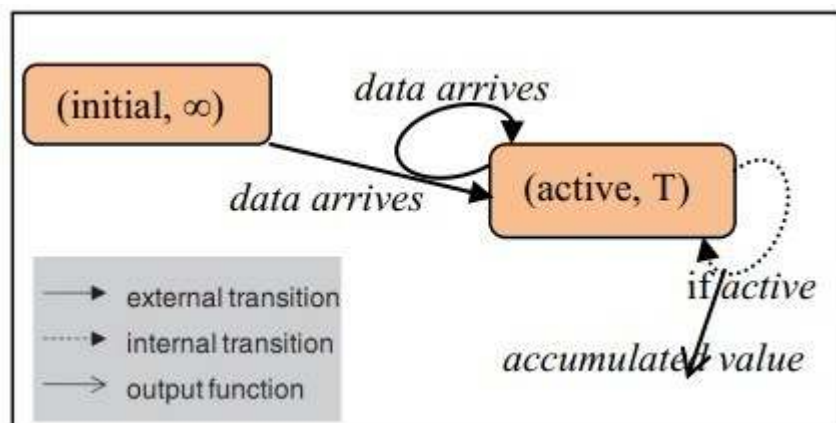


Figure 4.7 The state chart diagram for time granularity agent for delta message

Event granularity agent for value message will compare the difference of the received data value and the current value with a threshold. If the difference is equal or larger than the threshold, the current value is set to the received value and the current value is used as the output. Figure 4.8 displays the state chart diagram of this agent.

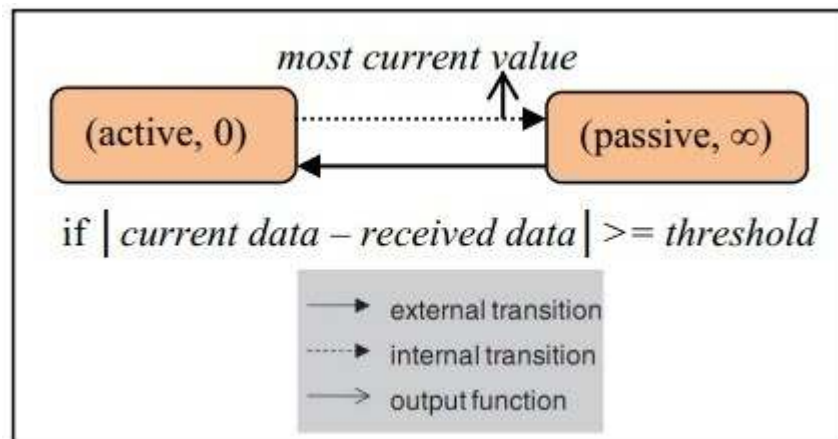


Figure 4.8 The state chart diagram for event granularity agent for value message

Event granularity agent for delta message will initially set its current value to zero. It accumulates the received data value to its current value. Once the current value is equal or larger than the threshold, the current value is sent out and then set to zero. Figure 4.9 displays the state chart diagram of this agent.

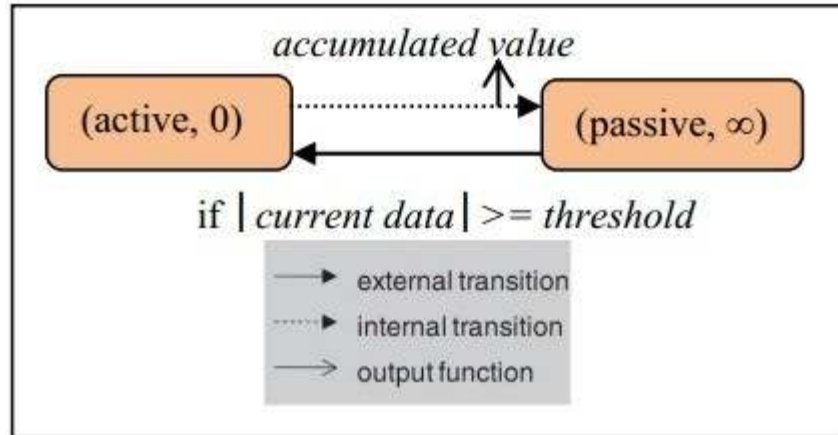


Figure 4.9 The state chart diagram for event granularity agent for delta message

4.4.5 Granularity-enhanced coupling in simulation service composition

To achieve systematic handling of time/event granularity, we enhance the semantics of couplings between simulation services with the capability of handling time/event granularity. If needed, a coupling (an output-input pair) can be associated with an agent based on user's need (no agent will be associated with the coupling if the mismatch does not exist). Different couplings can have different agents. We note that the agent is associated with the couplings, thus multiple can be used if there are multiple couplings between two agents. This is different from [57, 58] where typically a single agent is used between two simulation services. With the proposed granularity handling agents, the composition of the simulation service can be enhanced by using these agents in the output-input pair couplings in a “plug-and-play” style. It is noted that some couplings require these agents to work better, while others do not, as illustrated in Figure 4.10.

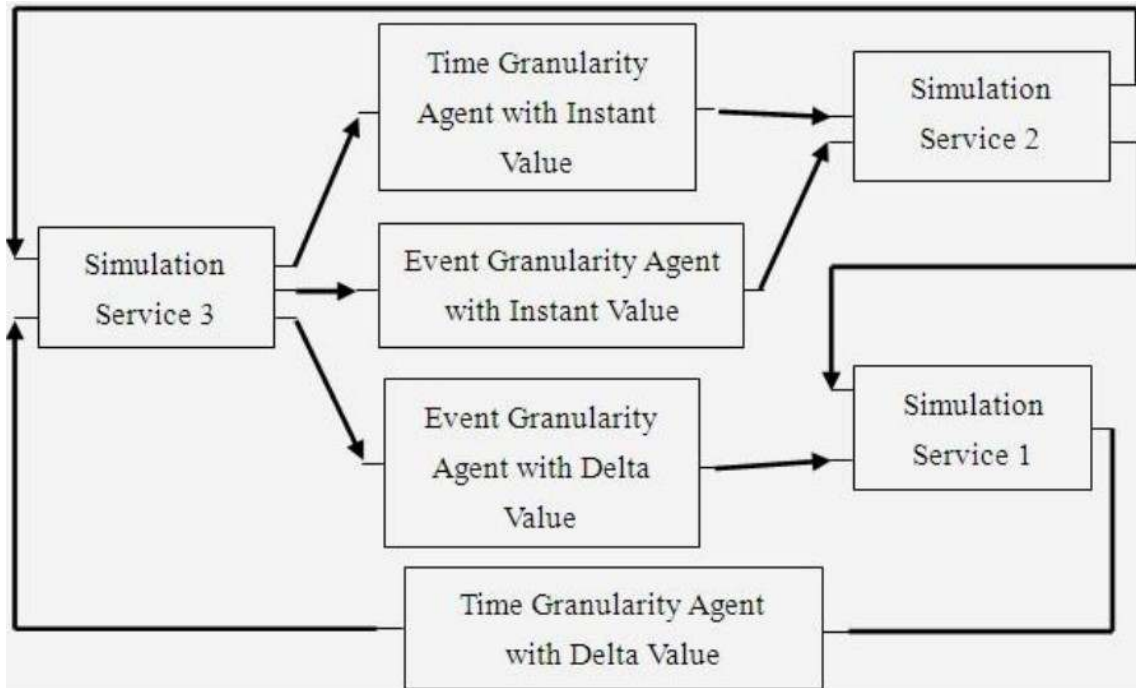


Figure 4.10 Granularity handling agents in the couplings of the composition

4.4.6 Experiment results

In our coupled weather-wildfire simulation research, the weather model is the ARPS model and the wildfire model is DEVS-FIRE model. However, we use a simplified “artificial” weather model developed for illustrating the developed methods. The artificial weather model is not based on any real knowledge in atmospheric dynamics and is not validated. We assume that the wind speed and wind direction are affected by the generated heat following the equations below. In our experiment, the weather simulation service generates weather condition, wind speed and wind direction, in instant value form every second, and the wildfire simulation service produces heat in delta value type every one hundred seconds.

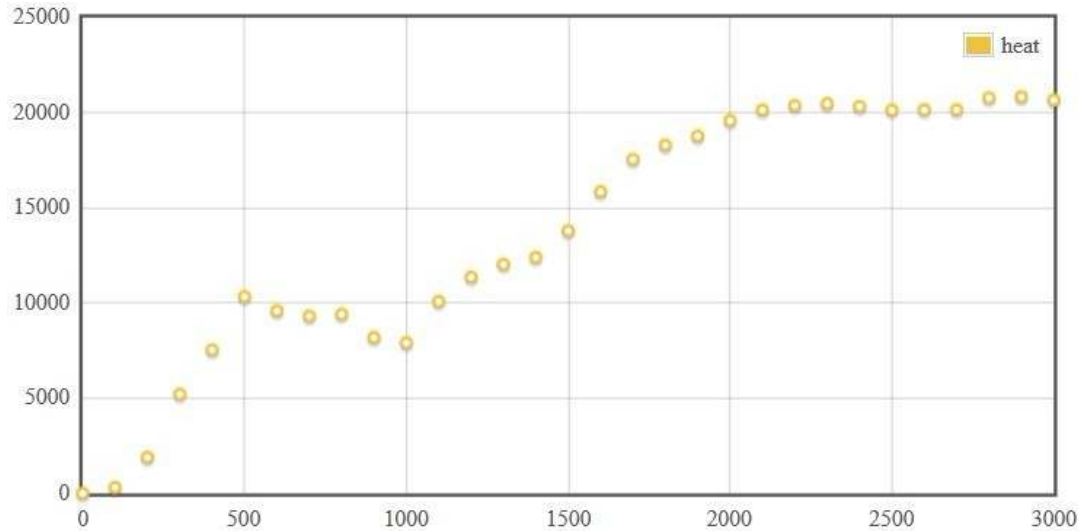
$$\frac{dW_{speed}}{dt} = \frac{\frac{d\Delta H}{dt} - \alpha}{1000} + \text{Gaussian noise}$$

$$\frac{dW_{direction}}{dt} = \frac{\frac{d\Delta H}{dt} \cdot \text{Gaussian Noise}}{200}$$

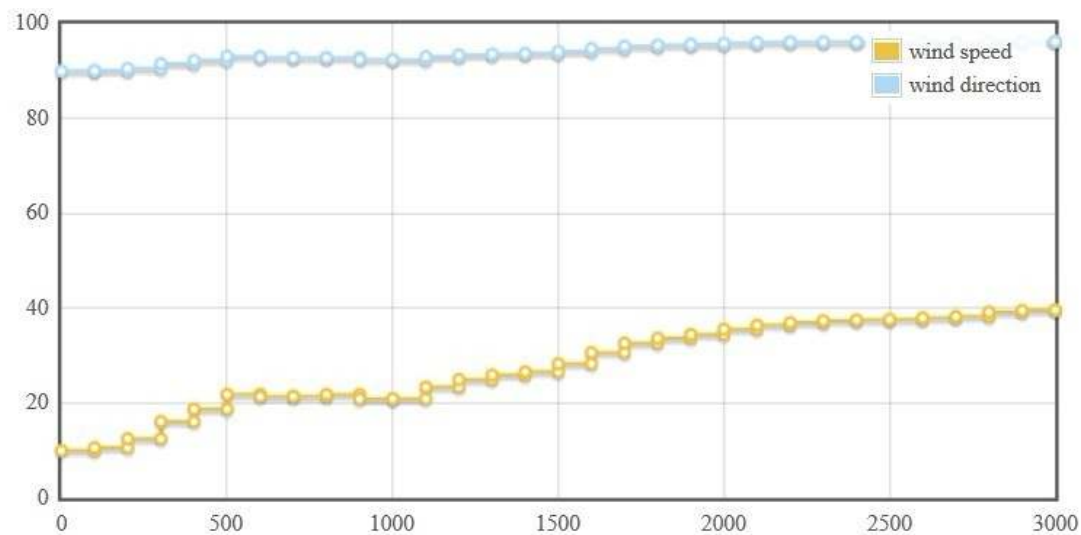
In the experiment, we use a time granularity agent for value message between the weather condition output-input coupling pair from the weather simulation service to the wildfire simulation service; and an event granularity agent for delta message between the heat output-input coupling pair from the wildfire simulation service to the weather simulation service. We set the initial wind speed to 10 mile/second, and wind direction to 90 degree. α is set to 2.0. The output of the time granularity agent in this experiment uses the most-recently-received data. Figure 4.11 displays the experiment compositions without configuring the proposed properties and its corresponding experiment results. Figure 4.12 displays the experiment compositions with time granularity set to 50 seconds and threshold set to 1000 and its corresponding experiment results. To make the comparison easy, the Gaussian Noise is fixed.



(a) The result fire shape at 3000 seconds in simulation time



(b) Heat outputs from the wildfire simulation service

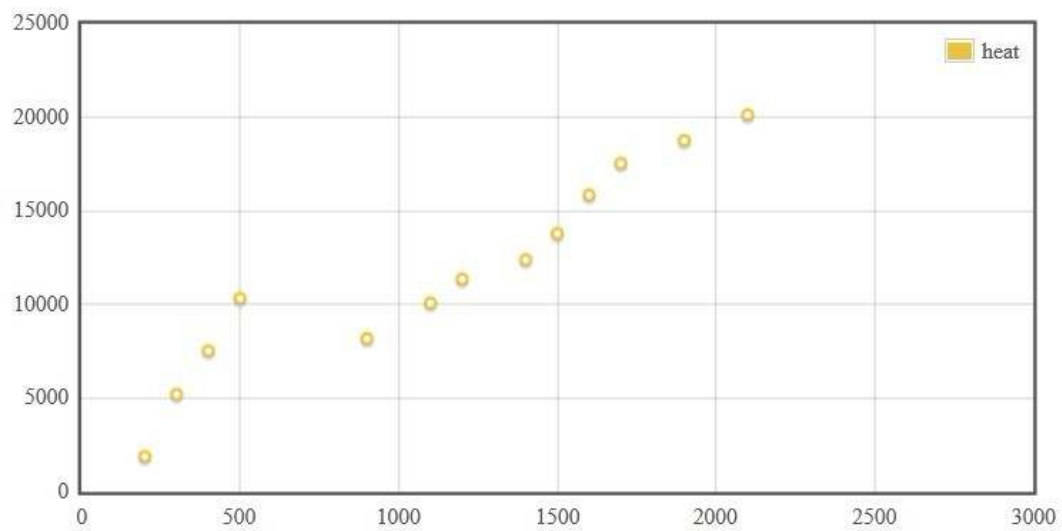


(c) Wind speed and wind direction from the weather simulation service

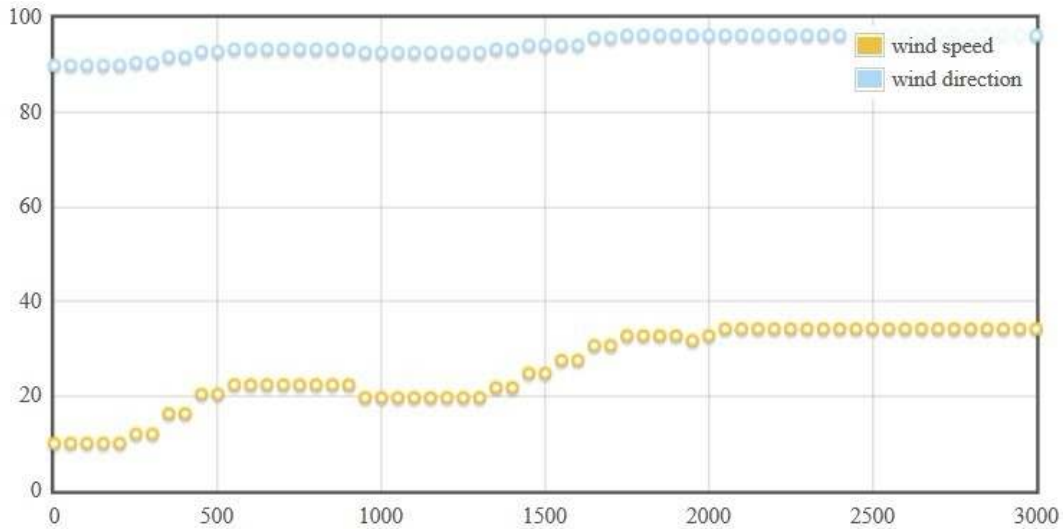
Figure 4.11 Simulation results of composition of the wildfire simulation service and the weather simulation service not using granularity handling agents.



(a) The result fire shape at 3000 seconds in simulation time



(b) Heat outputs from the wildfire simulation service



(c) Wind speed and wind direction from the weather simulation service

Figure 4.12 Simulation results of composition of the wildfire simulation service and the weather simulation service using granularity handling agents.

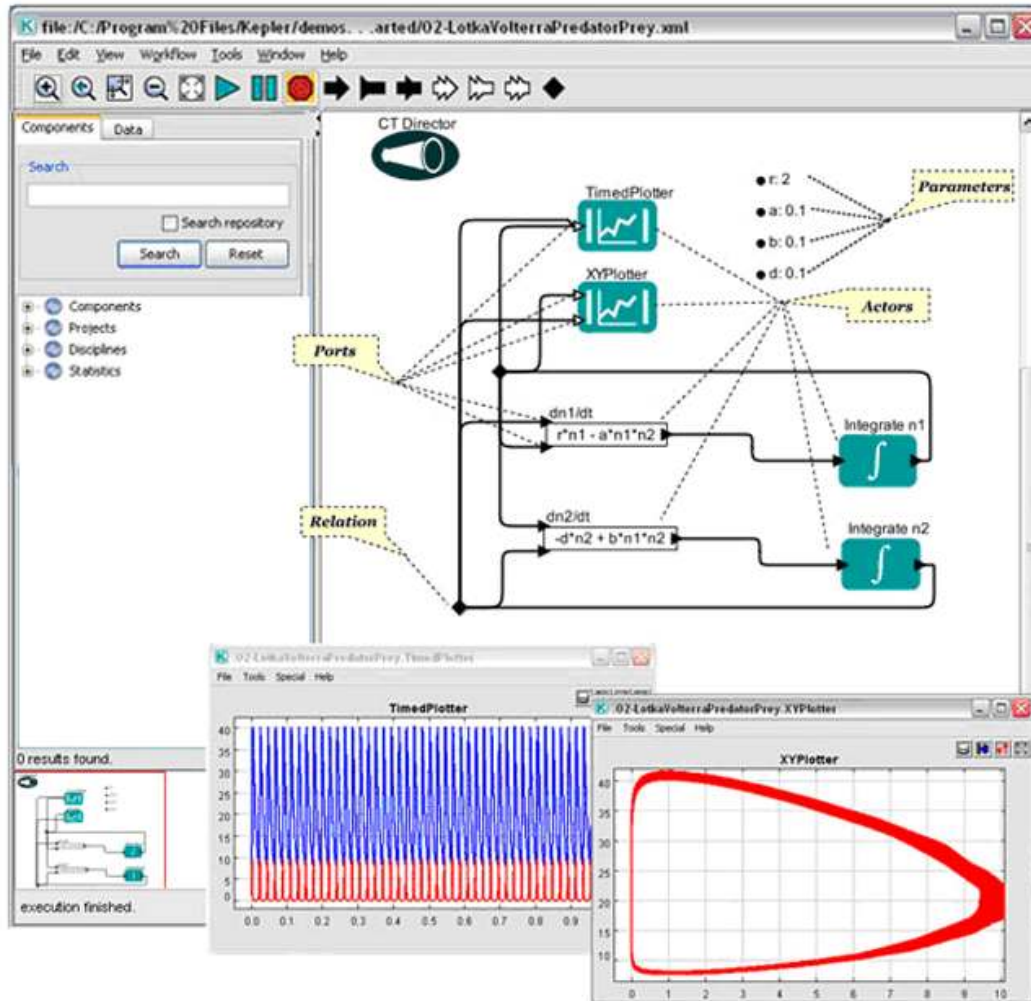
By setting the properties, Figure 4.12 (b) and (c) illustrate that the proposed pragmatic simulation composability is achieved. The cost/benefit analysis between reduced data exchanges and increased error was discussed in [65]. Although the fire shape after adding the time granularity handling agent and the event granularity handling agent is different from the one without agents, the fire spread trend still follows the original track. If the error is acceptable, it seems there is no harm using the granularity handling agents.

CHAPTER 5 SERVICE-ORIENTED SIMULATION EXPERIMENT

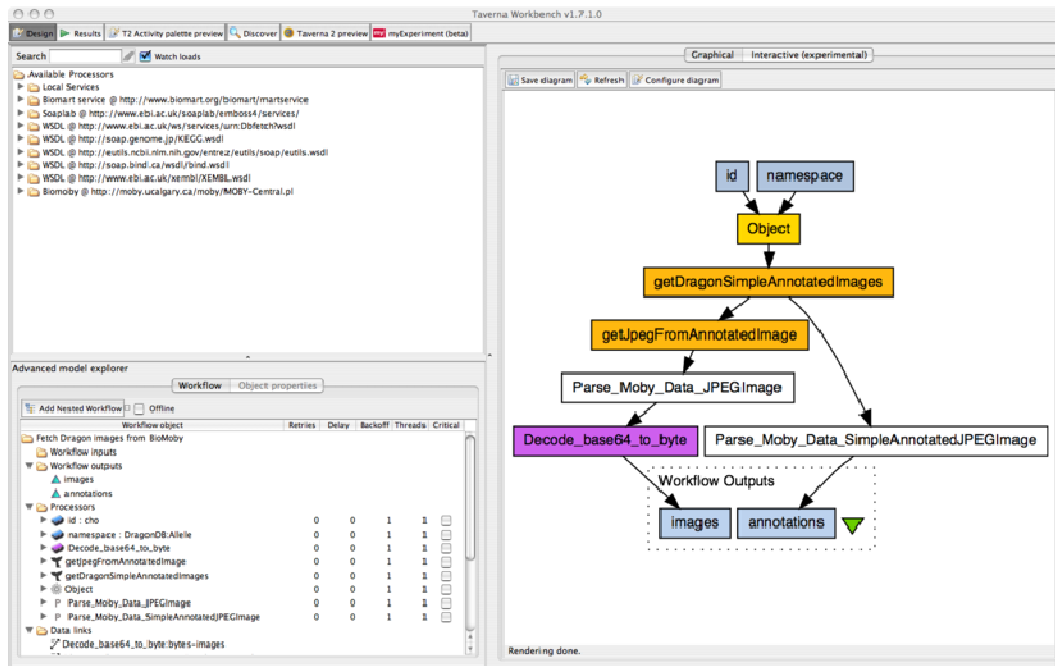
5.1 Existing workflow systems

Workflow is a depiction of a sequence of operations. A workflow management system is a computer system that manages and defines a series of tasks within an organization to produce a final outcome or outcomes [94]. For different types of jobs or processes, the workflow management system allows the user to define different workflows. In the organization of the workflow, each individual unit is responsible for a specific task. Once the task is complete, the workflow system ensures that the individual units responsible for the next task are notified and receive the data they need to execute their phase of the process. Workflow management systems also automate redundant tasks and ensure that uncompleted tasks are followed up.

A scientific workflow system is a specialized form of workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or a workflow [95]. In Figure 5.1, it illustrates two scientific workflow systems: Kepler scientific workflow system and Taverna workbench.



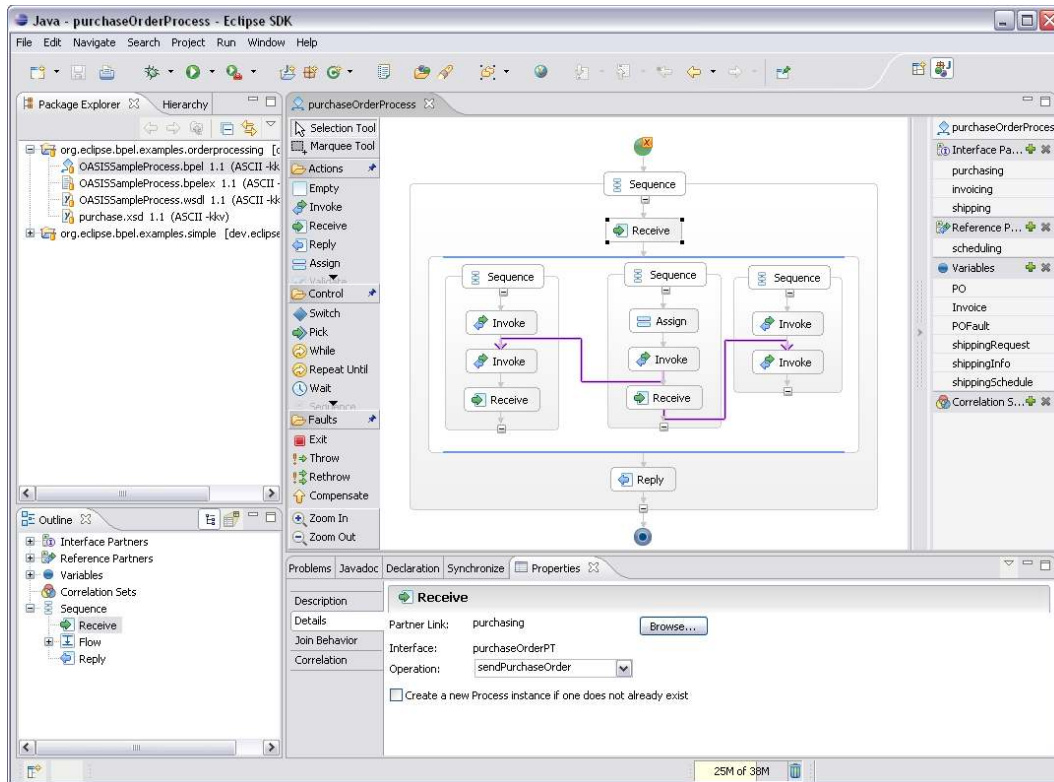
(a) Kepler scientific workflow system



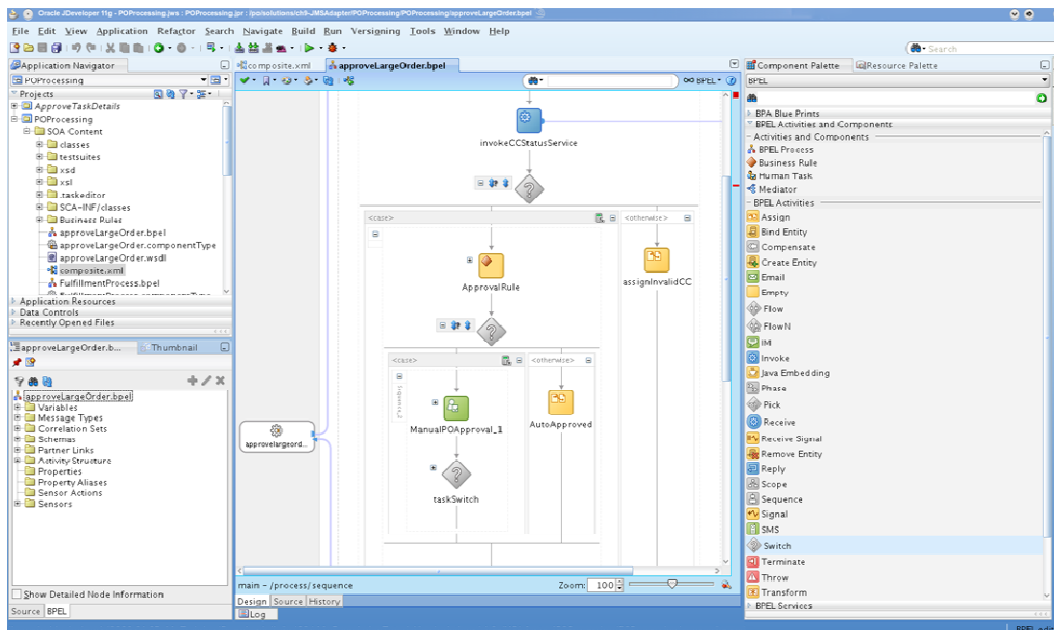
(b) Taverna workbench

Figure 5.1 Major scientific workflow systems

In the business world, BPEL (Business Process Execution Language) is an OASIS [96] standard executable language for specifying actions within business processes with web services [10]. By using this language together with the execution engine, the user can organize a batch of tasks in a workflow fashion. Figure 5.2 displays two BPEL development environments: eclipse BPEL designer and oracle BPEL process manager.



(a) Eclipse BPEL designer



(b) Oracle BPEL designer in JDeveloper

Figure 5.2 BPEL development environments.

From Figure 5.1 and Figure 5.2 we can see that all those workflow systems are equipped with a lot of components, which make them powerful but complex at the same time. Usually, scientists and researchers are technical limited and it would be a waste of time for them to learn those new complex technology tools just to carry out their experiments. To use these workflow systems to design a workflow, scientists and researchers not only need to learn how to use different components, but also need to understand the meaning of each activity, population of the parameters and configuration of each binding. Another disadvantage of these workflow systems is that they are desktop-based, and users need to obtain a copy of the software which may require complicated installation and configuration. This cannot catch up with the notion “thin client” in the cloud computing environment. In order to alleviate the burden of learning complex and unnecessary tools from scientists and researchers, we propose to develop a web-based easy-to-use light-weight service-oriented simulation experiment environment.

5.2 The Proposed Service-Oriented Simulation Experiment

5.2.1 Composed simulation service and simulation experiment

With the advent of cloud computing and service-oriented architecture, it is highly desirable that the simulation experiment can be carried out by taking advantages of those two aspects. A simulation experiment can be viewed as a workflow, and it consists of one or multiple organized components, which in our case are single simulation services, composed simulation services and/or other tool services, in a sequence of connected steps. Firstly, we need to differentiate the connections (one’s output fed to the other’s input) in the composed simulation service from the connections in the simulation service experiment. In a composed simulation service, when coupling two simulation services, it

means that the involved two simulation services are governed by a coordinator whose job is to schedule all the events in the services governed in the correct temporal order to avoid any causality. The composed simulation services can be integrated into the experiment with no difference compared to other simulation services and non-simulation services. In a simulation experiment, there is no such coherent relation between connected simulation services and/or non-simulation services, which are organized in a workflow and invoked based on the execution logic of the workflow. For example, the wildfire simulation service and the weather simulation service work closely in a composed simulation service in that these two simulations have mutual influence on a common temporal order basis, which requires to be governed by a coordinator service. However, the wildfire simulation service and the optimization service in a simulation experiment work in a sequential workflow order. That is only when the wildfire simulation service finishes and produces a result, the result is passed on to the optimization service, and the optimization service will start to execute based on the result from the wildfire simulation service.

In the service-oriented architecture, simulation services and non-simulation services are hosted completely distributed. From section 5.1 we can conclude that two elements are needed in order to design and execute a workflow: the representation of the workflow and the execution engine of the workflow. The representation of the workflow provides a formal way to describe the structure of the workflow. The execution engine of the workflow executes the workflow based on the structure. In the above example, Kepler and Taverna each has its own representation to describe the workflow in its own system. Their representations will be translated into executable instructions by their execution

engine respectively. On the other hand, BPEL is a formal orchestration language which provides a standardized programming model, by following which different execution engine can be achieved. In this work, we propose to provide a centralized web-based simulation experiment system for the users to design and execute their simulation experiments. For different users, we employ two approaches in our service-oriented simulation experiment system: web browser-based workflow execution and BPEL-based execution.

5.2.2 Web browser-based execution approach for service-oriented simulation experiment

The goal of this approach is to provide a simple way for the users to design their simulation experiments. In this approach, we provide a small set of work units, which represent different semantics, and a browser-based execution engine that executes the workflow constructed by employing and structuring multiple work units. To design a simulation experiment, the user can create a task, add work units to the task and organize those work units in a logical order based on the semantic of each work unit. In this work, we achieve this approach using javascript, which can be executed in a web browser, to implement the work units, task and a small execution engine that can run an experiment designed by the provided work units. Figure 5.3 illustrates the architecture of this approach.

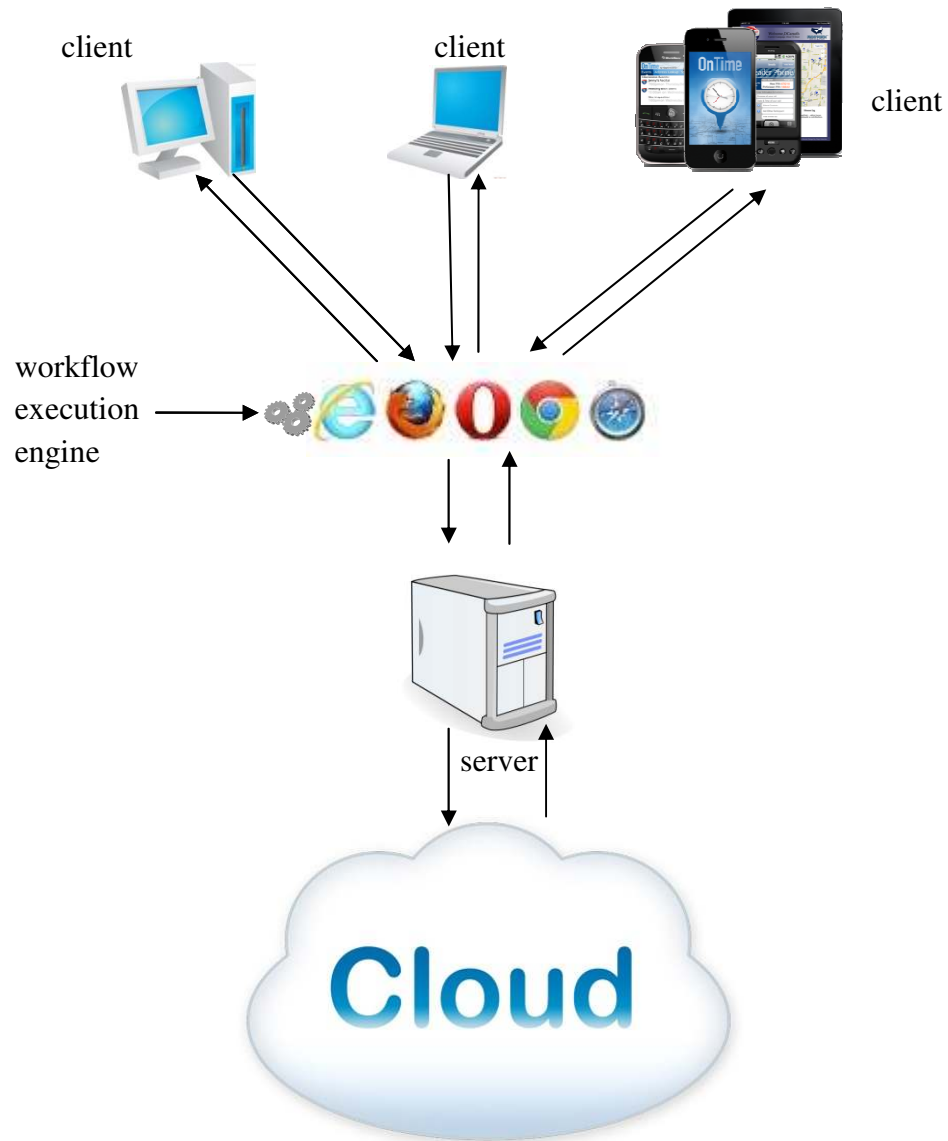


Figure 5.3 Architecture of the web browser-based execution approach

Next, some terms along with the provided set of work units are introduced.

A *task* is a group of work units organized together to form a process, which represents an experiment. All the work units within the task will be executed sequentially based on the created order.

A *work unit* is an atomic unit that carries out a specified action based on its semantic. In this work, we propose to provide four work units: *start work unit*, *end work unit*, *assign work unit*, *invoke work unit*, *while work unit*, *wait work unit*, and *if-else work unit*.

A *start work unit* implements the action of being the start point of the whole task. It stands as a dummy indicator of the beginning of the whole task and does nothing else.

An *end work unit* implements the action of being the end point of the whole task. It stands as a dummy indicator of the end of the whole task and does nothing else.

An *assign work unit* implements the action of assign a value to a variable within the experiment workflow. There are three fields to configure an assign work unit: *variable name*, *variable type*, and *variable value*. If the variable doesn't exist, it will be created using *variable name* and its value will be *variable value*. If the variable already exists, its value will be updated by *variable value*. In this work, we support three variable types: number, string and bool.

An *invoke work unit* implements the action of invoking an operation provided by a simulation service with specified input parameter(s). The result from the invocation of the operation, if there is any, will be stored by this invoke work unit. Figure 5.4 illustrates the semantic of this work unit.

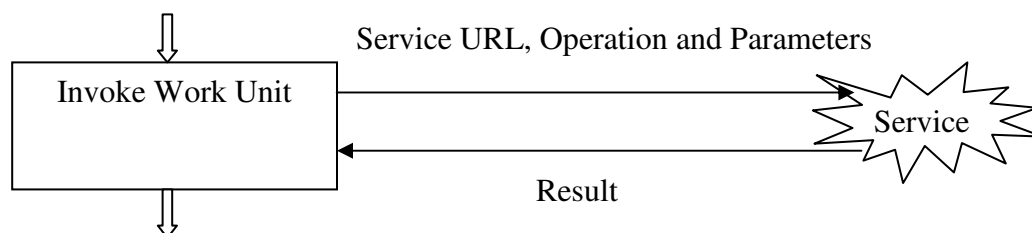


Figure 5.4 Invoke work unit

A *while work unit* implements the action of a while loop. This work unit evaluates the provided expression to a boolean value. When the process of the experiment encounters at this work unit, it will not go any further until the expression is evaluated as false and the included work units of this while work unit will be executed at each iteration of the loop. Figure 5.5 illustrates the semantic of this work unit.

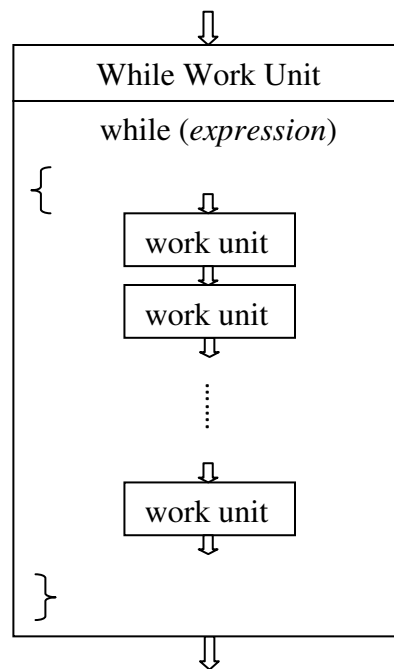


Figure 5.5 While work unit

A *wait work unit* implements the action of suspending or pausing the process of the experiment for specified time period, whose unit is second. Figure 5.6 illustrates the semantic of this work unit.

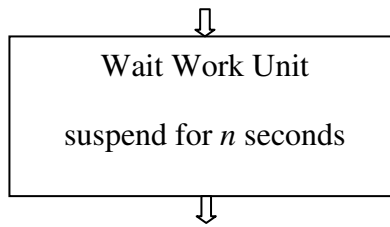


Figure 5.6 Wait work unit

An *if-else work unit* implements the action of conditionally branching of the process. It evaluates the provided expression to a boolean value. If the expression is evaluated as true, the process will continue to execute in one branch; otherwise it will continue to execute in the other specified branch. Figure 5.7 illustrates the semantic of this work unit.

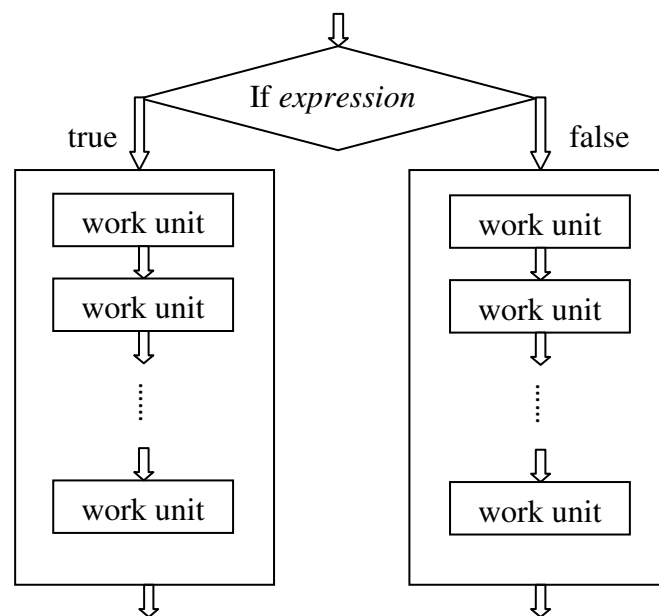


Figure 5.7 if-else work unit

The advantage of this approach is that the user does not have to install any extra software except the web browser, which makes it light weight. The downside of this approach is that it is not that powerful since only a limited set of semantic work units is

provided. In the next section, we will provide an example of this proposed approach by using the wildfire simulation service and the optimization service.

5.2.3 BPEL-based execution approach for service-oriented simulation experiment

If the user has advanced knowledge of workflow and BPEL, we propose to let the user to submit their BPEL script to our system and run the simulation experiment represented by the BPEL script on our server. In this approach, the workflow execution engine is moved from the web browser to the server, which means the uploaded BPEL script will be deployed on our server. When the experiment (represented by the BPEL workflow) finishes, the result is returned to the user. Figure 5.8 illustrates the architecture of this approach.

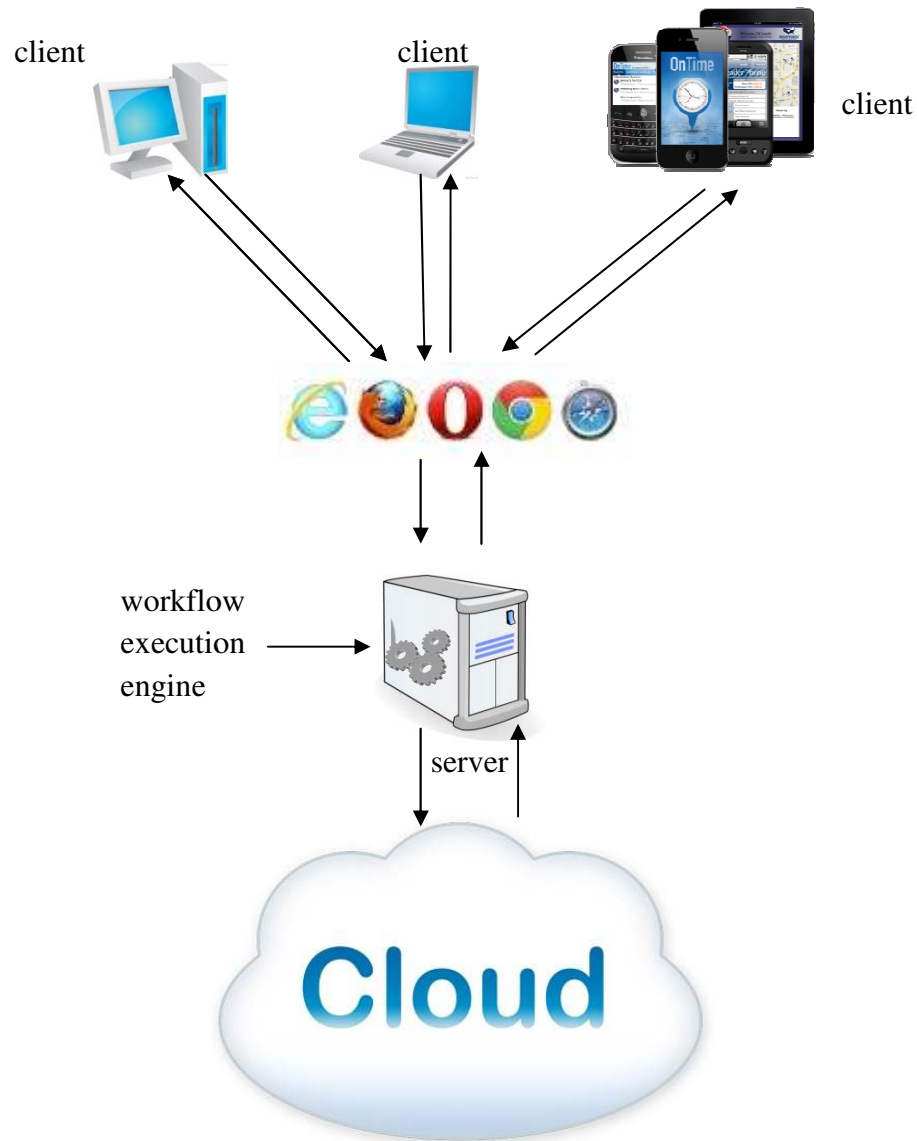


Figure 5.8 Architecture of the BPEL-based execution approach

There are plenty of BPEL execution engines such as Apache ODE [97], jBPM [98], Oracle BPEL Process Manager [99], and so on. In this work, we choose Apache ODE as our BPEL execution engine since it supports BPEL 2.0 and it is free.

The advantage of this approach is that BPEL provides a full set of support for more complicated simulation experiment design and the server can provide more powerful execution when the complexity of the experiment increases. On the other hand, it does require the user to have the BPEL knowledge to design the simulation experiment.

5.3 Experiments scenarios using the wildfire simulation service and the optimization service

In this section, we design several experiment scenarios using the wildfire simulation service and the optimization service to demonstrate the web-browser based approach.

1 Invoke work unit

Suppose you want to design an experiment to optimize the burned area. The steps you need to do by using the wildfire simulation service and the optimization service are:

- (1) create a simulation instance in the wildfire simulation service;
- (2) set the fuel data map for the created wildfire simulation instance;
- (3) set the aspect data map for the created wildfire simulation instance;
- (4) set the slope data map for the created wildfire simulation instance;
- (5) set the ignition for the created wildfire simulation instance;

(6) set the total simulation time for the created wildfire simulation instance and start to run;

- (7) query the burned area from the created wildfire simulation instance;

(8) use the returned burned area value as the input of the optimization service.

It is a sequential waterfall process and Figure 5.9 illustrates the organized experiment workflow using the *invoke work units*. In the next chapter, we will display this example in the implemented GUI.

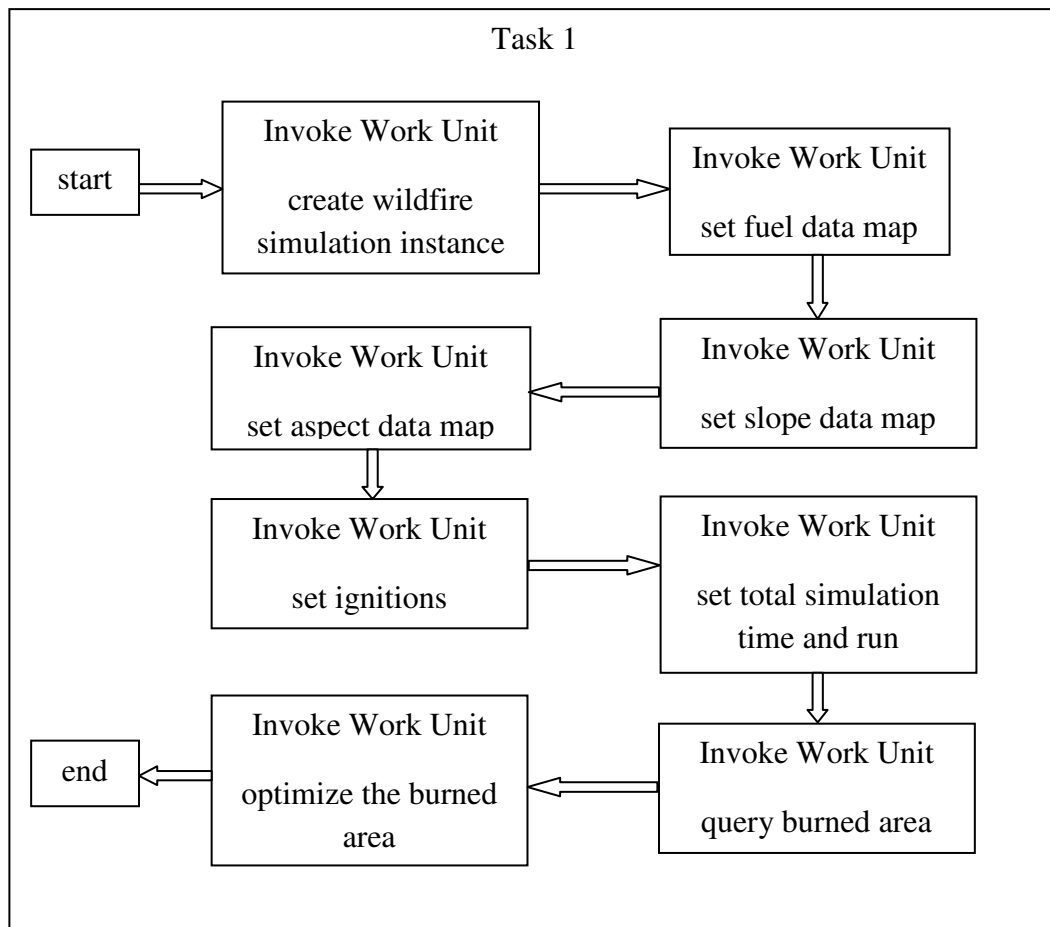


Figure 5.9 Experiment workflow scenario uses the *invoke work unit*

2 Assign work unit and while work unit

Suppose you need to do the previous experiment several times. You can first define a variable to represent the total times you want to do the previous experiment

using the *assign work unit*, and then put it into a while work *unit* as illustrated in Figure 5.10.

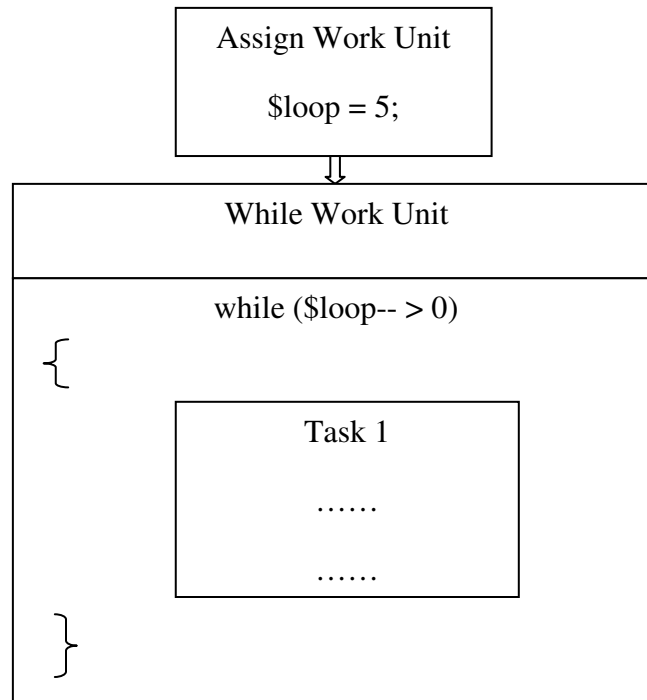


Figure 5.10 Experiment workflow scenario uses the *assign work unit* and the *while work unit*

3 If-else work unit

Suppose you want to do something based on the optimized burned area. For example, if the burned area exceeds certain threshold, you process it in one way; otherwise you process it in another way. This can be achieved using the *if-else work unit*, which is illustrated in Figure 5.11.

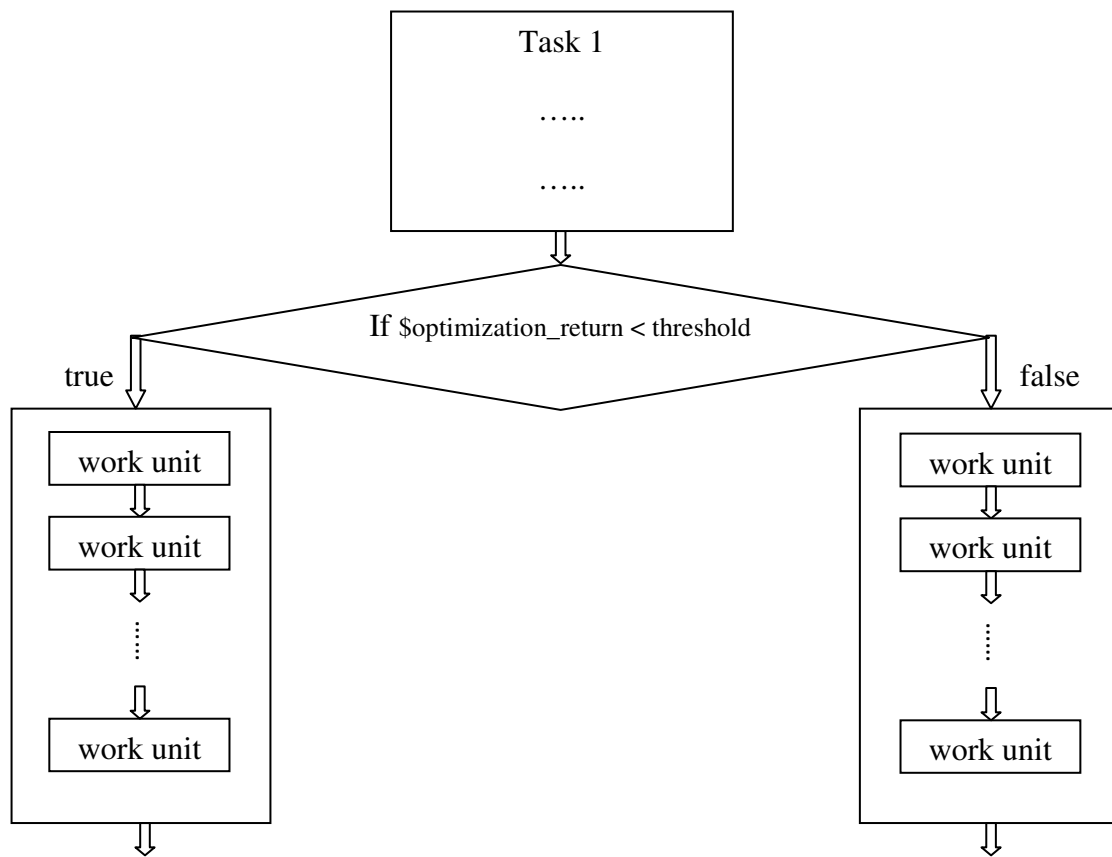


Figure 5.11 Experiment workflow scenario uses the *if-else work unit*

4 Wait work unit

Suppose you want to wait for some time before carrying out the next step. At this time, you can use the *wait work unit*. Figure 5.12 illustrates an experiment scenario.

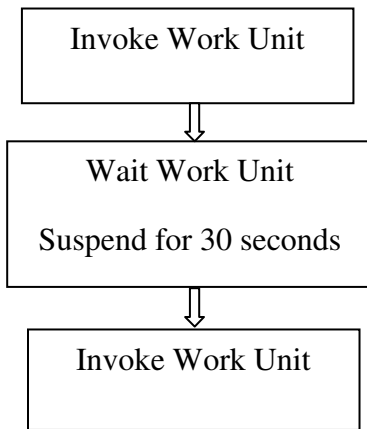











Figure 5.12 Experiment workflow scenario using the *wait work unit*

CHAPTER 6 GRAPHICAL USER INTERFACE OF THE PROPOSED WORK

In this chapter, we demonstrate the graphical user interfaces of our proposed work. We developed GUIs for wildfire simulation service, simulation service composition environment, and service-oriented simulation experiment environment. In this work, we assume that the provider of the simulation service is responsible for his/her own representation layer (GUI) of the simulation service.

6.1 Graphical user interface for wildfire simulation service

Wildfire simulation service is based on DEVS-FIRE [62, 69]. We implement the wildfire simulation service at the back end on the server by wrapping DEVS-FIRE, and provide a web-based GUI for its end users in the front end. Through the GUI, scientists and researchers can:

-  create a simulation service instance;
-  reset the simulation service;
-  destroy a simulation service instance;
-  choose a project to do the simulation;
-  set ignitions or fire lines;
-  set suppressions or suppression lines;
-  start the simulation;
-  pause the simulation;
-  stop the simulation;













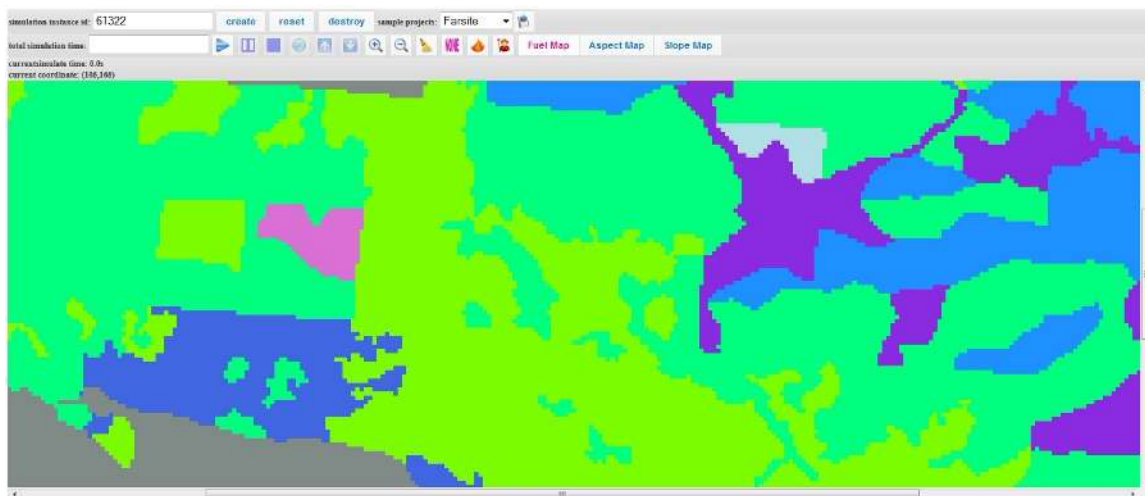
-  replay the simulation process;
-  view fuel map;
-  view aspect map;
-  view slope map;
-  zoom in map;
-  zoom out map;
-  accelerate the replay of the simulation;
-  slow down the replay of the simulation.

Figure 6.1 illustrates a scenario using the GUI. When a user first opens the GUI, Figure 6.1 (a) will be displayed. Each time a simulation is conducted, a corresponding simulation instance will be created and a corresponding simulation instance ID will be assigned. In this case, the simulation instance ID is 61322. To start to plan a simulation of wildfire, the user needs to choose a project, which basically determines the fuel, aspect, and slope data of the target area. After the user finishes choosing the project, the target area loaded with different maps will be displayed. By default, the fuel map is displayed when the project finishes loading. The user can select to see aspect map and slope map by clicking  button and  button. To start a wildfire simulation, ignitions need to be setup on the maps. The user can click  button first to adjust the current mouse click or drag event effect to setup an ignition or a fire line for the initial fire state. To setup an ignition, the user needs to left click on the map. To draw a fire line, the user needs to drag the mouse on the map with left button pressed, as shown in Figure

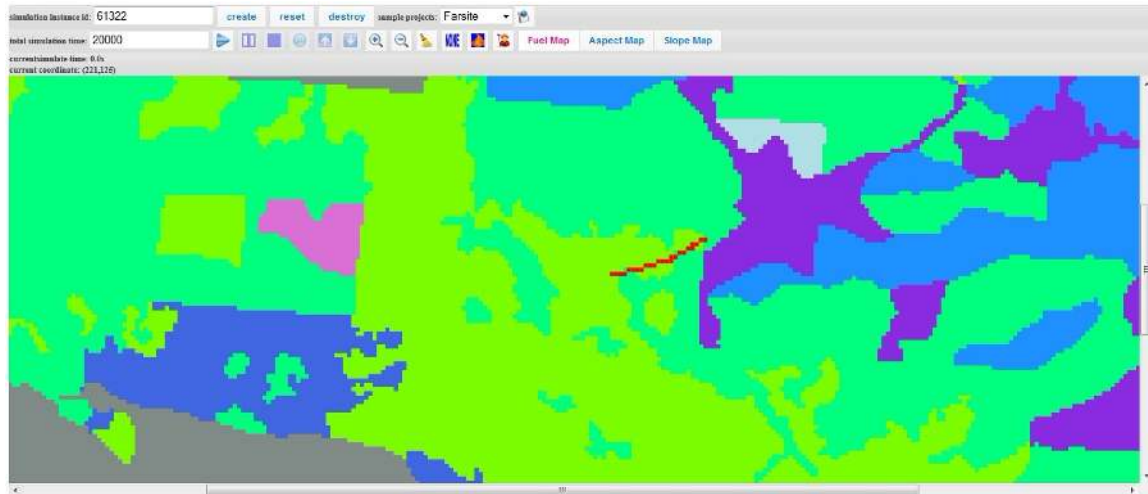
6.1 (c). The next step is to setup a total simulation time, as shown in Figure 6.1 (d) whose total simulation time is set to 2000 seconds. To start the simulation, the user needs to click  button; then the simulation runs and the fire starts to spread as shown in Figure 6.1 (d) with fuel map, (e) with aspect map and (f) with slope map.



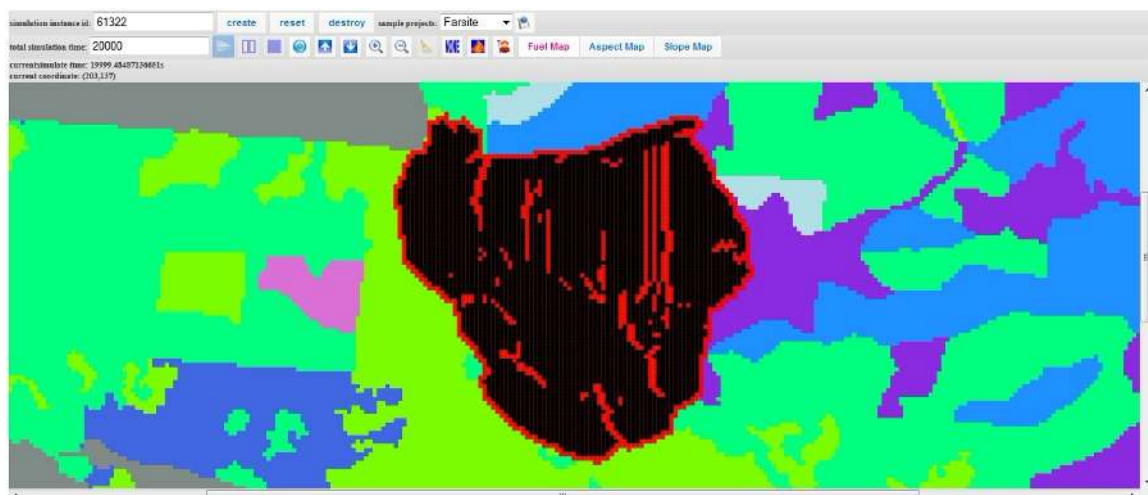
(a) Initial web page



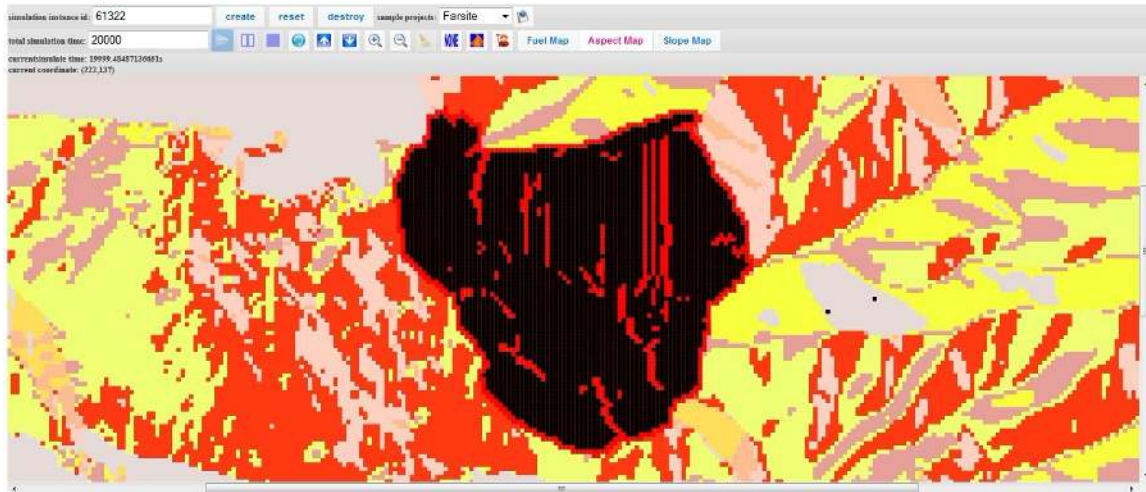
(b) A project is loaded



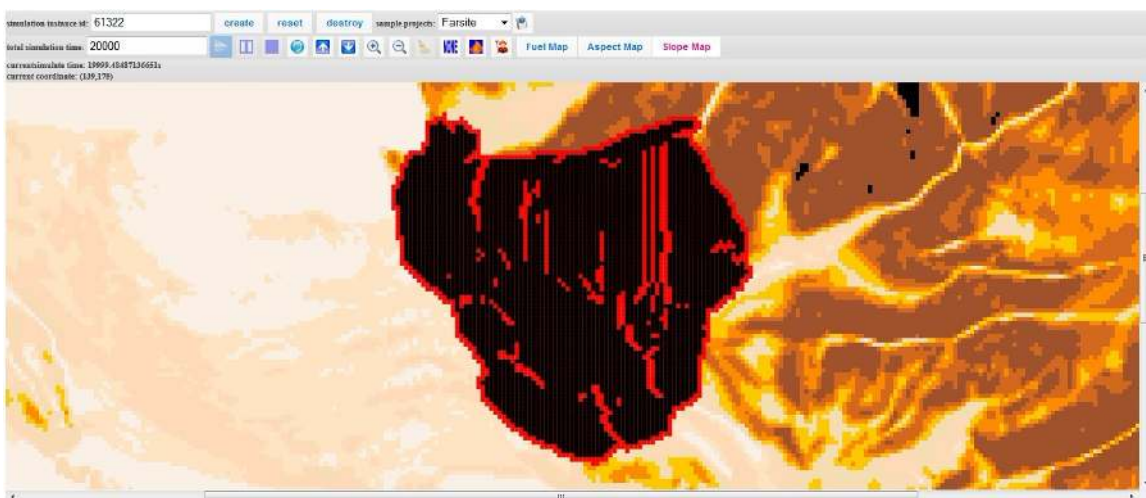
(c) Fire line is set



(d) Wildfire shape at the end of the simulation in fuel map



(e) Wildfire shape at the end of the simulation in aspect map



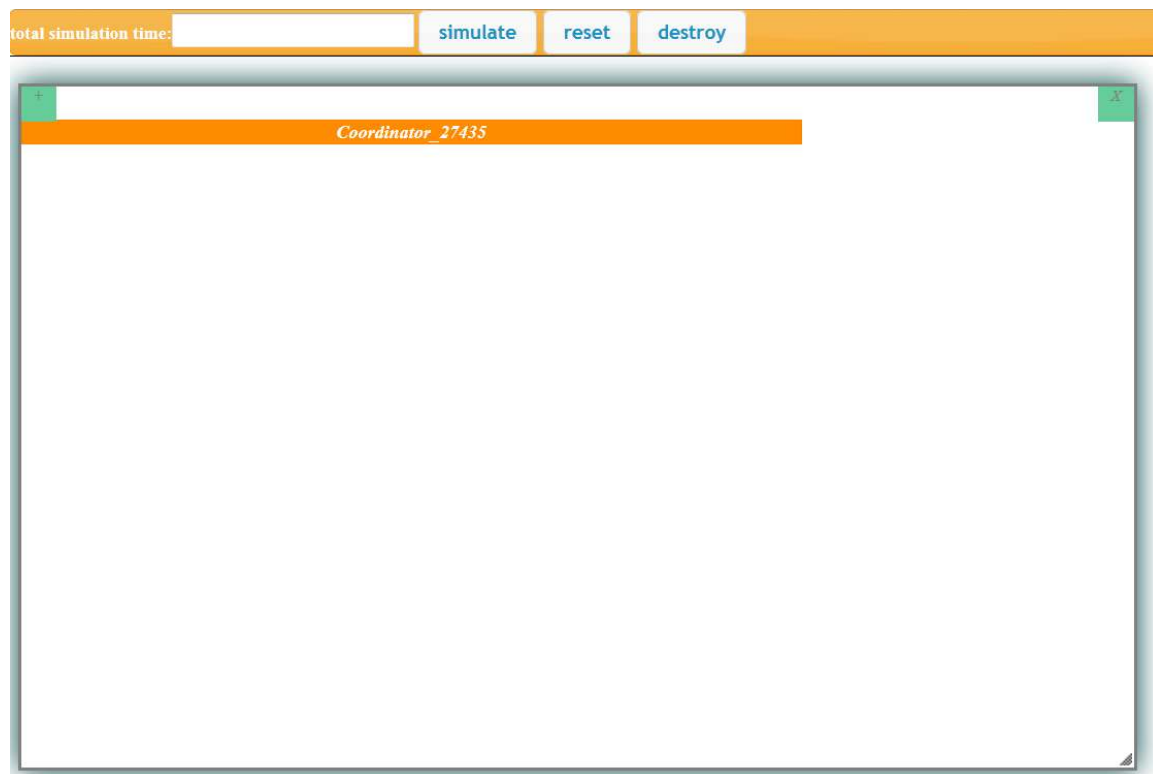
(f) Wildfire shape at the end of the simulation in slope map

Figure 6.1 GUI for wildfire simulation service.

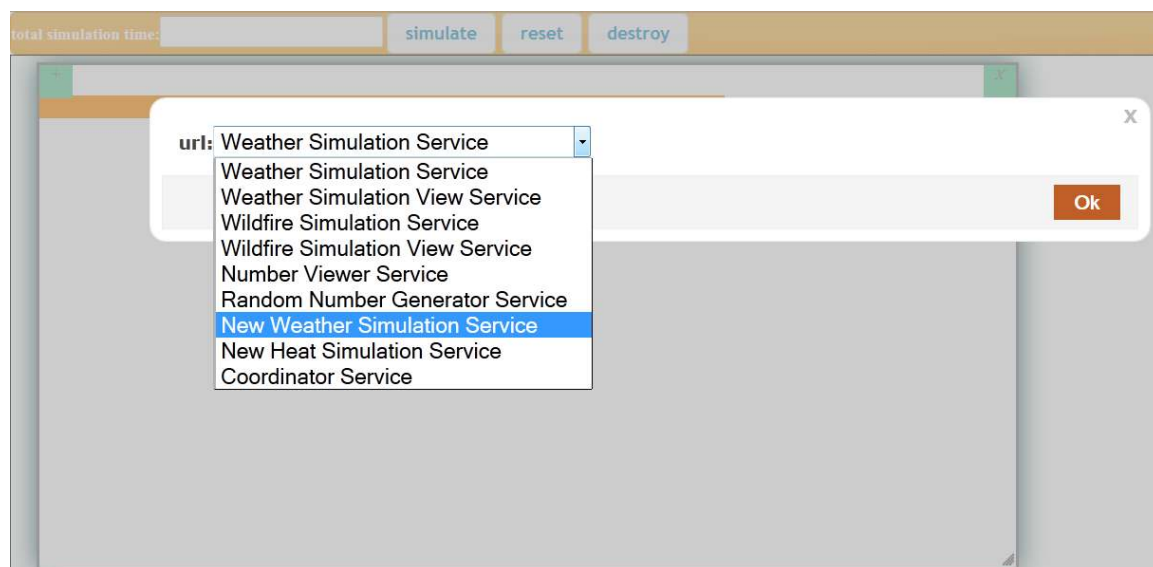
6.2 Graphical user interface for simulation service composition

To ease the process of composing simulation services, we developed an intuitive web-based interface. Next, we walk through an example to introduce the simulation service composition environment. When entering the interface, a root coordinator service

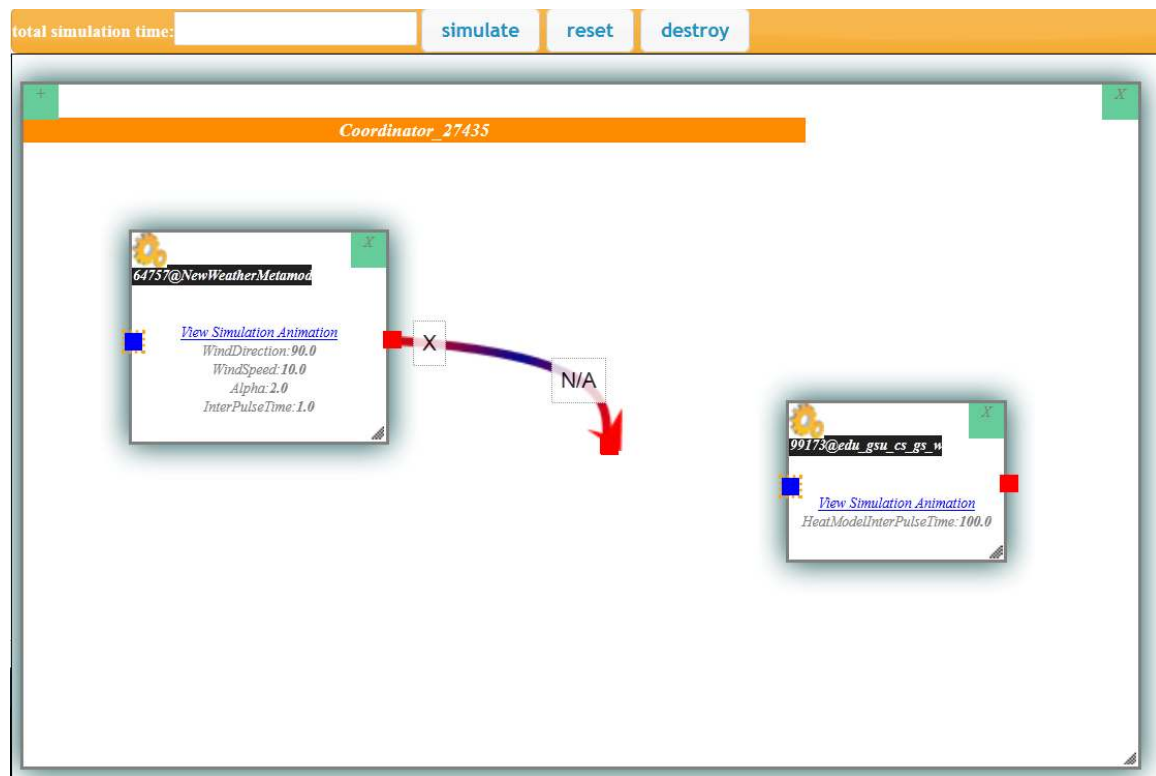
is created by default, as shown in Figure 6.2 (a). Users can add individual simulation services and coordinator service by clicking “+”. After clicking “+”, a prompt window asks users to choose simulation services, as shown in Figure 6.2 (b). If users do not want a simulation service, he or she can delete it by clicking the “X” on the top right corner of the box that represents that simulation service. Each simulation service has its own specific parameters which are listed in its “box”. Users can configure the simulation service by changing those parameters explicitly. After choosing the simulation service, one can set up couplings between simulation services by dragging the output port (displayed as red end) of one simulation service and dropping it to the input port (displayed as blue end) of the other simulation service, as shown in Figure 6.2 (c). After making the coupling, the output represented by the red end from the source simulation service will be fed to the input represented by the blue end of the destination simulation service. Note, each simulation service may have multiple input ports and output ports; and it is the user who decides how to make the couplings among input ports and output ports. After a coupling is made, users can delete a coupling by clicking the “X” on that coupling. To add a simulation granularity handling agent, users can double click on that coupling, and a prompt will pop up waiting for users to choose the agents and set up the value for that agent, as displayed in Figure 6.2 (e). To start the composed simulation service, the user needs to setup a total simulation time and click “simulate” button to run the composed simulation service governed by the coordinator service. To view the simulation animation or the simulation result provided by that individual simulation service, the user can click “View Simulation Animation” and a new window will be opened to show the corresponding simulation animation or simulation result.



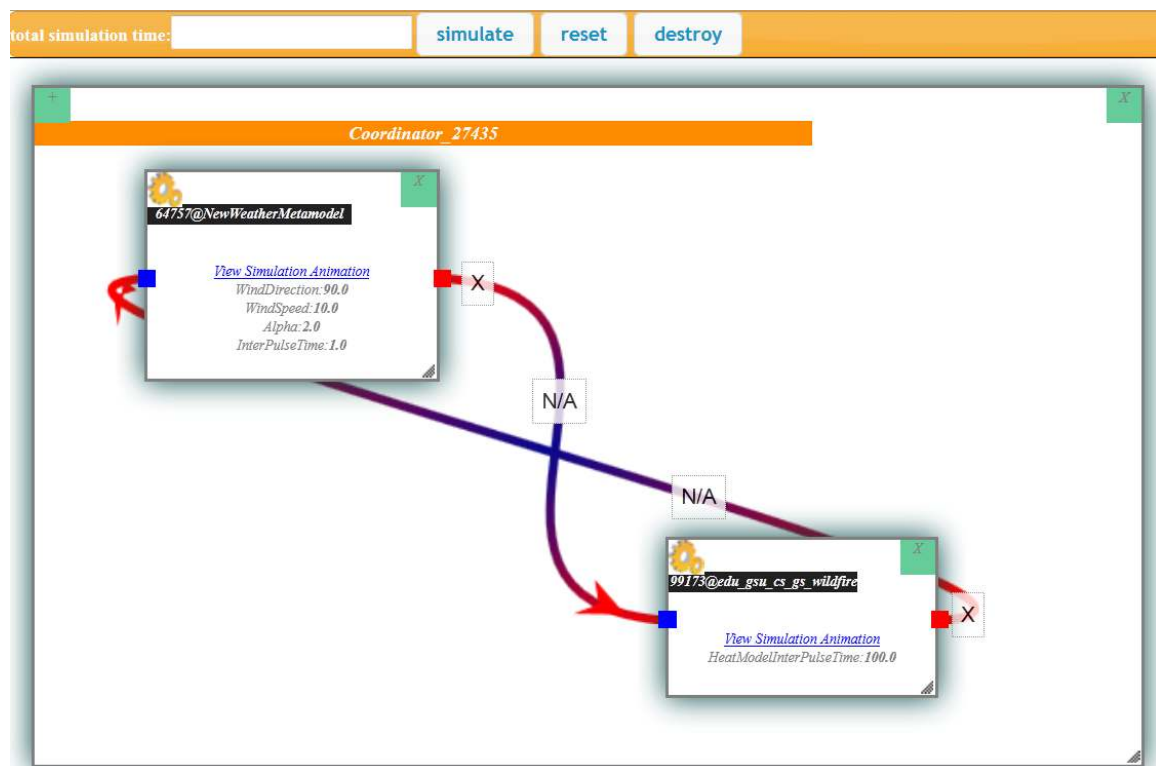
(a) A default coordinator service



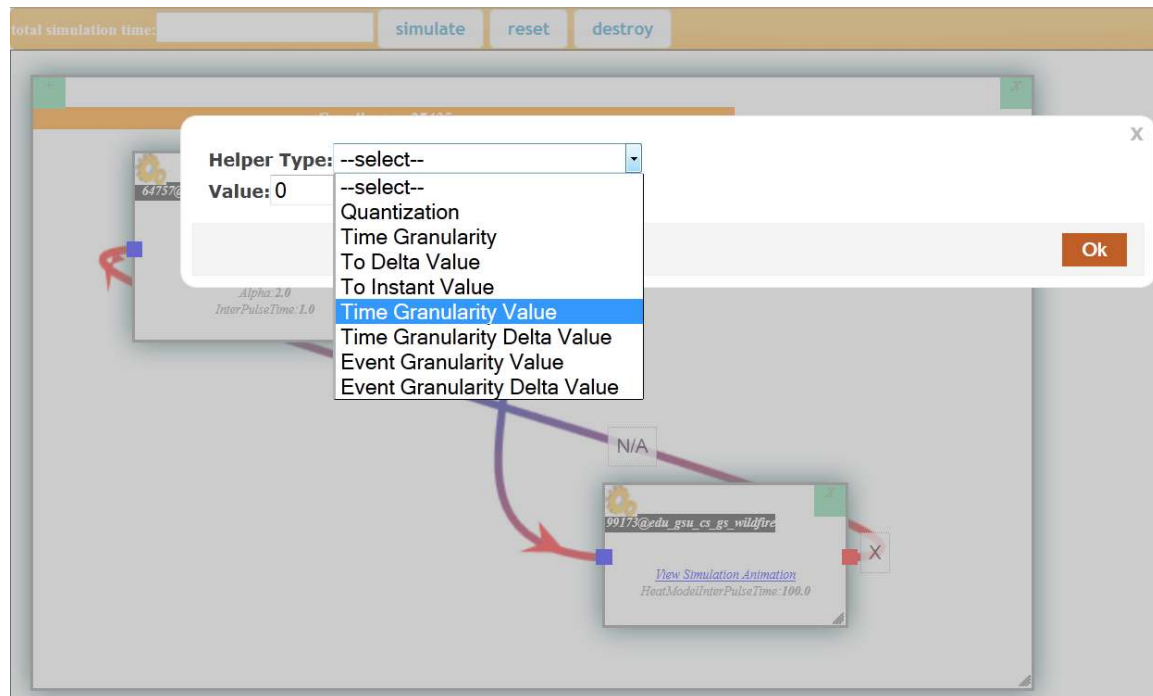
(b) A prompt showing available simulation services



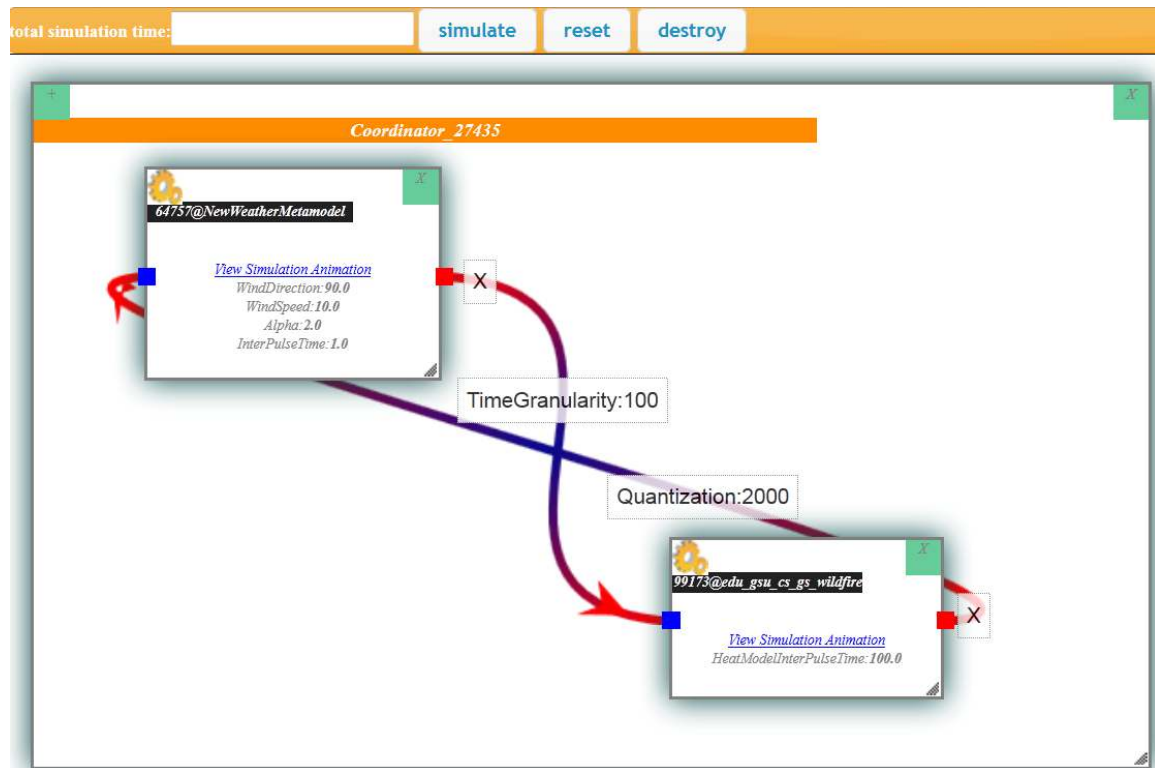
(c) Dragging the output to make a coupling



(d) Two coupling are made



(e) Chooseing granularity handling agents



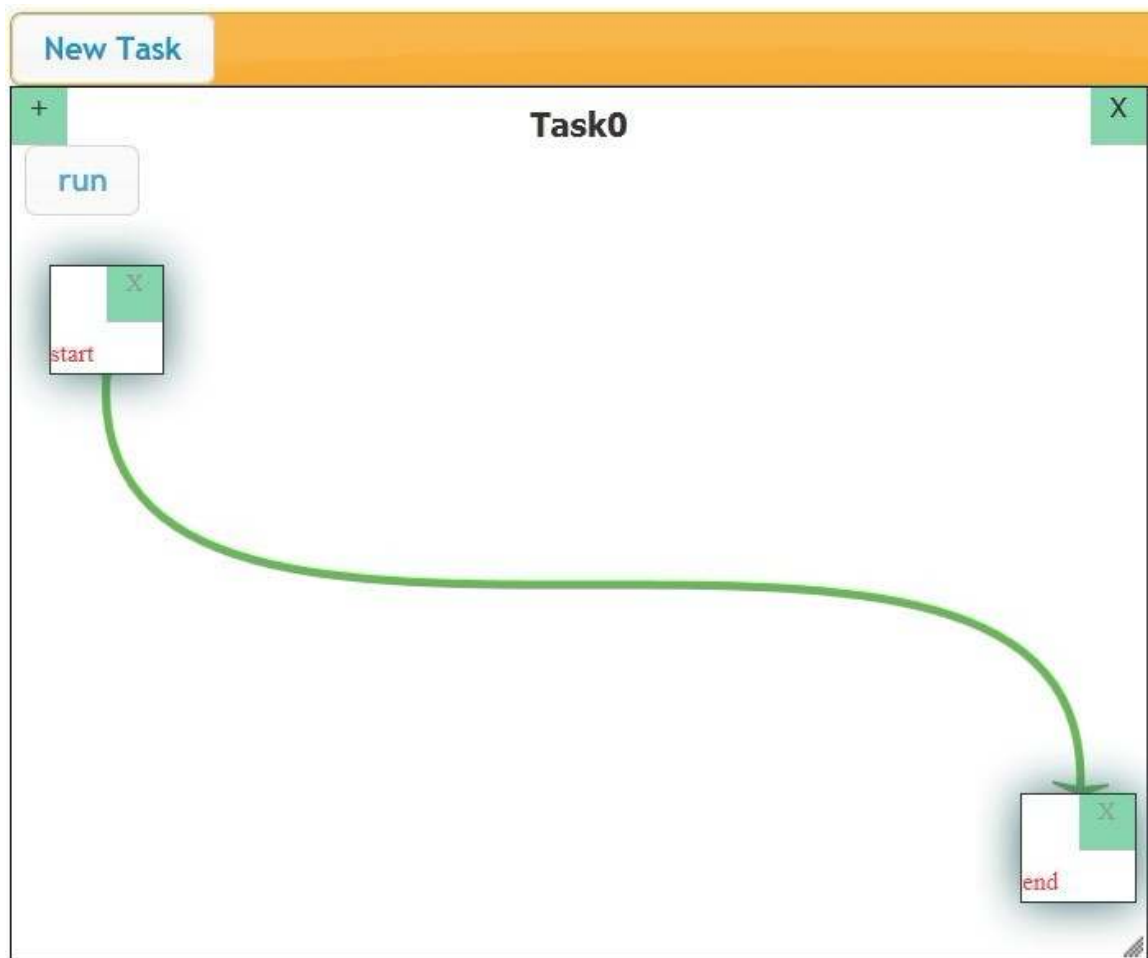
(f) Simulation composition enhanced with granularity handling agents

Figure 6.2 GUI for simulation service composition.

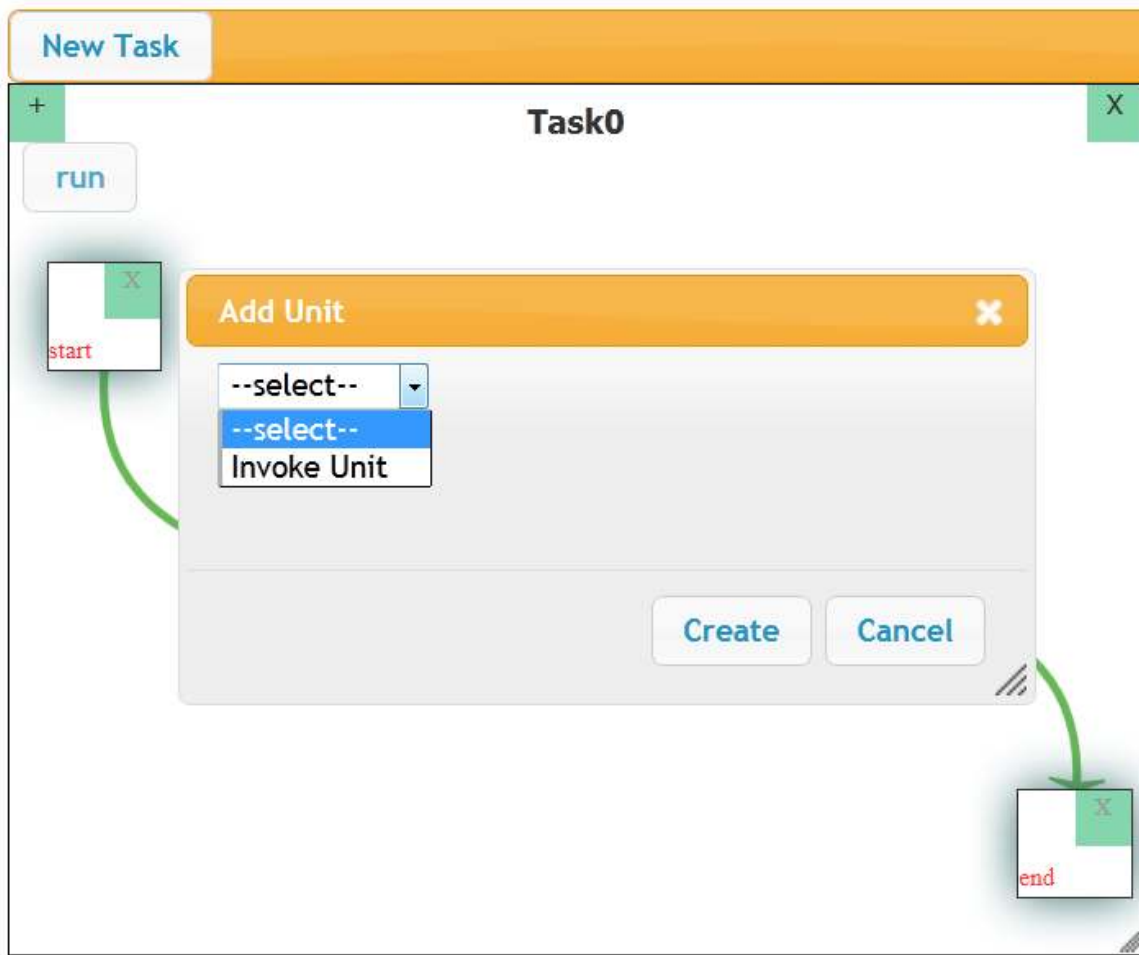
6.3 Graphical user interface for service-oriented simulation experiment

In this section, the GUI for service-oriented simulation experiment is introduced. We employ the wildfire simulation service and the optimization service to construct an experiment to illustrate this environment. In this demo experiment, we want to optimize the burned area of the wildfire by the optimization service. When entering the experiment environment, a “Task” is created by default, as shown in Figure 6.3 (a). A task represents a series of work that are organized in a sequence order. Initially there is no work between the “start” and the “end” dummy work unit. If the user wants to add work, he or she can click “+” and a dialog box shows up for the user to choose the work unit he or she wants to use, as shown in Figure 6.3 (b). When finished, a box which represents the work unit is

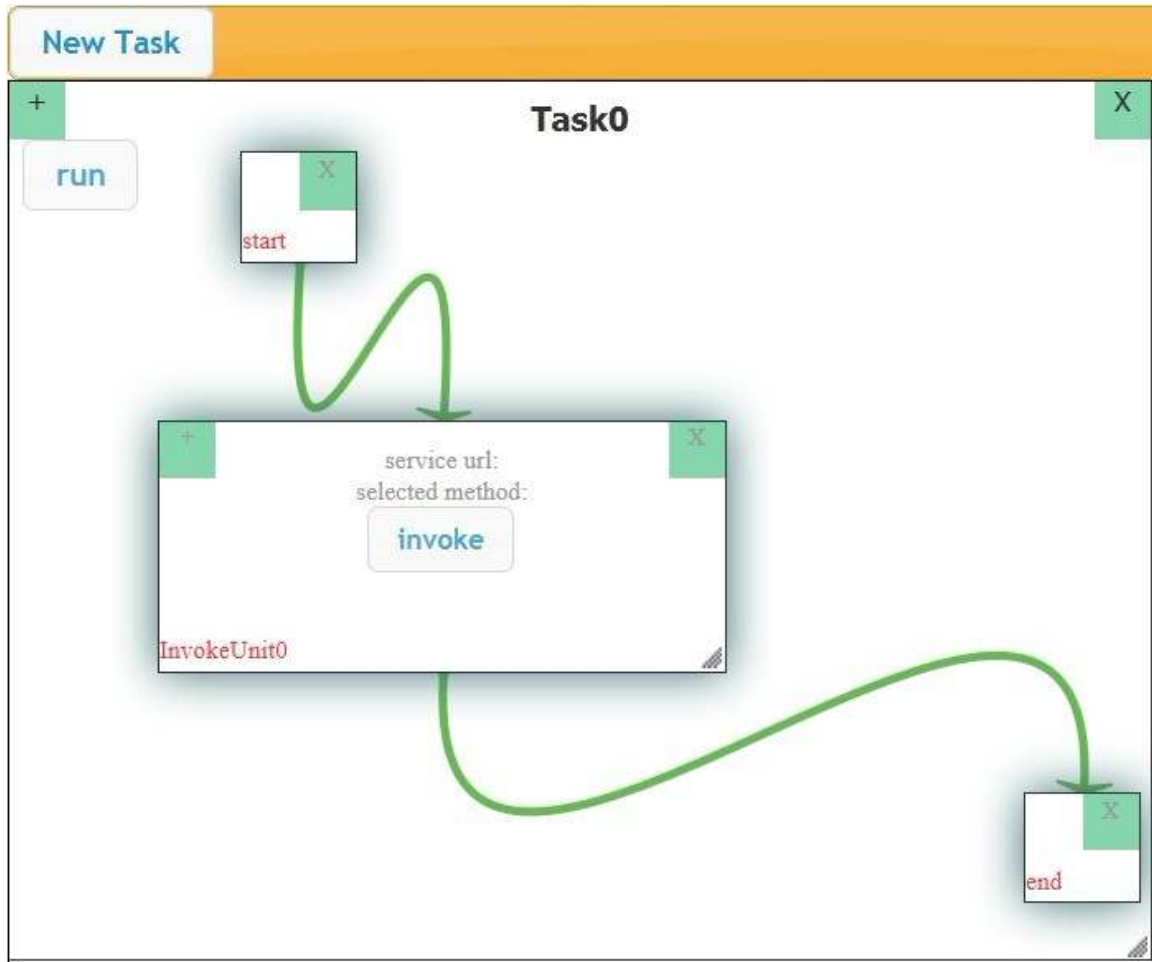
added between “start” and “end” dummy work units, as shown in Figure 6.3 (c). To configure this work unit, the user can click “+” in the box that represents this work unit and a dialog box will be prompted. Then, the user can enter the service URL that he or she wants to use and then select the operation this work unit (in this case the work unit is a invoke unit) will invoke provided by that service. This is shown in Figure 6.3 (d). After choosing the operation, the user can configure the parameters for this operation if there is any. If the user wants to use some return value of some unit, he or she can configure the parameter field by typing “\$”; and then a list of the return values will show up for the user to choose. For example, “\${InvokeUnit1_return}” represents the return value of the invoke unit 1. This is illustrated by Figure 6.3 (e) and (f). To obtain the burned area of the wildfire simulation service, we need to create a simulation service instance, set the fuel data, set the aspect data, set the slope data, add ignition, start the simulation and query the burned area. Then, we feed the burned area to the optimization service. Thus, we need eight invoke units, each of which will invoke the operation provided by the wildfire simulation service and the optimization service, as displayed in Figure 6.3 (g). After we configure all eight invoke units, we can click the “run” button in the task box to start the experiment and each work unit will execute one after another. When the experiment is done, the experiment result can be looked up by checking the return value of the corresponding work unit as shown in Figure 6.3 (h).



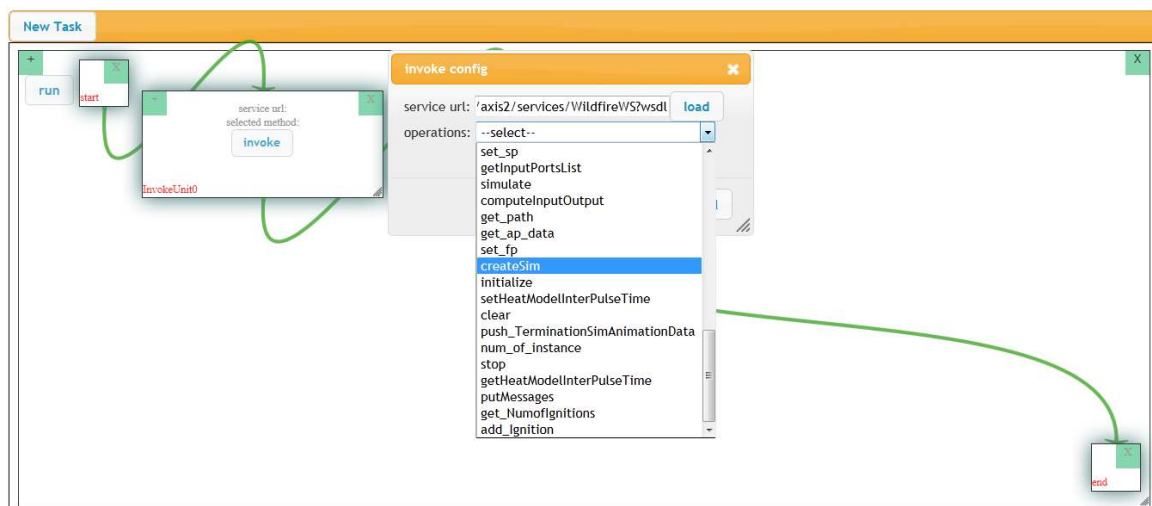
(a) The default task



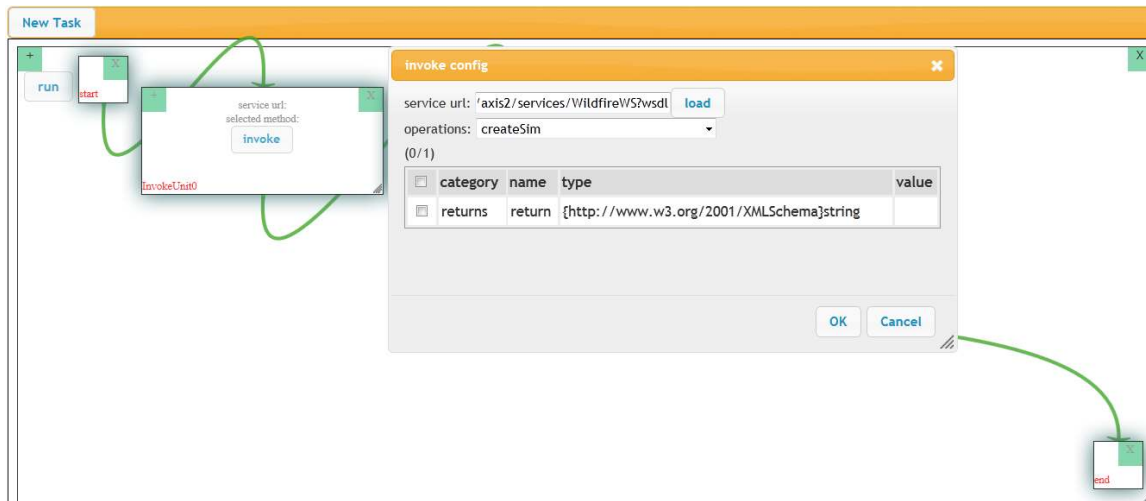
(b) Choosing work unit type



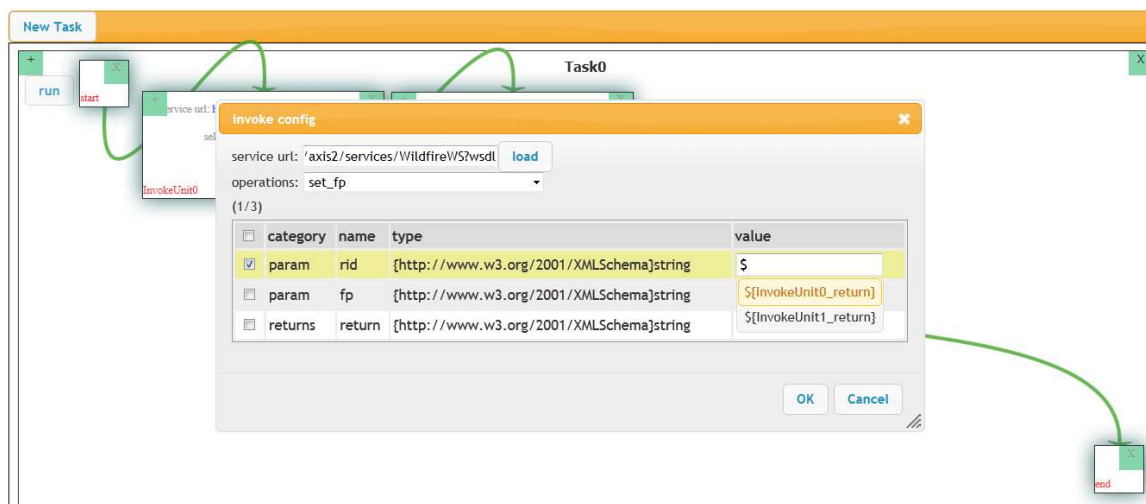
(c) An invoke unit is added



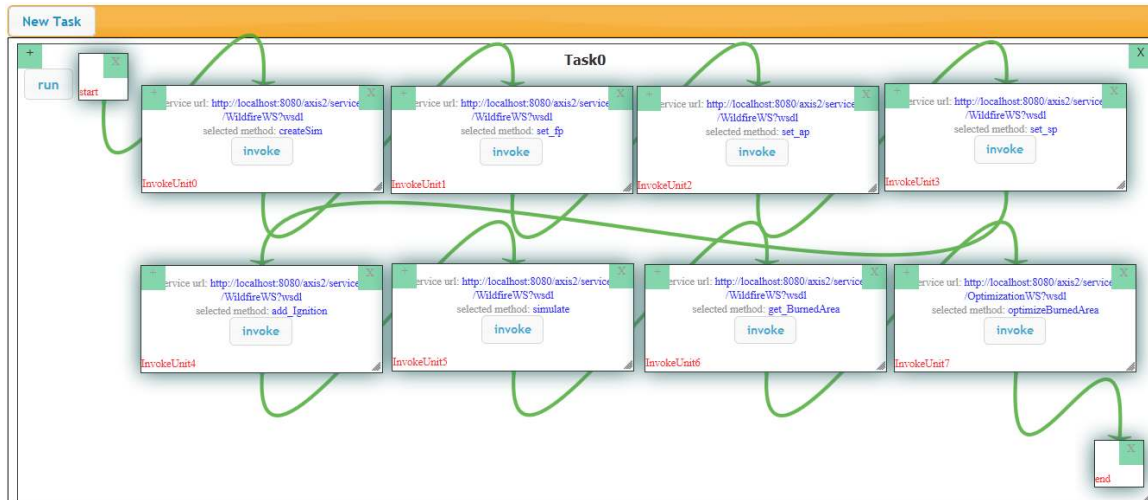
(d) Configure invoke unit



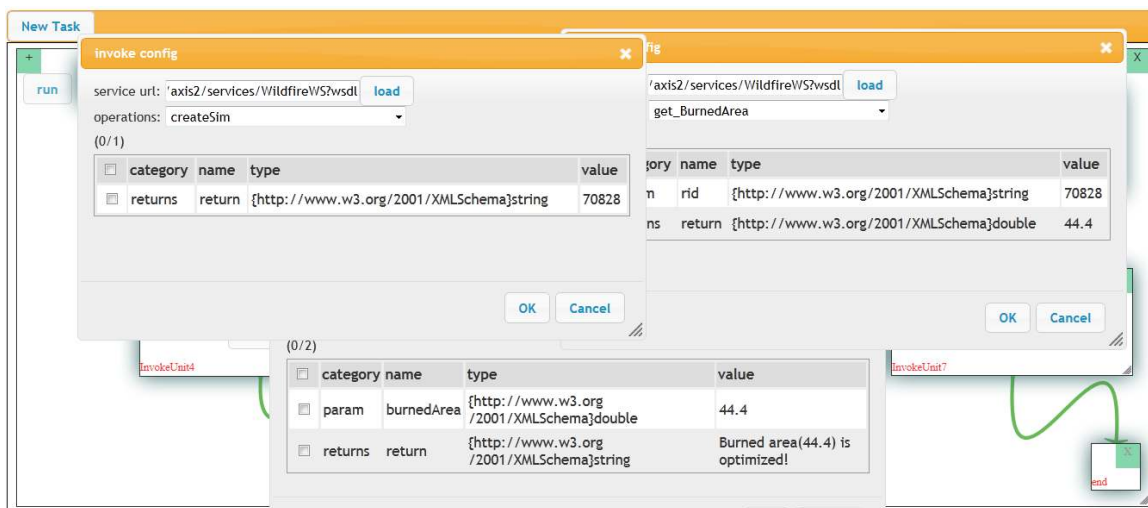
(e) The operation with no input parameter



(f) Configure operation with input parameters



(g) Eight invoke units for the wildfire-optimization simulation experiment



(h) Experiment result

Figure 6.3 GUI for service-oriented simulation experiment.

CHAPTER 7 SPATIAL PARTITIONING ALGORITHMS FOR PARALLEL WILDFIRE SIMULATION

In previous chapters, we proposed a framework for simulation software as a service and service-oriented simulation experiment. As for an individual simulation service, its performance might be influential to the performance of the composed simulation service and the whole experiment, especially when the scale of the simulation becomes extremely large.

Recent years have witnessed an increasing number of wildfires with increased burned areas as well as damages to the environment. Understanding the behavior of wildfires and predicting the fire intensity, rate of spread, and spread direction can support better management and control of wildfires. Towards this goal, several wildfire spread simulation models have been developed, including FARSITE [66], BehavePlus [67], Hfire [68], and DEVS-FIRE [62, 69]. Simulating large-scale wildfires is a computation-challenging task because of the complexity of the fire spread model and the large size and long duration of the simulations. To improve the performance of wildfire simulations, parallel simulation techniques are needed.

To support parallel simulations we need to divide the simulation tasks and assign them to multiple processing units (PUs). This step of task partitioning is critical and can have significant impact on the performance of a parallel simulation. Wildfire simulation models are characterized by their dynamic behavior in both space and time. Due to the spatial-temporal behavior, spatial partitioning is commonly employed for parallel simulations of wildfires. In spatial partitioning, the space is divided into multiple regions, which are distributed to multiple PUs. When a fire spreads to a new region, it triggers the

computation of the PU in charge of that region. This paper considers spatial partitioning for parallel simulation of wildfire spread using the DEVS-FIRE model. DEVS-FIRE is a discrete event wildfire spread and suppression model based on the DEVS formalism [13]. It employs a two dimensional cellular space model composed from multiple individual cells to represent the wildfire area. When simulating large-scale wildfires modeled by a large number of cells, there is a need to improve the simulation performance using parallel simulation techniques.

In this chapter, we employ the complex wildfire simulation as an example and provide two partition algorithms aiming to improve the performance of large scale wildfire simulation.

7.1 DEVS-FIRE Overview

7.1.1 System architecture of DEVS-FIRE

DEVS-FIRE is based on the discrete event system specification (DEVS) and uses a cellular space model for simulating wildfire spread and agent models for simulating wild fire containment DEVS. The overall structure of DEVS-FIRE is shown in Figure 7.1. When a cell is ignited, Rothermel's model (the Behave model) [70] is used to compute the speed and direction of fire spread. Built on the fire spread model, DEVS-FIRE also supports fire containment simulation by connecting to the fire fighting model. Our overview focuses on the aspect that is related to the profile-based method presented in this paper. Other aspects and more details of the DEVS-FIRE model can be found in [62].

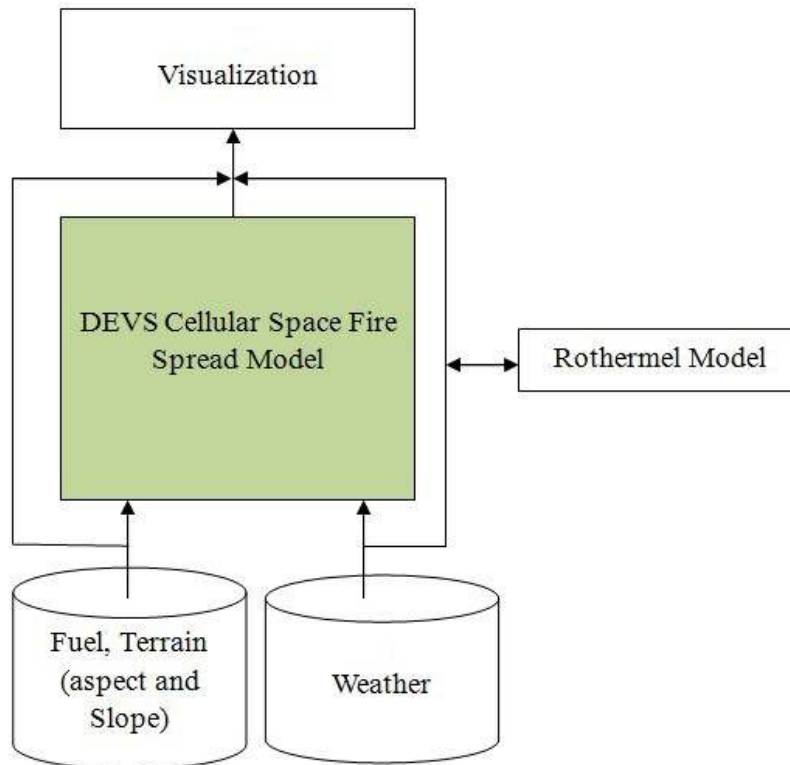
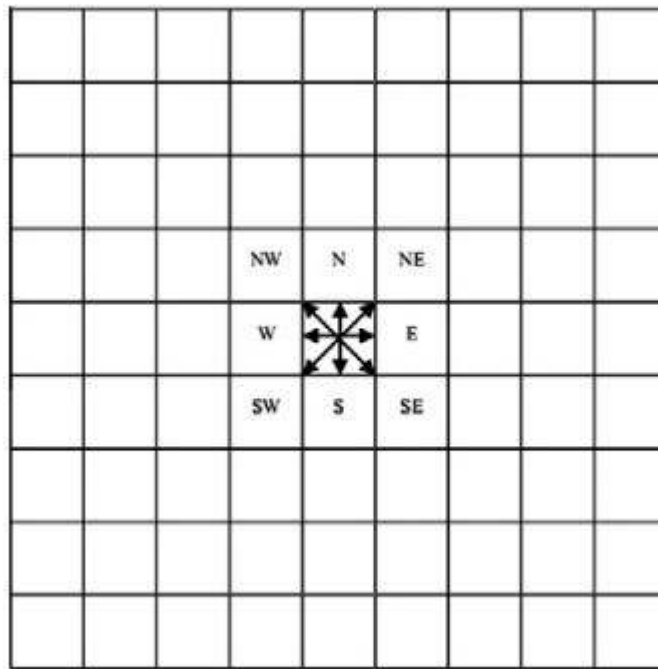


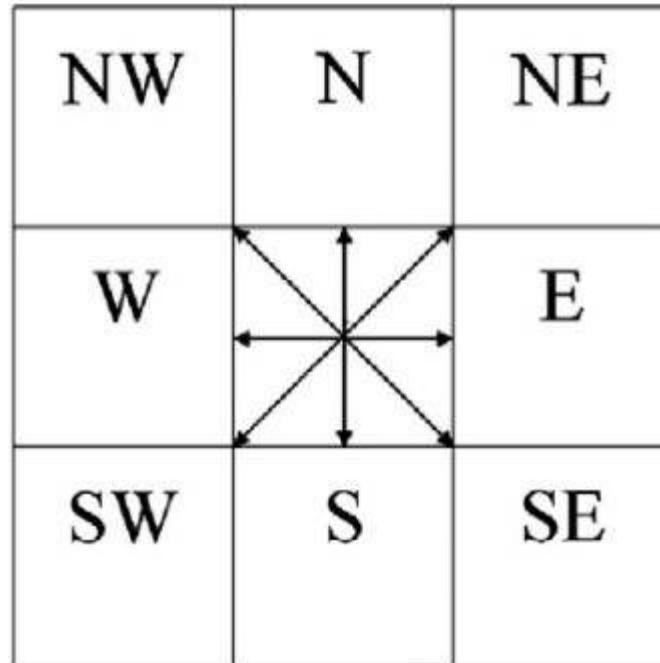
Figure 7.1 Overall system architecture of DEVS- FIRE

DEVS-FIRE is an integrated simulation environment for surface wildfire spread and containment simulation based on the DEVS formalism. It employs a two dimensional cell space to model the field where the fire occurs. The cells in the cell space are rectangular, whose sizes depend on the resolution of the GIS data (fuel, slope and aspect) [70]. The cell space contains individual forest cells each of which contains the GIS data and weather conditions assumed to be uniform within the cell. Each cell in the cell space is represented as a DEVS atomic model in the simulation and is coupled with its eight neighbor cells located at the N, NW, W, SW, S, SE, E, and NE directions respectively, as shown in Fig. 1. Consequently, the forest cell space is a coupled model composed of a

number of coupled forest cell models. Fire spreads from one cell to another is enabled via message passing between the two cells. As a result, the fire spreading is modeled as a propagation process where burning cells ignite their unburned neighbor cells. Figure 7.2 shows two cell space models at different spatial resolutions for the same forest area. Figure 1a shows a 9×9 cell space model and Figure 1b shows a 3×3 cell space model, where each cell corresponds to 3×3 cells in Figure 7.2(a). The arrows in Figure 7.2 represent the output couplings from the center cell to its eight neighbors.



(a) 9×9 high resolution cell space with smaller cells



(b) 3×3 low resolution cell space with larger cells

Figure 7.2 Examples of 2D cell space.

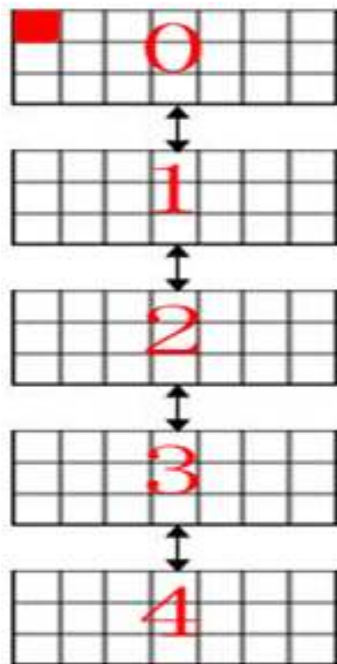
In DEVS-FIRE, all cells are initially set to the unburned (passive) state. If a cell receives an ignition message and its fire line intensity is larger than its burning threshold, its state changes to burning. A cell, once ignited, calculates its rate of spread and spread direction using Rothermel's fire behavior model based on its fuel, slope, aspect, and weather data. The rate (and direction) of spread is then decomposed into eight directions corresponding to its eight neighbor cells (see [62, 71] for more details). Based on these spread rates and the center-to-center distances between the cell and its neighbors, the cell schedules to ignite its neighbors (through message passing) according to the corresponding time delays. Finally, a burning cell changes to the burned state after it is burned out. Next, two spatial partitioning algorithms are presented.

7.2 Spatial uniform partition for parallel simulation of DEVS-FIRE

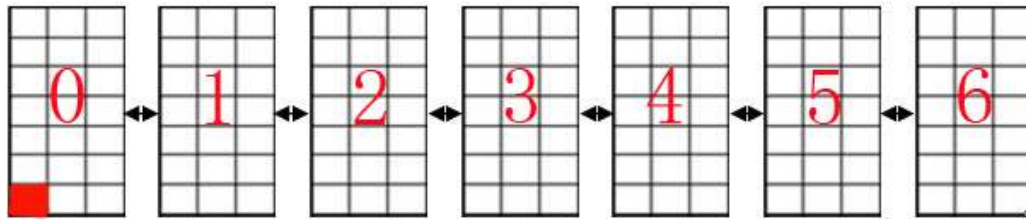
As illustrated above, an intuitive method to partition the cellular space is spatial uniform partition. The basic idea of this approach is to divide the two-dimensional cellular space of DEVS-FIRE into multiple smaller size-equivalent cellular spaces and allocate each partition to a PU for parallel simulation. There are three approaches to achieve this:

1. Horizontal spatial uniform partition, which is shown in Figure 7.3(a);
2. Vertical spatial uniform partition, which is shown in Figure 7.3(b);
3. If we have certain number of Pus (processing units), we can combine vertical spatial uniform partition and horizontal spatial uniform partition into a hybrid spatial uniform partition, which is shown in Figure 7.3(c).

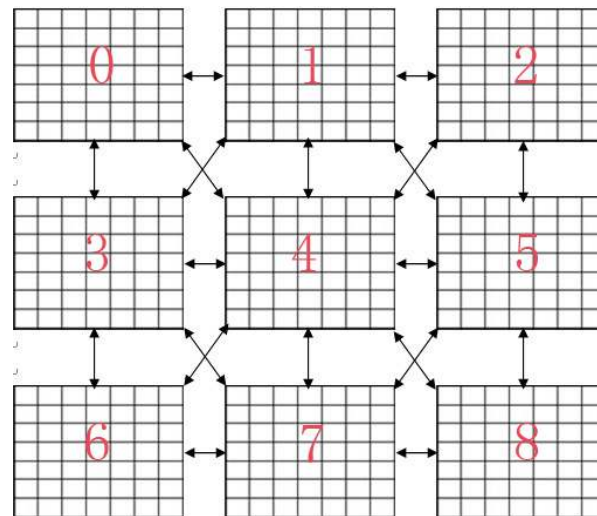
Figure xx illustrates three examples for horizontal spatial uniform partition, vertical spatial uniform partition and hybrid spatial uniform partition respectively.



(a) Horizontal spatial uniform partition



(b) Vertical spatial uniform partition



(c) Hybrid spatial uniform partition

Figure 7.3 Three simple approaches to partition the cellular space.

Simple as it is, this partition method suffers two defects. One defect is that the number and the locations of the ignition points can significantly influence the performance of parallel simulation. For example, if we have a 400×400 cellular space and 4 processing units, we can partition this cellular space into 4 200×200 cellular spaces. However, if there is only one ignition point, the performance will vary a lot as the location of the ignition point moves from corner to the partition boundary (Guo et al. 2009). The utilization of each PU also varies as the ignition point moves. If the ignition point locates far away from the partition boundary, only one processing unit is working while the other three are doing nothing but waiting for the burning messages. The other

defect is that the number of PUs involved in the parallel simulation is somehow restricted by how the space is divided. Generally, it is better to use square number (such as 4, 9, 16, ..., N^2 , where N is an integer) of PUs under uniform partition method. The reason is that we can result a hybrid partitioning rather than a partitioning in horizontal or vertical fashion. We note that it is still possible to produce a partitioning in a horizontal fashion or in a vertical fashion (see Figure 1 as an example) using square number of PUs. However, the result is either unbalanced or has poor parallelism. The advantage of hybrid partitioning is that each partition has more neighbor partitions (at least 3 neighbor partitions and 8 at most) than in horizontal or vertical partitioning (at most 2). Figure 1 shows a horizontal partitioning and a hybrid partitioning of a cell space using 9 PUs. In this example, if the fire is ignited at the top right corner of the partition0, the PUs in horizontal partitioning will be poorly utilized during the run time of the parallel simulation because the PUs work nearly sequentially. However, in hybrid partitioning, fire can spread from partition0 to partition1, partition3 and partition4, thus achieving more parallelism than in horizontal partitioning. In order to improve these two defects, we proposed another spatial partition algorithm called profile-based spatial partition.

7.3 Profile-based spatial partition for parallel simulation of DEVS-FIRE

From the simulation point of view, a cell is active when it is in the burning state. When a cell is active, the simulation needs to compute the cell's fire spread speed and direction, and to schedule ignition events for this cell to ignite its neighbors. On the other hand, if a cell has not been ignited or it is burned out, there is no computation needed for the cell. Figure 7.4 displays a simulation scenario, where the unburned, burning, and burned out areas are marked. As illustrated in Figure 7.4, in wildfire spread simulations

all the burning cells form one or multiple spatially active areas, where intensive computation occurs. If we can concentrate all the computing power only on these active areas, it will accelerate the performance of large scale simulations. This motivates us to develop the profile-based partitioning method for parallel simulations of DEVS-FIRE.

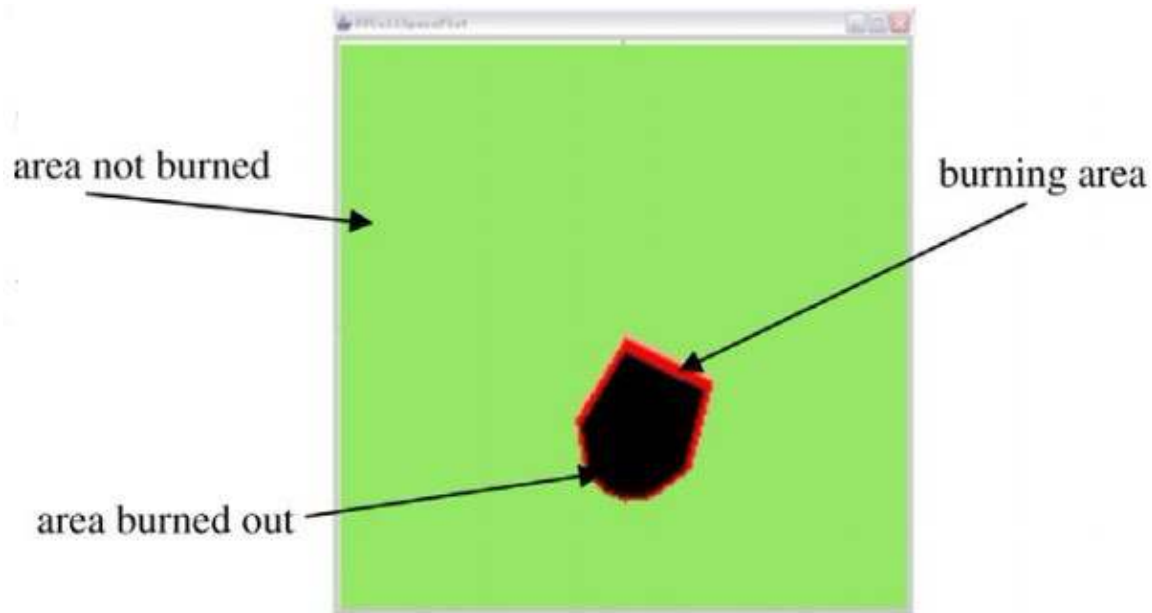


Figure 7.4 A simulation scenario in DEVS-FIRE

7.3.1 Architecture

Based on the active area concept described above, we propose a profile-based spatial partitioning method for parallel simulation of large scale wildfire using DEVS-FIRE. The basic idea is to employ the result from a low resolution simulation as a profile of the fire spreading behavior to guide the partitioning process of the cell space among available PUs. The profile is an ignition sequence of the low resolution cells in the low resolution simulation. Although not as precise as the high resolution simulation, it contains

knowledge regarding how the active area evolves in the high simulation. Once we have this profile, we partition the cells among participating PUs so that all PUs are devoted to the predicted active area at an early stage of the parallel simulation. Figure 7.5 shows the architecture of the profile-based spatial partitioning method. The architecture includes four parts that represent four major steps of the profile-based spatial partitioning method:

- (1) producing low resolution GIS data,
- (2) producing profile,
- (3) profile-based partitioning,
- (4) parallel simulation of DEVS-FIRE based on the partitioning.

Below we describe these four components in detail.

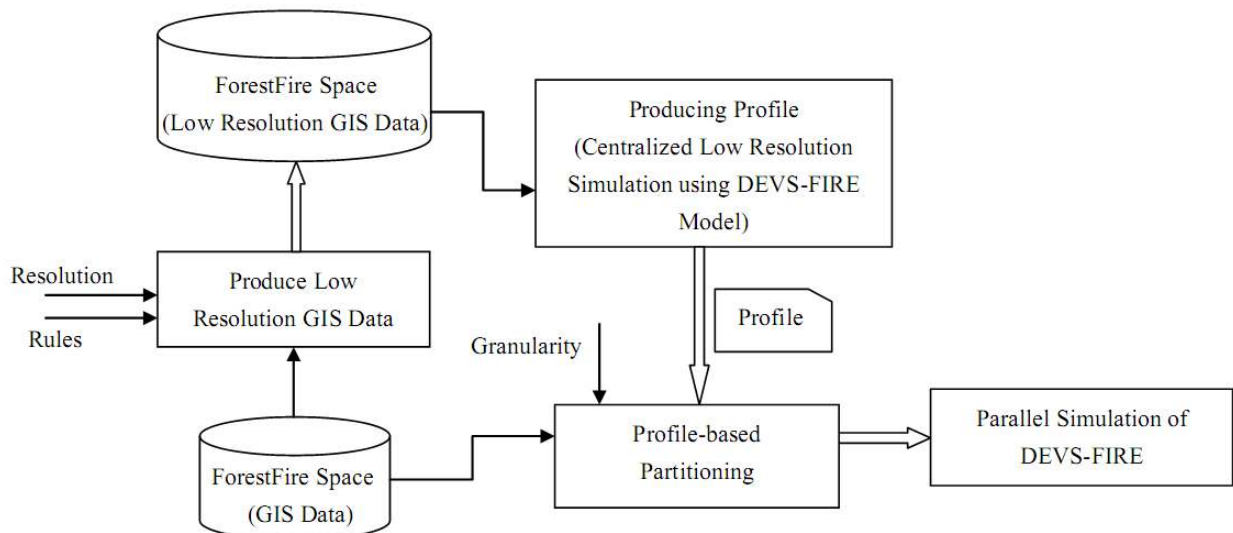


Figure 7.5 Architecture of the profile-based spatial partition method

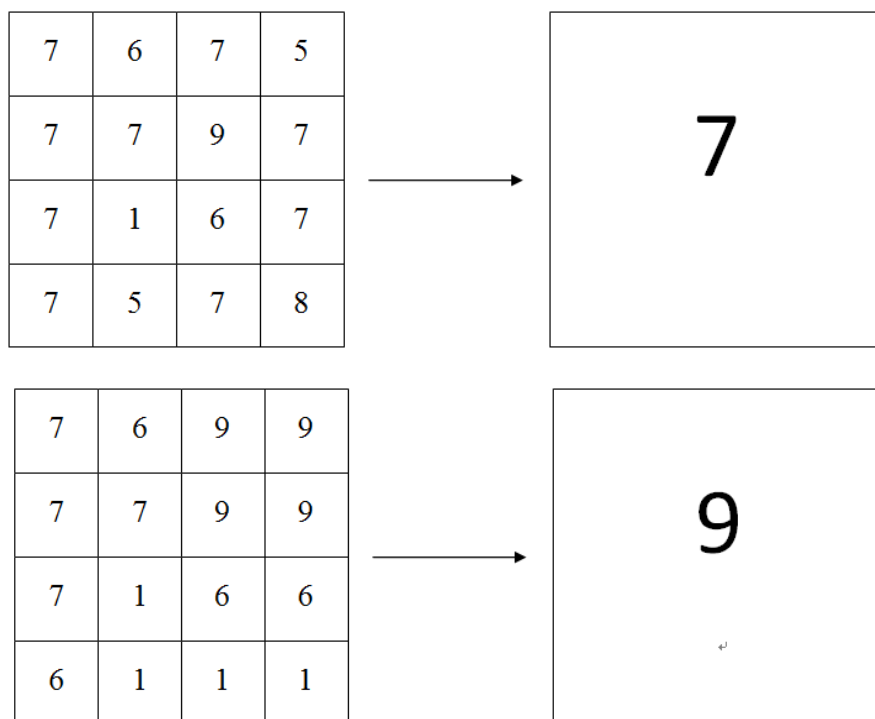
7.3.2 Producing low resolution GIS data

In order to obtain the profile from the low resolution simulation, we need a way to generate low resolution GIS data. Different resolution GIS data can be generated using dedicated software from the source files, e.g., satellite images. However this approach needs to have access to the source files and to use dedicated software. In our work, we produce low resolution GIS data from the original GIS data that already exist. We define R as the resolution factor from the original GIS data to the low resolution GIS data. This resolution factor specifies how many cells (in both horizontal and vertical dimensions) in the original GIS data are converted into one cell in the low resolution GIS data. For example, when $R=3$, every 3×3 cells in the original GIS data are converted into one cell in the low resolution GIS data (see Figure 1 for an illustration).

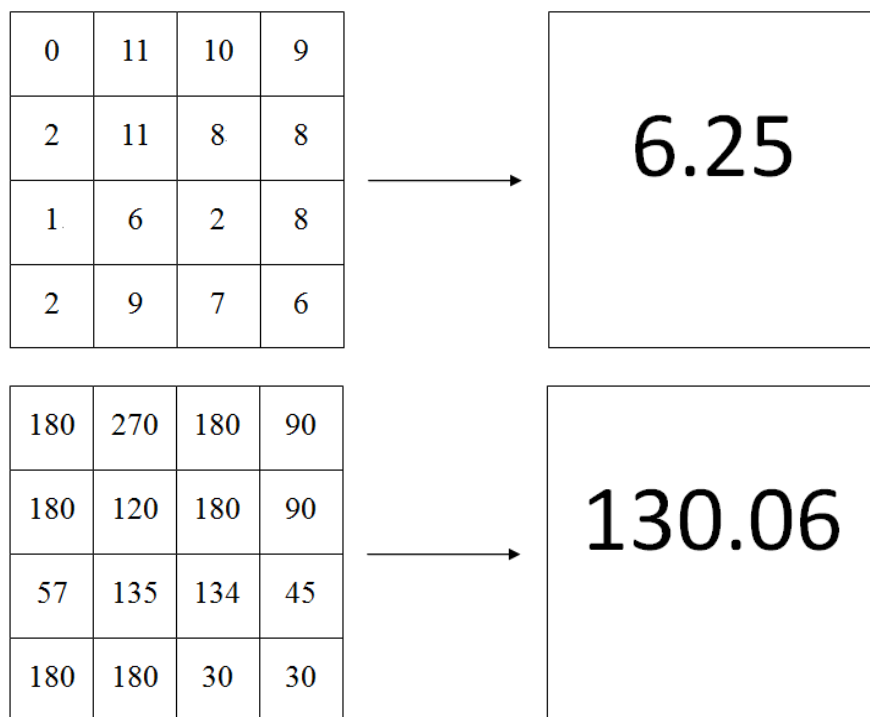
There are three types of GIS data used in DEVS-FIRE: fuel, aspect and slope. Different types of GIS data require different rules to transfer from high resolution to low resolution. In this paper, we follow the idea “dominant fuel, averaged aspect and slope” to generate the low resolution GIS data. We note that although these rules are arbitrary defined, they serve the purpose in our work to enable low resolution simulation for generating a profile of fire spreading behavior.

The idea of “dominant fuel” is to select the fuel type that has the maximum number of occurrence within an $R\times R$ partition in the high resolution cell space as the fuel type of the corresponding cell in the low resolution cell space, where R is the resolution factor. When there are several fuel types that have the equal maximum occurrence within the partition, we randomly choose one of them as the fuel type of the corresponding cell in the low resolution cell space. The idea of “averaged aspect and slope” is to use the average value of all the aspect/slope within the $R\times R$ partition as the slope or aspect of the

corresponding cell in the low resolution cell space. Figure 4 illustrates an example of this rule with $R=4$. In Figure 7.6 (a), the numbers represent the fuel types. On the left of Figure 7.6 (a), there are nine occurrences of fuel type 7 in the 4×4 sub cell space. So fuel 7 is chosen to be the fuel type in the corresponding cell of the low resolution GIS data. On the right, each fuel type has 4 occurrences so a randomly chosen fuel type, in this case fuel type 9, is used as the fuel type in the corresponding cell of the low resolution GIS data. In Figure 7.6 (b), the numbers on the left represent slope data and the numbers on the right represent aspects. On the left of Figure 7.6 (b), the slope sum of 16 cells is 100. So the average slope 6.25 is calculated to be the slope of the corresponding cell in the low resolution GIS data. The aspect is calculated in a similar manner.



(a) “dominant fuel”

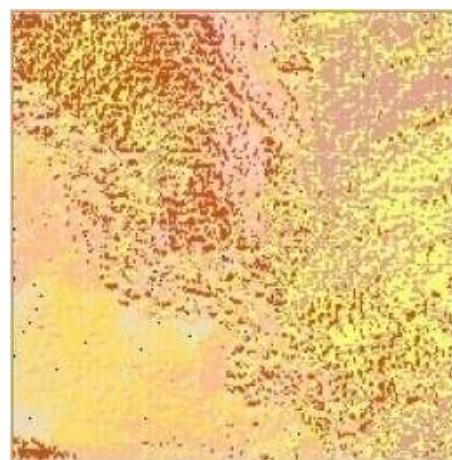
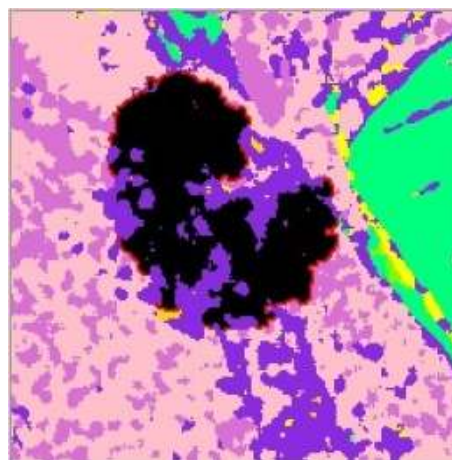
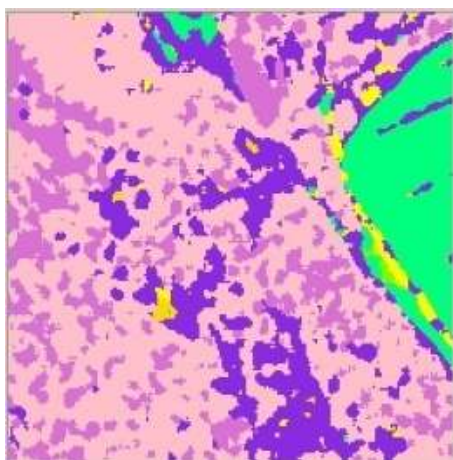


(b) “averaged aspect and slope”

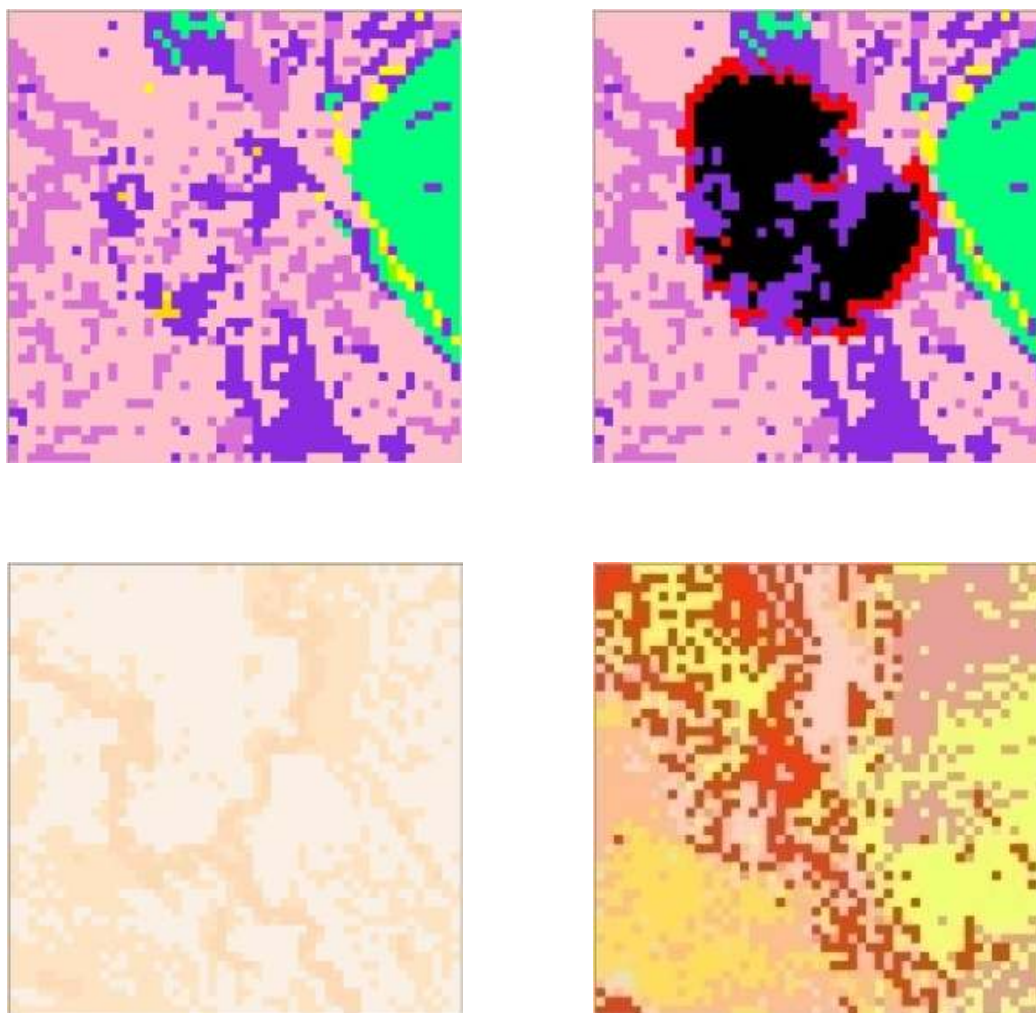
Figure 7.6 Examples of “dominant fuel, averaged slope and aspect” rule.

Figure 7.7 shows an example of producing low resolution GIS data (with resolution factor $R=4$) and the simulation results using the high resolution and low resolution GIS data. The low resolution GIS data are displayed in Figure 7.7 (b), which are generated from a set of high resolution GIS data displayed in Figure 7.7 (a). In Figure 7.7 (a), the size of the cell space is 200×200 and the size of each cell is $2.5\text{m} \times 2.5\text{m}$. In Figure 7.7 (b), the size of the cell space is 50×50 and the size of each cell is $10\text{m} \times 10\text{m}$. From the figures we can see that the “dominant fuel, averaged aspect and slope” rule is effective in that the low resolution GIS data conforms to the high resolution GIS data in all three different types. We conducted two simulations using these two sets of GIS data. The ignition point of the low resolution simulation locates at (24, 24) and the ignition point of the high resolution simulation locates at (99, 99). Both simulations use the same wind speed and

wind direction (from south to north). The simulation results on both resolutions at time = 7200 seconds are displayed on the right in Figure 7.7 (a) and Figure 7.7 (b) respectively. The simulation results show that the fire shape of low resolution simulation is similar to that of high resolution simulation at the given virtual time. This means the low resolution simulation is able to provide a good profile about the fire spread behavior (in a much faster execution time) of the high resolution simulation. However, it is important to note that the low resolution simulation cannot replace the high resolution simulation because it trades the simulation accuracy for speed.



(a) High resolution GIS data and simulation result at time 7200s



(b) Low resolution GIS data and simulation result at time 7200s

Figure 7.7 An example using our “dominant fuel, averaged slope and aspect” algorithm

7.3.3 Produce profile queue

The second step of the profile-based spatial partitioning method is to produce a profile using the generated low resolution GIS data. To produce the profile, we use the low resolution GIS data to run a low resolution simulation via DEVS-FIRE on a single

machine. The low resolution simulation uses the same weather data as that in the high resolution simulation. The ignition point in the low resolution simulation is converted from that in the high resolution simulation based on the following formula, where x_{LR} and y_{LR} denote the x and y coordinates of the ignition point in the low resolution simulation, x_{HR} and y_{HR} denote the x and y coordinates of the ignition point in the high resolution simulation, and R is the resolution factor.

$$x_{LR} = x_{HR}/R; y_{LR} = y_{HR}/R$$

In the low resolution simulation, once a cell is ignited, we record a virtual ignited time (VIT) that indicates when this cell is ignited. If a cell cannot be ignited, its VIT is infinity. After the simulation is finished, we sort the VIT of all cells from small to large and use the sequence of sorted cells to represent the fire spread behavior. In our work, a queue is used to store the profile. Each item in the queue contains the x and y coordinates of the cell as well as its corresponding VIT . We name this queue as the profile queue (denoted as PQ). The profile queue includes two types of information: 1) the ignition sequence of the cells, and 2) the time of ignition of the cells. The profile queue is defined as follows,

$$PQ = \{(x_{LR}, y_{LR}, VIT)_i | 0 \leq x_{LR} < L_{LR}, 0 \leq y_{LR} < W_{LR}, 0 \leq VIT_i \leq VIT_j, 0 \leq i < j < L_{LR} W_{LR}\}$$

where the subscript i denotes the order of the cell in the queue, x_{LR} and y_{LR} are the coordinates of the cell, L_{LR} is the length of the low resolution cell space and W_{LR} is the width of the low resolution cell space. Figure 7.8 shows an example of how to produce the profile and PQ . The top part of the figure shows the cells in the cell space (on the left) and their ignition sequence (on the right) in the low resolution simulation. The bottom

part of the figure shows the profile queue based on the low resolution simulation. For simplicity, all cells' virtual ignition time are denoted as *VIT* in the figure.

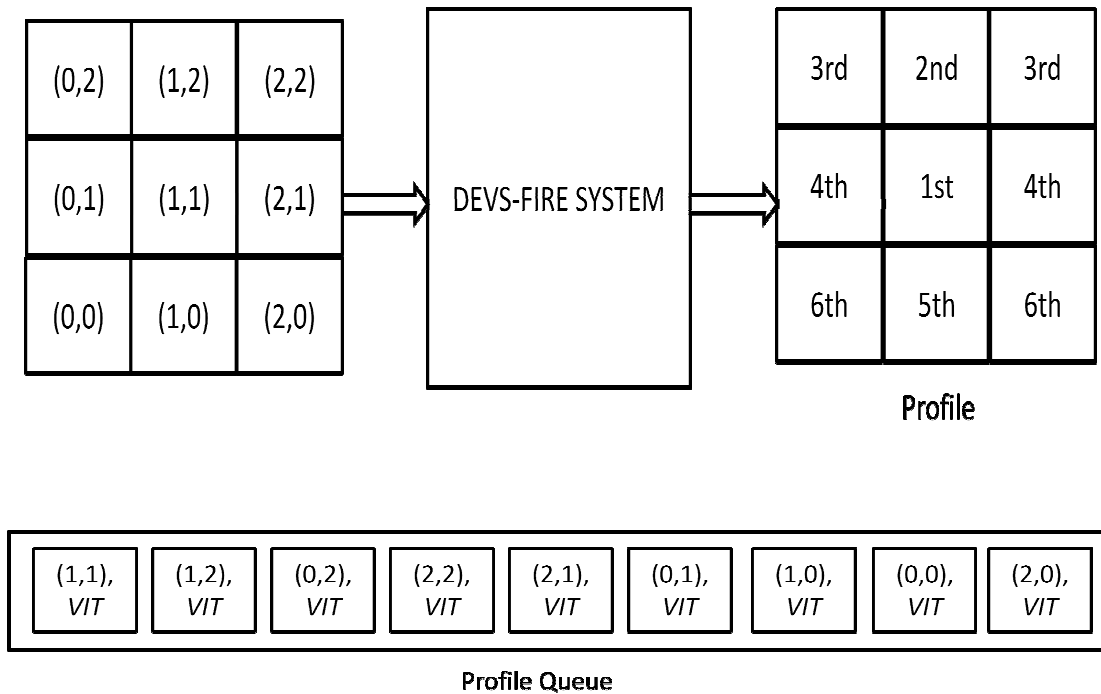


Figure 7.8 Producing profile using low resolution GIS data: the sequence in the *PQ* represents the order this cell is ignited

7.3.4 Profile-based partition with spatial granularity

To support parallel simulation, the cellular space needs to be divided among multiple PUs. In the profile-based spatial partitioning, the cellular space is divided according to the profile queue (*PQ*) generated in the previous step. The basic idea is to allocate the items in *PQ* sequentially to the PUs. Note that each item in *PQ* represents a block of cells (specified by the resolution factor *R*) in the high resolution cell space. Allocating the items individually can effectively support load balance among the PUs.

However, it may not achieve optimal result because each item is a relatively small block (especially when the resolution factor R is small). Dividing the space according to these small blocks will result in frequent message passing among the PUs and thus lead to high communication cost and cause more rollbacks in the time warp algorithm. To increase the block size, we introduce a spatial granularity factor (denoted as g) that allows the user to specify the granularity of partitioning.

With the granularity g , the partitioning method allocates a $g \times g$ sub cell space to a PU each time according to the profile queue. Figure 7 illustrates examples of partitioning with different granularities. Figure 7.9 (a) shows the ignition sequence of the cells in a 6×6 cell space. If the granularity g equals to 1, the partitioning process of the cells strictly follows the profile: the first ignited cell is allocated to $PU1$, the second ignited cell is allocated to $PU2$, ..., the n^{th} ignited cell will be allocated to PU_N (where N is the total number of PUs), and the $(n+1)^{\text{th}}$ ignited cell will be allocated to $PU1$ again, and so on. Figure 7.9 (b) illustrates the partitioning process when the granularity g equals to 2. In this case, the ignition order of each 2×2 sub cell space is decided by the next earliest ignited cell it contains. For example, the 2×2 sub cell space with “1st” on it contains the first ignited cell; thus this sub cell space will be allocated to $PU1$. After that, since the second and the third ignited cells have already been assigned to $PU1$, the 4^{th} ignited cell will be the next earliest ignited cell. Thus, the 2×2 sub cell containing this cell is marked as “2nd” and is allocated to $PU2$. This continues until the whole cell space is finished. Figure 7.9 (c) illustrates the partitioning sequence when the granularity g is 3 and Figure 7.9 (d) illustrates the partitioning sequence when the granularity g is 4.

12 th	16 th	24 th	29 th	34 th	36 th	4 th	7 th	9 th
9 th	13 th	21 st	26 th	32 nd	35 th			
6 th	10 th	18 th	22 nd	30 th	33 rd	2 nd	5 th	8 th
4 th	7 th	15 th	19 th	27 th	31 st			
2 nd	5 th	11 th	17 th	23 rd	28 th	1 st	3 rd	6 th
1 st	3 rd	8 th	14 th	20 th	25 th			

(a) $g=1$ (b) $g=2$

						2 nd	4 th
						1 st	3 rd

(c) $g=3$ (d) $g=4$

Figure 7.9 Examples of partitioning with different granularities

We define the profile queue with granularity (denoted as PQG) as the adjusted profile queue according to the granularity described above. Given a PQ and the granularity g , the algorithm of producing PQG is given below.

Algorithm (Producing PQG)

Input: profile queue PQ ; granularity g ; dimension of the low resolution cell space W_{LR} and L_{LR} ; number of participating PUs N .

Output: profile queue with granularity PQG .

$n \leftarrow 0$;

for each $item$ in PQ , **do**

if $item$ is not allocated, **do**

$h_start \leftarrow (item.x / g) * g$

$h_end \leftarrow h_start + g$

if $h_end > L_{LR}-1$, **do**

$h_end \leftarrow L_{LR}-1$

end if

$v_start \leftarrow (item.y / g) * g$

$v_end \leftarrow v_start + g$

if $v_end > W_{LR}-1$, **do**

$v_end \leftarrow W_{LR}-1$

end if

for each $item$ in PQ , **do**

```

    if h_start <= item.x < h_end and v_start <= item.y < v_end, do
        PQ.remove(item);
    PQG.push_back(item, n)
    end if
end for
n ← (n+1) % N
end if
end for
return PQG

```

After partitioning *PQG*, each PU obtains a sub set of *PQG*. Since each item in the *PQG* represents a block of cells in the high resolution cell space, the sub sets of *PQG* are then converted to the corresponding high resolution cells. These are the spatial partition for the PUs.

7.3.5 Parallel simulation based on the profile-based spatial partition

After the spatial partitioning, we run parallel simulation using the time warp algorithm for DEVS models [72]. Under a single PU environment, when an event that represents the fire spreading from one cell to another is scheduled by the simulator in DEVS-FIRE, an ignition message will be sent from the burning cell to the unburned cell. This is achieved by inserting the ignition message into the recipient's input message list in time stamp order. When the simulation is initiated under a parallel environment, an ignition message can be destined to a cell that is not located on the same PU. Thus, we

divide the ignition messages into two categories: inter-messages and intra-messages. For a cell, intra-messages are sent from its neighbor cells that locate on the same PU. Since both the source and destination are on the same PU, intra-messages are directly inserted into the target cell's input messages list. Inter-messages are needed when the source cell and destination cell are not on the same PU. When an inter-message is initiated, it is sent out from the source cell to the destination cell and it is up to the destination cell to process this inter-message. When an inter-message is received, it will first be stored in a message queue (*MQ*) which holds all the inter-messages a PU receives. The *MQ* keeps all its messages in ascending order based on their timestamps (*TS*s). That is the message that is at the head of the *MQ* has the smallest *TS* and the message that is at the tail of the *MQ* has the largest *TS*.

Since we are using time warp algorithm, rollbacks will be produced. To reduce the rollbacks, we propose a mechanism called watch dog in this paper. The basic idea of the watch dog is to exploit the *VIT* in the profile as a “watch dog” to screen the incoming inter-messages (sent from other PUs). The goal is to use the *VIT* to distinguish “out-of-order” ignition messages, that is, the messages that will cause rollbacks, and postpone processing them. This is based on the assumption that the *VIT* obtained in the low resolution simulation holds some “truth” about the high resolution simulation and thus can be utilized. Below we describe how the watch dog works. Because a cell in the low resolution simulation represents a block of cells ($R \times R$ cells) in the high resolution simulation, in our method the *VIT* obtained in the low resolution simulation is shared by all the cells in the $R \times R$ block. This time is denoted as $T_{\text{watch_dog}}$. The watch dog mechanism is checked only when receiving an inter-message sent from other PUs.

Specifically, when a cell receives an inter-message, it checks if any cell within its $R \times R$ block is already ignited. If yes, the message is processed. This is because an ignited block has already processed inter-messages before and thus there is no need to hold follow-on inter-messages. However, if this block has not been ignited, the TS associated with the incoming message is compared with $T_{\text{watch_dog}}$. If $TS > T_{\text{watch_dog}}$, this means other inter-messages (from other PUs) that contain a timestamp smaller than TS are likely to be received later (because the low resolution simulation indicates the block should be ignited earlier). If we process this message to ignite the block, it may cause rollbacks when those messages with smaller TS s are received. Thus the watch dog mechanism holds this message without processing it. We note it is possible that the $T_{\text{watch_dog}}$ obtained from the low resolution simulation is inaccurate. Thus to ensure the correctness of the parallel simulation, the held messages are not discarded and are stored in the MQ for later processing. As the simulation proceeds, when the local simulation time of the PU reaches a held message's ignition time, the held message will be re-processed. Therefore, even if the $T_{\text{watch_dog}}$ obtained from the low resolution simulation is not precise, the watch dog mechanism will not cause errors in the parallel simulation. In the extreme case, all of the real ignition messages may have the TS s all greater than the $T_{\text{watch_dog}}$ of the corresponding block, which may cause the simulation to be stalled. At this situation, we compare the smallest TS with the Global Virtual Time (GVT). If this TS is equal to the GVT , we will accept this message because it is time to process it. To update the GVT , we followed the algorithm in [73]. When an inter-message is sent or received, we update local Message Matrix (MM) and Table of Forcing Vectors (TFV) according to the steps in "Update on Send" or "Update on Receive" specified in [73]. If it is time to calculate

GVT , the minimum of all the $lvts$ and the ts in all the forcings in the TFV is selected as the GVT . To focus on the watch dog, we omit the detail of GVT update part in the description of the algorithm. Next, we provide the algorithm of the simulation with the watch dog mechanism. In the algorithm, $WATCHDOG[lx][ly]$ represents the virtual ignition time $T_{\text{watch_dog}}$ obtained from the profile for partition block at (lx, ly) , $message.TS$ are the time stamps associated with the message. More information about the time warp algorithm for DEVS models can be found in [72].

Algorithm (watch dog mechanism)

Input: MQ ; dimension of the low resolution cell space W_{LR} and L_{LR} ; resolution factor R ;

while terminating condition is not met, **do**

// process messages in the message queue MQ . Note that the MQ contains not only new received messages, but also previously received “out-of-order” messages that might be screened out by the watchdog mechanism.

for each $message$ in MQ , **do**

if $message$ has not been used for GVT , update TFV and MM according to [73];

$lx \leftarrow message.x/R$;

$ly \leftarrow message.y/R$;

if partition block at (lx, ly) is ignited or $message.type$ is ROLLBACK, **do**

 remove $message$ from MQ and insert $message$ into the local target cell($message.x$, $message.y$)’s input messages list;

else if partition block at (lx, ly) is not ignited, *do*

if $message.TS == GVT$ *or* $message.TS \leq WATCHDOG[lx][ly]$ *or* $message.TS \leq LVT$, *do*

remove *message* from *MQ* and insert *message* into the local target cell(*message.x*, *message.y*)’s input messages list;

else

put *message* back into the *MQ*;

end if

end if

end for

get all the imminent;

for each imminent *imm*, *do*

run the simulation using the time warp algorithm for DEVS models [72] on each *imm*.

Specially, when sending out a message, it includes sending intra-messages to the cells on the same PU, inter-messages to the cells not on this PU. When sending messages to another PU, MM and TFV are updated and then piggy-backed in the inter-messages according to [73]. If this cell is ignited, the partition block that contains this cell is marked as ignited. When a collision between the external transition function and the internal transition function happens to the *imm*, the confluent function occurs, in which the internal transition function executes first and then the external transition function executes.

// When imminents execute output functions, the output messages are transferred to be the input messages of some models. These models may be in imminent or may not.

The models that are not in imminent are called influenced models.

get all the influenced models;

for each influenced model, *do*

run the simulation using the time warp algorithm for DEVS models [72] on the influenced model.

Reschedule the imminents and schedule the influenced models;

update *LVT* by using the smallest *lvt* of all the scheduled models;

update this PU's *lvt* in TFV using *LVT*

if it is time to update *GVT*, do the calculation specified in [73];

end while

In the above algorithm, we note that although multiple blocks may be partitioned together based on the granularity factor g , in the algorithm each block still uses its own $T_{\text{watch_dog}}$. In other words, the granularity factor g acts as a partitioning factor only. It does not mean the different blocks in the same partition will use the same $T_{\text{watch_dog}}$. We also note that the watch dog mechanism can only alleviate the problem of rollback to some extent. There are other (important) situations that the watch dog cannot prevent rollbacks.

7.4 Experiment results

In this section, several experiments based on different measurements are developed to test performance of the parallel simulation using both the uniform spatial partitioning method and the profile-based spatial partitioning method. Based on the section 5, we

consider several factors in our experiments: including number of PUs, granularity g and the initial locations of ignition points. We choose the values of these factors in order to demonstrate the advantages and limitations of the profile-based partitioning method. For example, we vary the ignition point location and the granularity factor g in different experiments to show how they affect the performance results. The experiment environment is on Cheetah, a Linux cluster with five nodes and four GPUs, 18.0G memory and CentOS release 5.4 (Final). The software employed in the experiments are mvapich2/gnu [74] and adevs-2.1 [75]. We implemented the time warp algorithm for DEVS models specified in [72] on top of adevs-2.1 by using MPI as our parallel mechanism. We also implemented our own version of [73] to update *GVT* on top of the time warp algorithm [72]. All the parallel simulations are conducted on a 1000×1000 cell space. In all of our experiments, the wind speed is constant and the wind direction is from south to north. The simulations stop when all the burnable cells in the 1000×1000 cell space are burned out. Each simulation is conducted five times, and the average of the experiment results is calculated. In all the experiments using the profile-based spatial partitioning method, the resolution factor is set to 10, which allows us to generate a reasonably meaningful profile in a fast time compared to the high resolution simulation. This means the low resolution simulation for producing the profile is based on a cell space with dimension 100×100 . The *GVT* is updated every ten seconds. The execution time results for the profile-based partitioning also includes the time spent on the low resolution simulation.

7.4.1 Simulations with different ignition locations and multiple ignitions points

This section shows how the fire spread behavior can influence the simulation performance for the two spatial partitioning methods. To produce different fire spread behavior we vary the location and the number of the ignition points in the wildfire spread simulations. For the profile-based spatial partitioning method, granularity g is set to 10. Together with the resolution factor of 10, this means each partitioning block for the spatial partitioning method corresponds to 100×100 cells in the high resolution simulation. Four PUs are used for both the uniform spatial partitioning method and the profile-based spatial partitioning method.

The first set of simulations (shown in Figure 7.10) tests the impact of the ignition location on the simulation performance when there is only one ignition point. In this set of simulations, the locations of the only ignition points are set at $(0, 0)$, $(249, 249)$ and $(499, 499)$ respectively. Figure 7.10 (a) illustrates the ignition points in the cell space under the uniform spatial partitioning method. When the ignition point locates at $(0, 0)$, it is far away from the partitioning boundary. When the ignition point locates at $(499, 499)$, it is right at the partitioning boundary. When the ignition point locates at $(249, 249)$, it is at the center of the partition on $PU0$. Figure 7.10 (b) shows the ignition point in the cell space under the profile-based spatial partitioning method. Compared to the uniform spatial partitioning, there are more partitions and each partition is much smaller. When the ignition point locates at $(0, 0)$, it is much closer to the partitioning boundary in Figure 7.10 (b) than that in Figure 7.10 (a). When the ignition point locates at $(249, 249)$ or $(499, 499)$, they are right at the partitioning boundary.

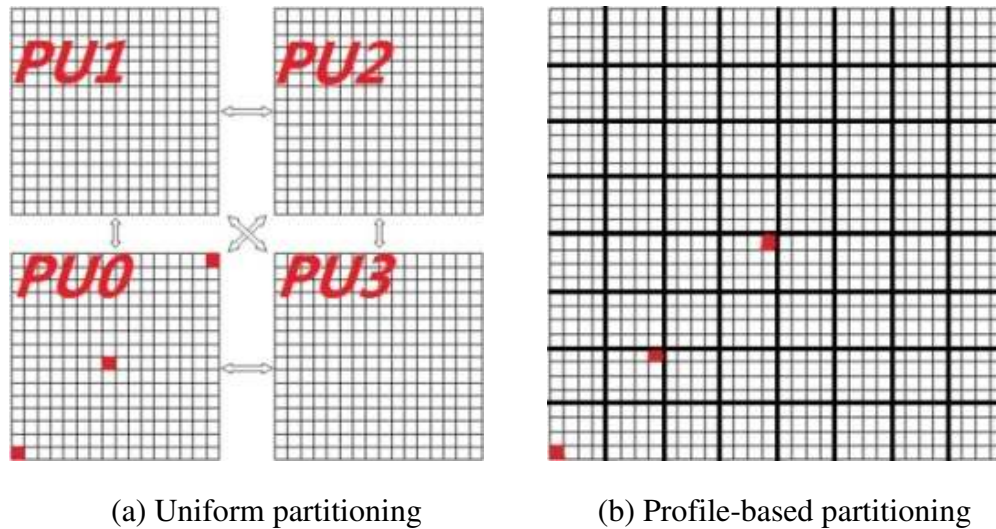


Figure 7.10 Different locations of the ignition point in the cell space. (a) (b)

Figure 7.11 shows the execution time (Figure 7.11 (a)) and the number of rollbacks (Figure 7.11 (b)) for both the profile-based spatial partitioning method and the uniform spatial partitioning method. For the profile-based spatial partitioning, the time spent on producing the profile and the partitioning process is also included in Figure 7.11 (a), which is only a small portion of the execution time as shown in the figure. To show the effect of the parallel simulations Figure 7.11 (a) also displays the simulation execution time when using a single-processor (denoted as 1PU in Figure 7.11 (a)). From the figure, we can see that when using the uniform spatial partitioning the execution time decreases as the ignition point moves towards the center of the space. This is expected because the center of the space is right to the partitioning boundary. The closer the ignition point locates to the partitioning boundary, the faster the fire spreads onto other partitions on other PUs. Different from that in the uniform spatial partitioning, the execution time of the profile-based partitioning is more stable when the ignition location changes. The

execution time for location (0, 0) is a little bit higher than those for the other two locations. This is because the other locations are right at the partitioning boundary thus it takes less time for the fire to spread to all four PUs. Figure 13(a) also shows when the ignition point locates at (0, 0) and (249, 249), the profile-based spatial partitioning has better performance (less execution time) than the uniform spatial partitioning. This is because both these two locations are far away from the partitioning boundary (see Figure 12(a)) for the uniform spatial partitioning. Thus the computation load is unbalanced in the first stage of the simulation for the uniform spatial partitioning because the fire exists in only one PU. When the ignition point is at (499, 499), the uniform spatial partitioning produces better performance result. This is mainly due to the number of rollbacks as explained below.

Figure 7.11 (b) shows the total number of rollbacks of all PUs for the two partitioning methods. The result confirms our analysis in Section 5 that the profile-based spatial partitioning produces more rollbacks than the uniform spatial partitioning does. When the ignition point locates at (0, 0) and (249, 249), the profile-based method produces about 80,000 more rollbacks in total than that of the uniform partitioning method. When the ignition point locates at (499, 499), the profile-based partitioning method produces about 120,000 more rollbacks than that of the uniform partitioning method. In this case, both partitioning methods lead to balanced work load among the PUs (see Figure 7.12 later). However, the less number of rollbacks make the uniform spatial partitioning has better performance. Table 7.1 lists speedup of both partitioning methods compared to the single-processor simulation for the different ignition locations. The speedup of the profile-based partitioning is stable for different ignition locations,

while the speedup of the uniform spatial partitioning varies as the ignition location changes.

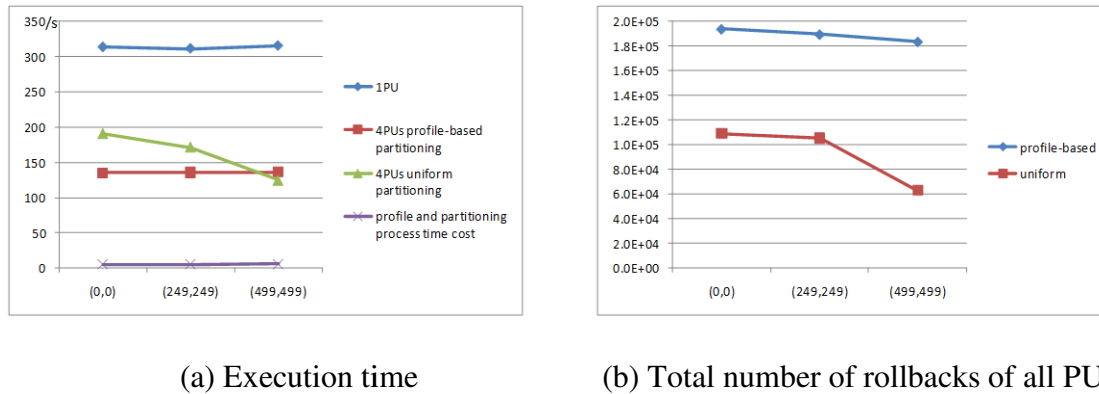


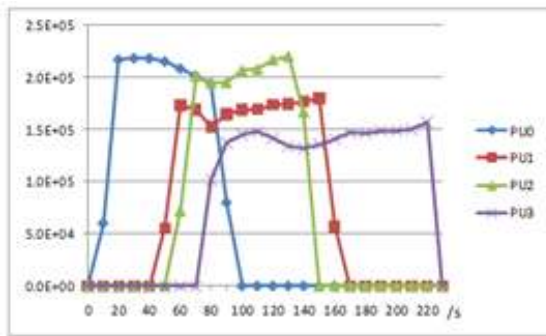
Figure 7.11 Experiment result

Table 7.1 The speed up of both partitioning methods

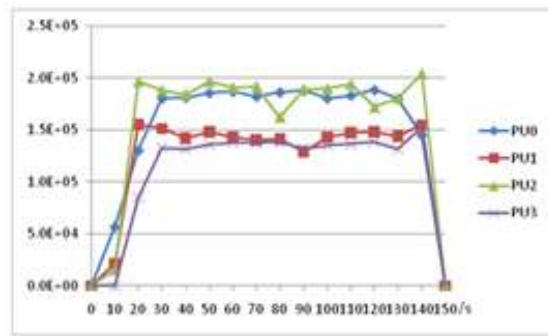
Ignition locations	(0, 0)	(249, 249)	(499, 499)
Profile-based spatial partitioning	2.41	2.48	2.46
Uniform spatial partitioning	1.65	1.82	2.68

To better examine the execution time and its relationship with the load balance, Figure 7.12 shows the work load of the two methods. The horizontal axis represents the execution time and the vertical axis represents amount of computation (calculated as the number of simulation iterations) during every time interval of 10 seconds. The figure confirms that for the profile-based partitioning, all four PUs participated in the computation from the very beginning and lasted to the end of the simulation. However, for the uniform spatial partitioning, the PUs have balanced work load only for the case of location (499, 499). In the two other cases, different PUs started and ended their computations at different time points. For example, in Figure 7.12 (a), in the first 40

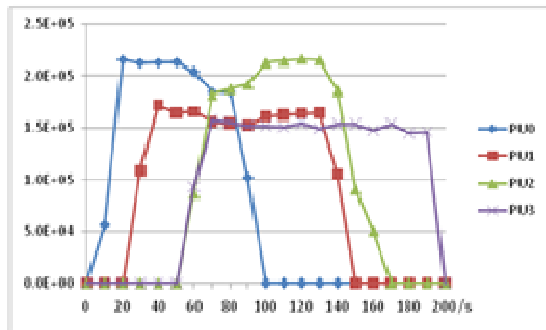
seconds only PU0 was active and then from time 170s, only PU3 was active while all other PUs were idle. The profile-based partitioning method is superior to the uniform spatial partitioning method in temporally balanced workload as shown in Figure 7.12, and thus leads to better performance results.



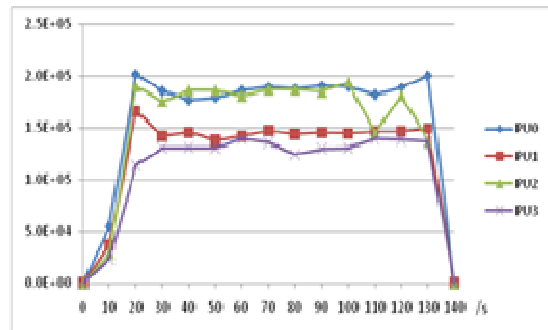
(a)



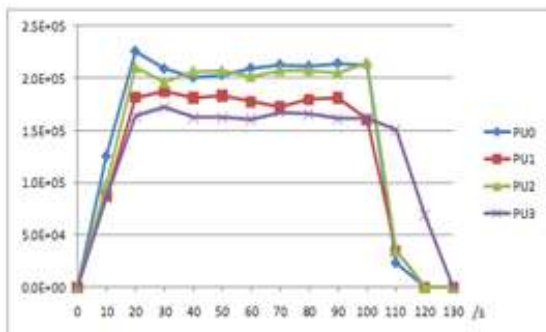
(d)



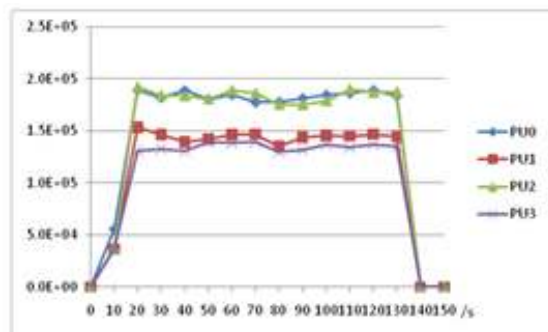
(b)



(e)



(c)



(f)

Figure 7.12 Work load diagrams. (a) uniform partitioning and ignition point at (0,0) (b) uniform partitioning and ignition point at (249,249) (c) uniform partitioning and ignition point at (499,499) (d) profile-based partitioning and ignition point at (0,0) (e) profile-based partitioning and ignition point at (249,249) (f) profile-based partitioning and ignition point at (499,499)

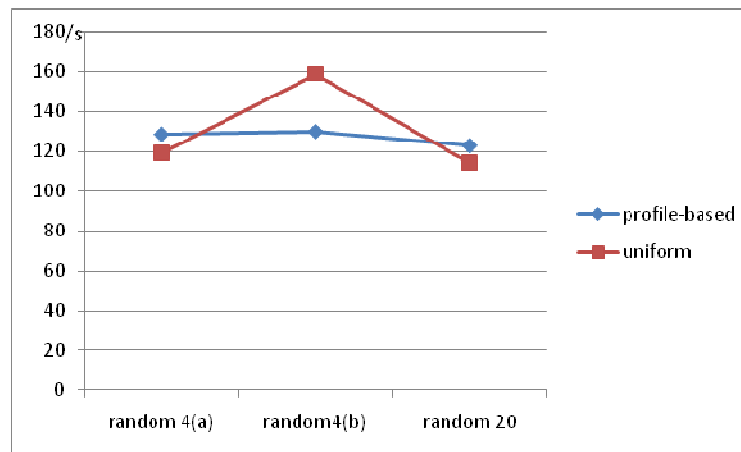
To further examine how fire spread behavior can influence the simulation performance, our second set of simulations use multiple ignition points. We keep all the other configurations same as before, except that the number of the ignition points is increased. The locations of the ignition points in this set of simulations are randomly generated. Table 7.2 shows the number and locations of the ignition points. In random 4(a) and random 4(b), four ignition points with random locations are used. In random 20, twenty ignition points with random locations are used. If the uniform spatial partitioning method is used, there is one ignition point on each PU for the situation random 4(a). For the situation random 4(b), the four ignition points locate on one PU. For the situation random 20, there is at least one ignition point on each PU. If the profile-based partitioning method is used, each PU will obtain at least one ignition point in all three situations in Table 7.2. Figure 7.13 shows the experiment results of all three situations using both partitioning methods.

Table 7.2 Locations of the ignitions

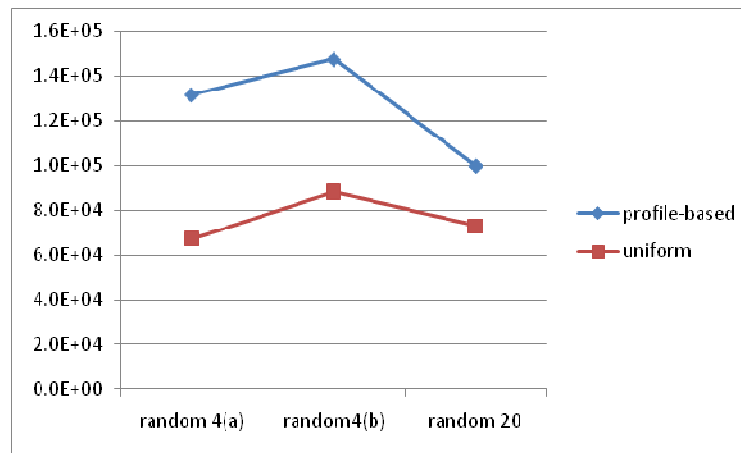
Name	Ignition Locations
random 4(a)	(388, 595), (148, 102), (993, 870), (997, 463)
random 4(b)	(18, 26), (487, 17), (19, 387), (492,

	267)
random 20	(419, 30), (431, 114), (127, 125), (798, 860), (133, 174), (311, 513), (979,592), (155, 701), (691,810), (779,277), (531, 289), (186, 879), (342, 210), (946, 521), (207, 853), (261, 626) , (236, 44), (92, 715), (521, 890), (576, 654)

From Figure 7.13, one can see that the uniform spatial partitioning method has slightly better performance when there is at least one ignition point on each PU as shown in random 4(a) and random 20. This is because in these cases all PUs can start the simulation at the very beginning and achieve high parallelism. For the case random 4(b) where all four ignition points locate in one partition, the execution time of the uniform spatial partitioning increases and is larger than that of the profile-based partitioning. This is due to the poor parallelism as explained before. On the contrary, the performance of the profile-based partitioning is stable as the ignition points vary. This set of experiments confirms with the previous experiments that the profile-based partitioning was able to achieve good parallelism independent of the fire spread behavior.



(a) Execution time



(b) Total number of rollbacks of all PUs

Figure 7.13 Experiment results

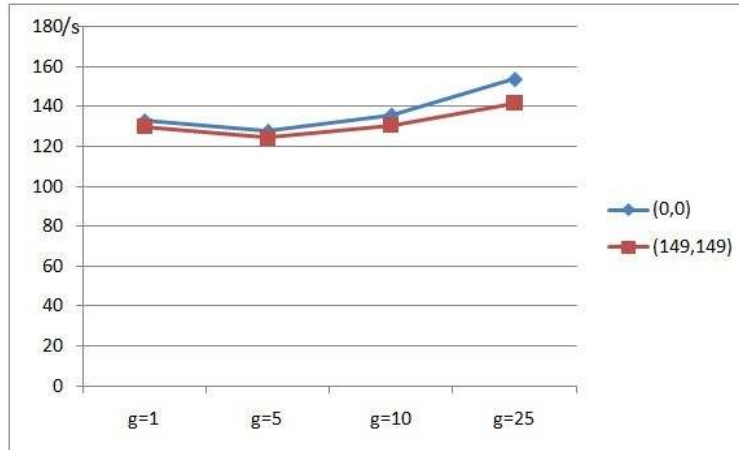
7.4.2 Simulations with different granularities

This experiment varies the granularity g and shows the impact of this factor on the performance of the profile-based spatial partitioning method. Similar as before, we use four PUs to run the parallel simulations. The granularity g is set to 1, 5, 10, and 25 respectively. We use only one ignition point and its location is set at (0, 0) in the first case and (149, 149) in the second case. Figure 16 shows the execution time and the number of rollbacks as the granularity g increases.

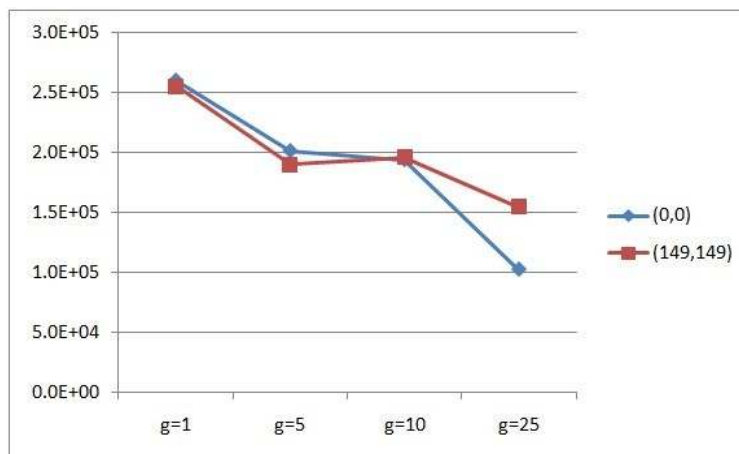
For the first case where the ignition point locates at (0, 0), Figure 16(a) shows that the execution time is reduced when g increases from 1 to 5. The total number of rollbacks is also reduced as g increases from 1 to 5. When $g=1$, each partition block is a 10×10 ($R * g = 10$) sub cell space. When $g=5$, each partition block is a 50×50 ($R * g = 50$) sub cell space. Since the fire is ignited at (0,0), it takes more time for the fire to spread out of a 50×50 cell space (thus to ignite other partitions in other PUs) than that of a 10×10 cell

space. However, as analyzed the number of rollbacks will be reduced when the granularity g increases. Figure 7.14 (b) shows that the number of rollbacks decreases about 50,000 when g increases from 1 to 5. Thus, even though it takes more time for the fire to spread out of a 50×50 cell space, the decrement of rollbacks dominates the time difference and makes the execution time reduced when g increases from 1 to 5. When g further increases to 10, the size of each partition block increases to 100×100 ($R * g = 100$) while the number of rollbacks only slightly drops a little. In this case, the time spent on spreading out of the 100×100 sub cell space dominates. Therefore, the execution time increases as g increases from 5 to 10. When g is 25, the size of each partition block is 250×250 ($R * g = 250$). The number of rollbacks drops about 100,000. However, it takes longer time for the fire to spread out of the partition block and this time dominates. As a result, the execution time increases too.

For the second case where the ignition point locates at (149, 149), the result curves are similar to those in the first case but with less execution time. This is because when g is 1 and 5, ignition point (149, 149) locates at the partitioning boundary. The fire will immediately spread onto other partitions on other PUs. When g is 10 and 25, the ignition point (149, 149) locates much closer to the partitioning boundary than in the first case. Thus, it takes less time for the fire originating from the ignition point (149, 149) to reach the partitioning boundary, which leads to less execution time as shown in Figure 7.14 (a).



(a) Execution time



(b) Total number of rollbacks of all PUs

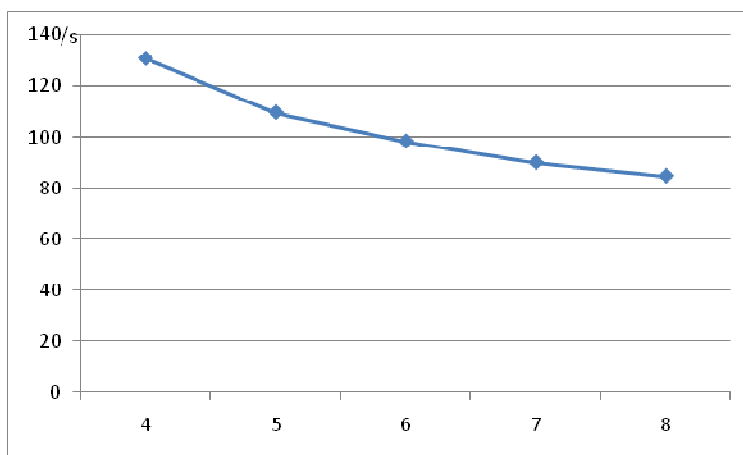
Figure 7.14 Experiment results

This experiment shows that the granularity g can have impact on the simulation performance. Different g leads to different partitioning results. In general, as g increases, the size of the partition block increases. This means it takes longer time for a fire to spread to other partition blocks. Thus it increases the simulation execution time due to unbalanced work load, e.g., one PU is active while others are waiting for the fire to

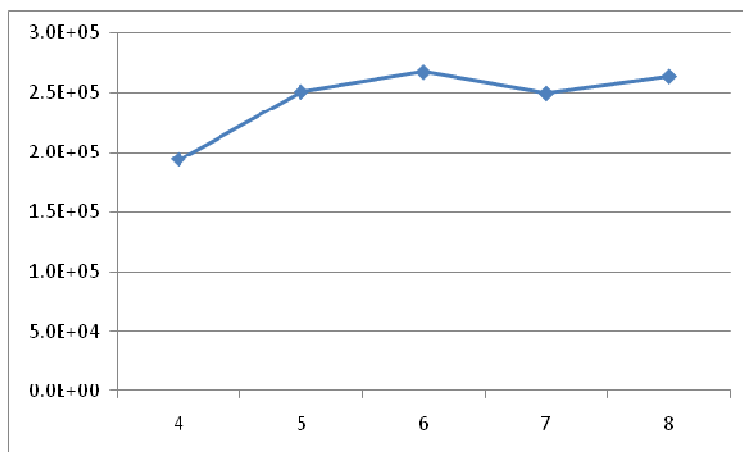
spread to their partitions. On the other hand, when g increases, the number of rollbacks reduces because the number of message passing among PUs is reduced.

7.4.3 Simulations with different number of PUs

This experiment varies the number of participating PUs to test the scalability of the profile-based spatial partitioning method. In these simulations, the resolution R is 10 and granularity g is 10. Thus, the 1000×1000 cell space is divided into one hundred 100×100 blocks. The ignition point is set to $(0, 0)$. The number of participating PUs varies from 4 to 8. As a result, each PU will obtain the same amount of cells when the number of participating PUs is 4 and 5. When the number of participating PUs is 6, 7 and 8, the load is a little unbalanced on each PU. The load difference is 10,000 (one 100×100 block) cells. In this experiment, we only consider the profile based method. This is because the uniform spatial partitioning method works best when the number of PUs is k^2 (k is an integer) [76] or there are ignitions in each partition after the uniformly spatial partitioning (see section 5). When there is only one ignition point, the execution is nearly linear: PU containing the ignition point will run first and all other PUs will wait until an ignition message is received. Figure 7.15 show the experiment results, from which we can see that the execution time decreases as the number of participating PUs increases. Table 3 lists the speedup of the parallel simulations compared to a single-processor simulation.



(a) Execution time



(b) Total number of rollbacks of all PUs

Figure 7.15 Experiment results

Table 7.3 Speedup of different number of PUs

	4	5	6	7	8
speedup	2.41	2.86	3.19	3.49	3.70

Although the execution time decreases as the number of PUs increases, the curve of decreasing becomes less steep as shown in Figure 7.15 (a). This is partially due to the

nature of wildfire spread behavior and how the profile-based partitioning method works. Based on the profile-based spatial partitioning method, some PUs will obtain their first partitions that are close to the ignition points; while other PUs will obtain their first partitions having a distance from the ignition point. As the number of PU increases, the distance between the ignition point and the first partitions of some PUs will increase as well. Together with the nature of fire spread behavior (a partition has no computation until it is ignited); some PUs will have to wait until the fire ignites their first partitions to start their simulations. The larger the number of PUs, the more PUs will wait and the longer time they will wait. To the extreme case when the number of PUs is equal to the number of partitions, the execution of the simulation is nearly sequential. This is no better than running on a single machine.

7.4.4 Simulations with “watch dog” mechanism

In this section we examine the “watch dog” mechanism described in section 4.4 for reducing rollbacks. In this experiment, we employ two ignition points as shown in Figure 7.16: one at (249,249) and the other at (999,999). Four PUs are used. For the profile-based spatial partition, the resolution R is set to 10 and granularity g is set to 5. In our experiment, $LIMIT$ is set to 20. We carry out the experiments using simulations with the watch dog mechanism and simulations without the watch dog mechanism and compare their experiment results.

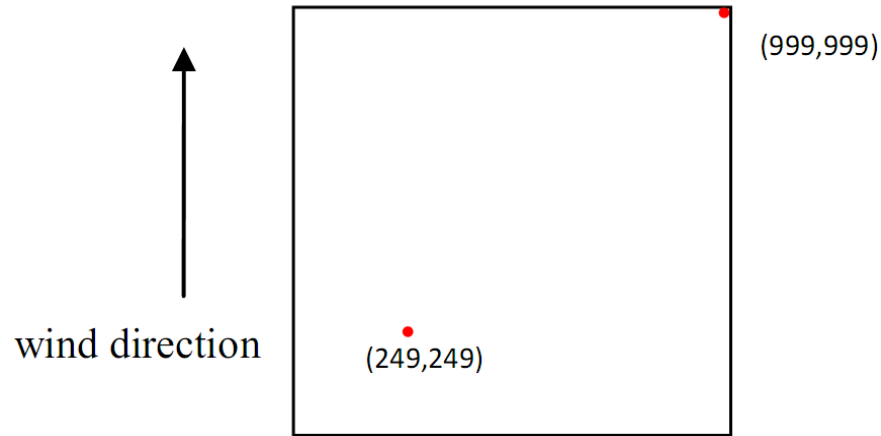
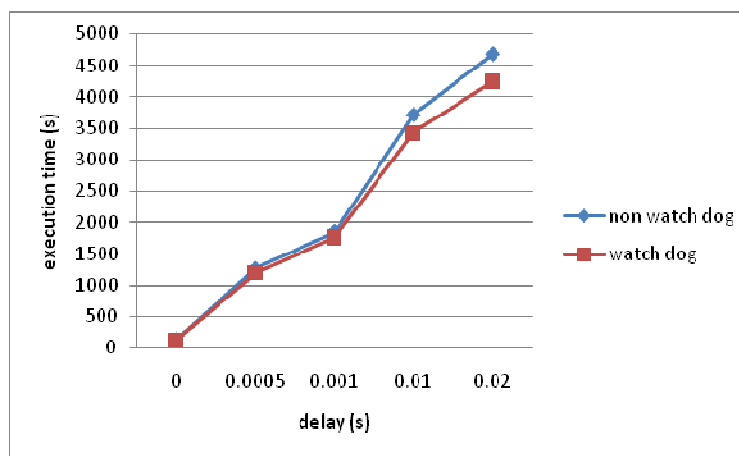


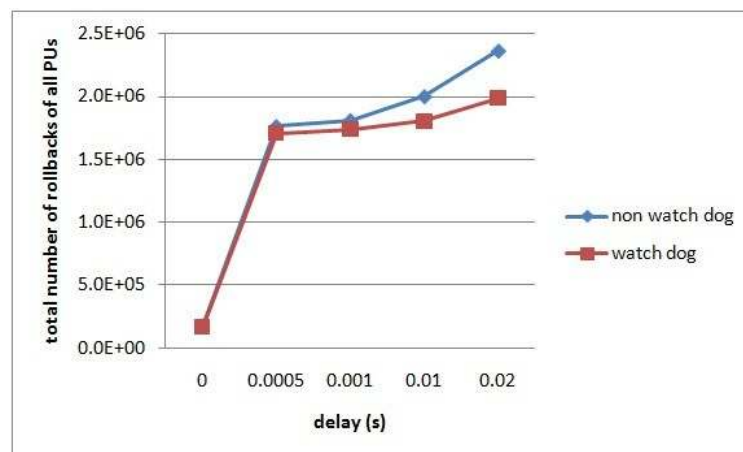
Figure 7.16 Experiment scenario for “watch dog” mechanism

Since the wind direction is from south to north, the burning area originating from ignition point (249,249) should be larger than that originating from ignition point (999,999) (because the fire is spreading to north). However, in a parallel simulation environment, the burning area originating from ignition point (999, 999) can be larger than the burning area originating from ignition point (249, 249) if ignition point (999, 999) is assigned to a faster PU and ignition point (249, 249) is assigned to a slower PU. If that is the case, when the burning area originating from ignition point (249, 249) eventually encounters the burning area originating from ignition point (999, 999), rollbacks will be triggered to ensure the correctness of the parallel simulation. The more area mistakenly “burnt” originating from ignition point (999, 999) which otherwise should be burnt originating from ignition point (249, 249), the more rollbacks will be triggered later on when the two burning areas encounter. When the watch dog mechanism is implemented, each partition block can check the ignition time of the incoming ignition message and compare it with the watch dog time $VIT_{\text{watch_dog}}$ obtained from the low resolution simulation to screen out the potential “out of order” incoming ignition messages. To

better demonstrate this effect of the watch dog mechanism, we make the PU containing ignition point (999, 999) faster than other PUs by adding a delay to all other PUs. The delay is achieved by the sleep system call and it is varied from 0s (which means no delay) to 0.02s. We measure the performance difference between the one using watch dog and the one without watch dog to see the effects of watch dog when there is difference in computing speed among participating PUs. Figure 7.17 shows the experiment results.



(a) Execution time



(b) Total number of rollbacks of all PUs

Figure 7.17 Experiment results

From Figure 7.17 we can see that the simulation performance is improved by using the watch dog mechanism. When the delay is zero, the watch dog nearly has little effect. This is because all the PUs have the same processing speed so that the fire originating from ignition point (999, 999) does not burn too much area. As the delay increases, more area is burnt by the fire originating from ignition point (999, 999). Thus, the number of rollbacks increases as well. After the watch dog is used, both the number of rollbacks and the execution time are reduced. The gaps between the two curves in both figures become larger when the delay increases. This indicates the watch dog screens out more “out of order” ignition messages. From this experiment we can see that the effect of the watch dog becomes more obvious when the PUs’ processing speeds vary largely.

CHAPTER 8 CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

Modeling and simulation is an abstraction of our complex physical world, which is a powerful tool for scientists and researchers to study the reality. During past decades, many modeling and simulation theories, formalisms, frameworks and tools have been proposed and developed. In the past, not only do those simulation systems and tools have to be physically delivered to scientists and researchers, but also it requires scientists and researchers to learn how to install, configure and use these simulation systems and tools. This extra work may cause distractions and unnecessary burdens to scientists and researchers since this is not their focus and should not stand in their way.

As technology advances, new computing paradigms such as service-oriented architecture and cloud computing bring new avenue to software distribution methodology. Infrastructure as a Service, Platform as a Service and Software as a Service are three musketeers that ease the way we do our computation and free us from the burden of managing infrastructure, platform and software ourselves. This also enlightens the realm of modeling and simulation. In this work, a layered framework to support simulation software as a service and service oriented simulation experiment is designed. Each layer utilizes the service provided by the layer below. Particularly, our work focuses on the experiment layer, the model layer and the engine layer. For the experiment layer and the model layer, specifications are proposed and designed, which provides general standards within this framework. At the model layer, we pay the attention to the composability issues when coupling multiple simulation services to construct a composed simulation service. Due to the fact that the simulation is complex dynamic system in which time

plays an important role, a coordinator service is developed for the composition of the simulation services to guarantee the correct temporal order of the events occurred in the composed simulation service. To achieve better composition of the simulation services, several similar simulation composability models are adopted. In particular, the mismatch of the time granularity and the event granularity issue existing in pragmatic level of the simulation service composability is investigated. To solve the mismatch and enhance the composition of the composed simulation service, four simulation agents are developed to fix the mismatch of the time granularity and the event granularity. From the experiment results, we can see those simulation agents can improve the composition of simulation services and improve the simulation performance only by sacrificing certain accuracy of the simulation which is bounded.

For the simulation engine layer, we proposed two spatial partition algorithms can improve the performance of an individual simulation, wildfire simulation, by taking advantage of parallel computing methodology. Both of these two proposed spatial partition algorithms have their advantages and disadvantages.

8.2 Future work

Future work should include improving the specifications of simulation software as a service and service-oriented architecture, which could better serve as the framework foundation. To better integrate into the cloud computing environment, simulation engines and simulation services need to be transferred to the platform that supports cloud computing. Another very important aspect that requires further research work is the composability of the simulation service at different levels. We believe that there are different types of mismatches and inconveniences when composing varied simulation

services, where require more simulation agents to enhance the compositions. It should be a big challenge to find a general way to develop those agents to balance the composed simulation accuracy and performance.

For the implementation part, we are going to improve the GUI for simulation service composition and service-oriented simulation experiment to make it have more powerful functionalities. One direction for the service-oriented simulation service GUI is to develop web browser-based BPEL designer in javascript, the equivalent to the eclipse BPLE designer plugin, to enable the user to design the experiment directly in the web browser. When deploying the experiment, the front end designed experiment will be translated into its BPEL equivalent, which will be deployed on the Apache ODE server for execution.

References

- [1] Cloud computing, http://www.wikinvest.com/concept/Cloud_Computing, Last accessed February 24, 2012.
- [2] SOA, <http://www.sun.com/products/soa/index.jsp>, Last accessed February 24, 2012.
- [3] WSDL2.0, <http://www.w3.org/TR/wsdl20-primer/>, Last accessed February 23, 2012.
- [4] SOAP1.2, <http://www.w3.org/TR/soap12-part0/>, Accessed February 23, 2012.
- [5] UUID, <http://www.ietf.org/rfc/rfc4122.txt>, Last accessed February 23, 2012.
- [6] SaaS, http://www.wikinvest.com/concept/Software_as_a_Service, Last accessed February 23, 2012.
- [7] PaaS, <http://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS>, Last accessed February 23, 2012.
- [8] IaaS, <http://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-a-Service-IaaS>, Last accessed February 23, 2012.
- [9] Workflow, <http://searchcio.techtarget.com/definition/workflow>, Last accessed February 23, 2012.
- [10] BPEL, <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf>, Last accessed February 23, 2012.
- [11] WSFL, <http://www.ibm.com/developerworks/library/ws-ref4/>, Last accessed February 23, 2012.
- [12] XLANG, <http://msdn.microsoft.com/en-us/library/aa577463%28v=bts.70%29.aspx>, Last Accessed February 23, 2012.
- [13] B. P. Zeigler, H. Praehofer, and T. G. Kim, Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems, 2nd ed. Waltham, Massachusetts, USA: Academic Press; 2000.

- [14] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler. DEVSML: automating DEVS execution over SOA towards transparent simulators. In *Proceedings of SpringSim*, pp. 287-295, 2007.
- [15] S. Mittal, DEVS unified process for integrated development and testing of service oriented architectures. Ph.D Thesis. United States -- Arizona: The University of Arizona, 2007.
- [16] S. Mittal, J. L. Risco-Martin, and B. P. Zeigler, DEVS/SOA: A cross-platform framework for net-centric modeling and simulation in DEVS unified process, *SIMULATION*, vol. 85, pp. 419-450, 2009.
- [17] B. P. Zeigler, Y. Moon, D. Kim, et al, DEVS/C++: A High Performance Modeling and Simulation Environment, *HICSS '96 Proceedings of the 29th Hawaii International Conference on System Sciences* vol. 1: Software Technology and Architecture, 1996.
- [18] DEVS-Java Reference Guide, <http://www.acims.arizona.edu/SOFTWARE/software.shtml>, Last accessed on May 4, 2011.
- [19] M. Zhang, B.P. Zeigler, P. Hammonds, DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies, *International Test & Evaluation Association (ITEA) Journal of Test and Evaluation*, vol. 27, no. 1, pp. 49-60, 2006.
- [20] H. Sarjoughian, S. Kim, M. Ramaswamy, et al, An SOA-DEVS modeling framework for service-oriented software system simulation, *Proc. of the Winter Simulation Conference*, 2008.

- [21] G. A. Wainer, R. Madhoun, and K. Al-Zoubi, Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-services, *Simulation Modeling Practice and Theory*, vol. 16, no. 9, pp. 1266-1292, 2008.
- [22] R. Madhoun, Web service-based distributed simulation of discrete event models, Ph.D Thesis, Canada: Carleton University (Canada), 2006.
- [23] NPS, Naval postgraduate school (NPS) moves institute: Extensible modeling and simulation framework (XMSF), <http://www.movesinstitute.org/xmsf>, Last accessed on February 1, 2007.
- [24] D. Brutzman, M. Zyda, M. Pullen, et al, Extensible modeling and simulation framework (XMSF) challenges for Web-based modeling and simulation, <http://www.movesinstitute.org/xmsf/XmsfWorkshopSymposiumReportOctober2002.pdf>, Last accessed on February 1, 2007.
- [25] HLA integration employing Web services for federate communications the extensible modeling & simulation framework (XMSF), <http://www.movesinstitute.org/xmsf>, Last accessed on February 1, 2007.
- [26] A. Buss, and J. Ruck, Joint modeling and analysis using XMSF Web services, *Proc. of the 2004 Winter Simulation Conference*, 2004.
- [27] M. M. Sohn, Advanced M&S framework based on autonomous web service in KOREA: Intelligent-XMSF approach, *Systems modeling and simulation: Theory and applications*, Berlin: Springer-Verlag Berlin, pp150-158, 2005.
- [28] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, The Department of Defense High Level Architecture, in *Proceedings of the 1997 Winter Simulation Conference*, 1997.

- [29] W. G. Wang, W. G. Yu, Q. Li, et al, Service-oriented high level architecture, in *Proceedings of the European Simulation Interoperability Workshop*, Edinburgh, Scotland: Simulation Interoperability Standards Organization.
- [30] K. L. Morse, D. L. Drake, and R. P. Z. Brunton, WMDOA integration employing Web services for federate communication-an XMSF exemplar, in *Proceedings of the Spring Simulation Interoperability Workshop*, Kissimmee, Florida: Simulation Interoperability Standards Organization, 2003.
- [31] K. L. Morse, D. L. Drake, and R. P. Z. Brunton, Web enabling an RTI-an XMSF profile, in *Proceedings of the Euro Simulation Interoperability Workshop*, Stockholm, Sweden: Simulation Interoperability Standards Organization, 2003.
- [32] B.Möller, and S. Löf, A management overview of the HLA evolved Web service API, in *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, Florida: Simulation Interoperability Standards Organization, 2006.
- [33] B.Möller, and S. Löf, Mixing service oriented and high level architectures in support of the GIG, in *Proceedings of the Spring Simulation Interoperability Workshop*, San Diego, California: Simulation Interoperability Standards Organization, 2005.
- [34] Levels of Information Systems Interoperability, C4ISR Architectures Working Group, 30 March 1998, available at: US DoD, OSD (C3I), CIO, Director for Architecture and Interoperability Website: <http://www.c3i.osd.mil/org/cio/i3/>, Last accessed on November 23, 2010.
- [35] NATO Allied Data Publication 34 (ADatP-34), NATO C3 Technical Architecture (NC3TA), Version 4.0, obtainable via the NATO standard website: <http://www.nato.int/docu/standard.htm>, Last accessed on November 23, 2010.

- [36] A. Tolk , and J. A. Muguira, The Level of Conceptual Interoperability Model, in *Proceeding s of the 2003 Fall Simulation Interoperability Workshop*, Orlando, FL, 2003.
- [37] H.Sarjoughian, Model Composability, in *Proceeding of the Winter Simulation Conference*, pp.149-158, 2006.
- [38] The OASIS Committee: Web Services Business Process Execution Language (WSBPEL) Version 2.0 (April 2007), <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf>, Last accessed on November 23, 2010.
- [39] B. Wassermann, W. Emmerich, B. Butchart, et al., Sedna: A BPEL-based environment for visual scientific workflow modeling, *Workflows for eScience -Scientific Workflows for Grids*, 2006.
- [40] Eclipse BPEL Designer, <http://www.eclipse.org/bpel/>, Last accessed on February 23, 2012.
- [41] ActiveBPEL Designer, http://www.activebpel.org/samples/samples-3/eclipseWTP_and_BPEL/doc/index.html, Last accessed on February 23, 2012.
- [42] Oracle BPEL Process Manager, <http://www.oracle.com/technetwork/middleware/bpel/overview/index.html>, Last accessed on February 23, 2012.
- [43] W. van der Aalst, and A. ter Hofstede, Yet another workflow language, *Information Systems* 30(4), pp, 245–275, 2005.
- [44] XML Process Definition Language, <http://www.xpdl.org/nugen/p/xpdl/public.htm>, Last accessed on February 23, 2012.

- [45] M. Lipp, The Danet Workflow Component V2.1 (2007), <http://wfmopen.sourceforge.net/>, Last accessed on February 23, 2012.
- [46] E.Deelman, and Y.Gil, Workshop on the Challenges of Scientific Workflows, Technical report, Information Sciences Institute, University of Southern California, 2006.
- [47] D. Hull, K. Wolstencroft, and R. Stevens, et al., Taverna: a tool for building and running workflows of services, *Nucleic Acids Research*, vol. 34, iss. Web Server issue, pp. 729-732, 2006.
- [48] T. Oinn, M. Addis, J. Ferris, et al., Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics* 20(17), pp. 3045–3054, 2004.
- [49] J.Zhao, C. Coble, M. Breenwood, et al., Annotating, linking and browsing provenance logs for e-Science, in *International Semantic Web Conference*, 2003.
- [50] T. Oinn, J. Ferris, J. Marvin, et al.: Delivering Web Service Coordination Capability to Users, in *WWW 2004*, New York, pp. 438-439, 2004.
- [51] B. Ludascher, L. Altintas, C. Berkley, et al., Scientific workflow management and the kepler system, *Concurrency and Computation: Practice and Experience*, vol. 18 number 10, pp. 1039-1065, 2005.
- [52] J.T. Buck, S. Ha, E.A. Lee, et al., Ptolemy: A framework for simulating and prototyping heterogeneous systems, *Journal of Computer Simulation*, vol. 4, pp. 155-182, 1994.
- [53] T. Dörnemann, T. Friese, S. Herdt, et al., Grid Workflow Modelling Using Grid-Specific BPEL Extensions, in *Proceedings of German e-Science Conference*, 2007.
- [54] X. Hu and B. P. Zeigler, A Proposed DEVS Standard: Model and Simulator Interfaces, *Simulator Protocol*, draft 1.0, 2008

- [55] P.K. Davis and R.H. Anderson, Improving the Composability of Department of Defense Models and Simulations, *RAND Corporation*, 2003.
- [56] M. Cefkin, S.M. Glissman, P.J. Haas, et al., SPLASH: A Progress Report on Building a Platform for a 360 Degree View of Health, in *Proceedings Of the 5th INFORMS Workshop on Data Mining and Health Informatics*, 2010.
- [57] L. Yilmaz and S. Paspuleti, Toward a Meta-Level Framework for Agent-Supported Interoperation of Defense Simulations, *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 2, no. 3, pp. 161-175, 2005.
- [58] D. Huang, Composable Modeling and Distributed Simulation Framework for Discrete Supply-Chain Systems with Predictive Control, Computer Science Engineering Department, Ph. D Thesis, Arizona State University, Tempe, AZ, 2008.
- [59] B. P. Zeigler, S. Mittal, and X. Hu, Towards a Formal Standard for Interoperability in M&S/System of Systems Integration, in *Proceedings GMU-AFCEA Symposium on Critical Issues in CAI*, 2008.
- [60] M. Xue, K.K. Droegemeier, and V. Wong, et al., The Advanced Regional Prediction System (ARPS) – A multi-scale nonhydrostatic atmospheric simulation and prediction tool. Part II: Model physics and applications, *Meteor. Atmos. Phys.*, vol. 76, pp. 143-166, 2001.
- [61] M. Xue, K. K. Droegemeier, and V. Wong, The Advanced Regional Prediction System (ARPS) – A multiscale nonhydrostatic atmospheric simulation and prediction tool, Part I: Model dynamics and verification, *Meteor. Atmos. Physics*, vol. 75, pp. 161-193, 2000.

- [62] L. Ntaimo, X. Hu, and Y. Sun, DEVS-FIRE: Towards an Integrated Simulation Environment for Surface Wildfire Spread and Containment, *SIMULATION*, vol. 85, no. 5, pp. 335-351, 2009.
- [63] C. Bettini, C.E. Dyreson, W.S. Evans, et al., A Glossary of Time Granularity Concepts, *Lecture Notes in Computer Science*, vol. 1399, 1998.
- [64] C. Bettini, X. S. Wang, and S. Jajodia, A General Framework for Time Granularity and Its Application to Temporal Reasoning, *Annals of mathematics and Artificial Intelligence*, vol. 22, pp. 29-58, 1998.
- [65] B.P. Zeigler, DEVS Theory of Quantization, DARPA Contract N6133997K-0007: ECE Dept., UA, Tucson, AZ, 1998.
- [66] M. Finney, FARSITE: Fire Area Simulator-model Development and Evaluation, Research Paper RMRS-RP-4, US Dept. of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, Utah, 1998.
- [67] P. Andrews, C. Bevins, and R. Seli, Behaveplus Fire Modeling System, Version 3.0: User's Guide, General Technical Report RMRS-GTR-106WWW Revised US Dept. of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, Utah, 2005.
- [68] M. Morais, Comparing Spatially Explicit Models of Fire Spread through Chaparral Fuels: A New Model based Upon the Rothermel Fire Spread Equation, Master's Thesis, The University of California, Santa Barbara, 2001.
- [69] X. Hu, Y. Sun, and L. Ntaimo, Design and application of formal discrete event wildfire spread and suppression models, *Simulation*, 2012.

- [70] R. Rothermel, A Mathematical Model for Predicting Fire Spread in Wildland Fuels, Research Paper INT-115. Ogden, UT: US Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1972, 40 p.
- [71] Y. Sun, and X. Hu, Partial-modular DEVS for improving performance of cellular space wildfire spread simulation, in *Proceedings 2008 Winter Simulation Conference (WSC'08)*, pp. 1038–1046, 2008.
- [72] J. Nutaro, On constructing optimistic simulation algorithms for the discrete event system specification, *Transactions on Modeling and Computer Simulation*, vol. 19, no. 1, 2008.
- [73] E. Deelman, and B.K. Szymanski, Continuously monitored global virtual Time, in *International Conference Parallel and Distributed Processing Techniques and Application*, pp. 1-10, 1997.
- [74] mvapich2/gnu, <http://mvapich.cse.ohio-state.edu/>, Last Accessed on February 23, 2012.
- [75] adevs, <http://www.ornl.gov/~1qn/adevs/index.html>, Last Accessed on February 23, 2012.
- [76] S. Guo, and X. Hu, Profile-based partition for parallel simulation of DEVS-FIRE, in *Proceedings of 43rd Annual Simulation Symposium (ANSS)*, pp. 155-155, 2010.
- [77] T. Kwok and A. Mohindra, Resource Calculations with Constraints and Placement of Tenants and Instances for Multi-Tenant SaaS Applications, in *International Conference on Service Oriented Computing (ICSOC)*, 2008.

- [78] H.Yaish, M.Goyal, and G.Feuerlicht, An Elastic Multi-tenant Database Schema for Software as a Service, Dependable, Autonomic and secure Computing (DASC), *IEEE Ninth International Conference*, 2001.
- [79] O. Schiller, B. Schiller, A. Brodt, et al., Native support of multi-tenancy in RDBMS for software as a service, in *Proceedings of the 14th International Conference on Extending Database Technology*, NY, 2011.
- [80] Nitu, Configurability in SaaS (software as a service) applications, in *Proceedings of the 2nd India software engineering conference*, 2009.
- [81] X. Li, T. Liu, Y. Li, et al., SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment, in *Proceedings of ICSOC*, pp. 649-663, 2008.
- [82] P. Mell, and T. Grance, Draft NIST Working Definition of Cloud Computing, 2009.
- [83] L. Group, Simulation as a service to business process management (BPM), http://www.lanner.com/comms/090924/LSIM_September.pdf, Last accessed on June 12, 2012.
- [84] J. F. Thomas Paviot, Implementation of a SaaS Based Simulation Platform Using Open Standards and Open Source Software, presented at the 12th NASA-ESA Workshop on Product Data Exchange (PDE2010), 2010.
- [85] W. Tsai, W. Li, H. Sarjoughian, et al, SimSaaS: simulation software-as-a-service, in *Proceedings of the 44th Annual Simulation Symposium*, pp. 77-86, 2011.
- [86] J. A. Miller, Y. Ge, and J. Tao, Component-based simulation environments: JSIM as a case study using Java Beans, *Simulation Conference Proceedings*, vol. 1, pp. 373-381, 1998.

- [87] A. Buss, Component based simulation modeling with simkit, Simulation Conference Proceedings, vol. 1, pp. 243-249, 2002.
- [88] R. A. Kilgore, Silk, Java and object-oriented simulation, Simulation Conference Proceedings, vol. 1, pp. 246-252, 2000.
- [89] O. Balci, A. I. Bertelrud, C. M. Esterbrook, et al., Visual simulation environment, Simulation Conference Proceedings, vol. 1, pp. 279-287, 1998.
- [90] Y. J. Son, A. T. Jones, and R. A. Wysk, Component based simulation modeling from neutral component libraries, Computers and Industrial Engineering, vol. 45, issue 1, pp. 141-165, 2003.
- [91] J. Sommer, and W. Franz, A Component-based Simulation Model and its Implementation of a Switched Ethernet Network, http://www.jacobsschool.ucsd.edu/GordonCenter/g_leadership/l_summer/docs/saase/papers/SommerFranz.pdf, Last accessed on July 5, 2012.
- [92] P. H. Cheung, K. Hao, and F. Xie, Component-Based Hardware/Software Co-Simulation, *Digital System Design Architectures, Methods and Tools*, pp. 265-270, 2007.
- [93] TinyOS, www.tinyos.net, Last accessed on June 12, 2012.
- [94] Workflow, <http://en.wikipedia.org/wiki/Workflow>, Last accessed on February 23, 2012.
- [95] Scientific workflow system, http://en.wikipedia.org/wiki/Scientific_workflow_system, Last accessed on February 23, 2012.
- [96] OASIS, <https://www.oasis-open.org/>, Last accessed on February 23, 2012.
- [97] Apache ODE, <http://ode.apache.org/>, Last accessed on February 23, 2012.

[98] jBPM, <http://www.jboss.org/jbpm/>, Last accessed on February 23, 2012.

[99] Oracle BPEL Process Manager,
<http://www.oracle.com/technetwork/middleware/bpel/overview/index.html>, Last accessed
on February 23, 2012.