MAMSaaS: Mashup Architecture with Modeling and Simulation as a Service

by

Sixuan Wang, M.Sc., M.Eng.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

© 2016, Sixuan Wang
Acknowledgments

This work is dedicated to my parents and my wife for their endless support and care. I also would like to thank my supervisor Dr. Gabriel Wainer for his inspiration, support, encouragement, and for always being there both on the academic and personal levels. In addition, I would like to thank Profs. Emil Petriu, Michael Weiss, and Babak Esfandiari for their scientific insights and valuable comments on my thesis proposal. Likewise, I would like to thank Profs. Gregory Zacharewicz, Emil Petriu, Michael Weiss, and Gregory Franks for their willingness to be on my defence committee.
ABSTRACT

In recent years, developing Modeling and Simulation (M&S) applications has become more and more complex. New technologies, like Web Services (WS) and Cloud computing, have been recently used in Modeling and Simulation (M&S). However, developing M&S applications using these technologies is still a complicated process. The reasons for this include: 1) it is hard to develop web services for varied M&S resources; 2) it is complicated to deploy M&S resources in the Cloud; 3) it is hard to integrate with varied M&S services; 4) it is complex to identify and select resources and services based on their meaning. In this research, we aim to simplify the development and integration of M&S applications using web technologies by solving the issues mentioned above. To do so, we propose the Mashup Architecture with Modeling and Simulation as a Service (termed MAMSaaS). MAMSaaS is a layered and lightweight M&S application development approach. It has five layers, which are Cloud, Box, Wiring, Mashup, and Tag Ontology Layers. It has a simplified life cycle to develop, deploy, identify, select, integrate and execute varied M&S resources as services in the Cloud. In the Cloud Layer, we developed CloudRISE middleware to expose RESTful Modeling and Simulation as a Service (MSaaS) for varied M&S resources; in addition, we propose new methods using Cloud computing and Experimental Framework concept to simplify the deployment of experiment environment. In the Box, Wiring and Mashup Layers, we present a new method based on mashup technologies to simplify the integration, execution and visualization of M&S applications. In the Tag Ontology Layer, we propose a new semantic selection approach using tag-mining and ontology-learning technologies, to identify and select M&S resources based on their meanings.
# Table of Contents

ABSTRACT ........................................................................................................ iii

List of Tables ...................................................................................................... vi

List of Figures .................................................................................................... vi

List of Acronyms and Abbreviations ............................................................... ix

Chapter 1: Introduction ...................................................................................... 1
  1.1 Research Goals .......................................................................................... 5
  1.2 Contributions ........................................................................................... 7
  1.3 Thesis Organization .................................................................................. 11

Chapter 2: Background ...................................................................................... 13
  2.1 M&S Resources ....................................................................................... 13
  2.2 M&S Application Development ............................................................... 15
  2.3 Web-based Simulation ............................................................................. 17
  2.4 Cloud-based Simulation ......................................................................... 21
  2.5 Mashups .................................................................................................. 24
  2.6 Semantic Mashup and Ontology ............................................................... 27
    2.6.1 Semantic Mashup .............................................................................. 27
    2.6.2 Ontology in M&S ............................................................................. 28
    2.6.3 Ontology Learning .......................................................................... 30
    2.6.4 Tagging System .............................................................................. 30
  2.7 DEVS and Related Resources ................................................................. 32
  2.8 Summary ................................................................................................. 34

Chapter 3: The MAMSaaS Architecture ......................................................... 35

Chapter 4: Cloud Middleware and MSaaS .................................................... 40
  4.1 Cloud Layer ............................................................................................ 40
  4.2 Implementing an MSaaS: the CloudRISE Middleware ................................ 47
  4.3 Case Study: Crowd M&S using CloudRISE in the Cloud ....................... 55
  4.4 Summary ................................................................................................. 63

Chapter 5: M&S Mashup .................................................................................. 64
  5.1 M&S Mashup Methodology ...................................................................... 65
    5.1.1 Box – developing Mashup components ........................................... 67
    5.1.2 Box Wiring – linking boxes ......................................................... 70
5.1.3 M&S Mashup – building mashup applications ........................................ 72
5.2 M&S Mashup Implementation .................................................................. 73
  5.2.1 Box Development Tool ...................................................................... 73
  5.2.2 M&S Mashup Platform ...................................................................... 76
5.3 Simple Case Study: Fire Spreading M&S Mashup.................................. 81
5.4 Summary ............................................................................................... 87

Chapter 6: Semantic Selection ....................................................................... 88
  6.1 Semantic Selection Methodology ............................................................. 88
    6.1.1 Identifying Resources – Signature .................................................. 90
    6.1.2 Getting Meanings - Tag Mining ...................................................... 92
    6.1.3 Learning Ontology – Tag Tree ....................................................... 94
    6.1.4 Recommending resources – Semantic Selection ......................... 101
  6.2 Implementation ...................................................................................... 104
  6.3 Case Study: Semantic Model Selection ................................................ 107
    6.3.1 Semantic Model Selection Process .............................................. 108
    6.3.2 Success Rate Testing ..................................................................... 109
    6.3.3 Fast Model Development Example .............................................. 111
  6.4 Summary ............................................................................................... 113

Chapter 7: Case Studies: M&S Mashup Applications .................................... 114
  7.1 General Mashup for Cell-DEVS Models .............................................. 114
  7.2 A Mashup for Crowd Modeling and Building Information Management ...... 122
  7.3 A Land Use with GIS M&S Application .............................................. 128
  7.4 General Mashup for DEVS Models ..................................................... 133
  7.5 Summary ............................................................................................... 136

Chapter 8: Conclusion ................................................................................... 138
  8.1 Thesis Summary ................................................................................... 138
  8.2 Future Work .......................................................................................... 142

Appendix A: Alternative Integration Approach using Workflow ...................... 145

References ................................................................................................. 153
List of Tables

Table 1. IaaS Compute APIs: AWSEC2Instances.java for Amazon AMI and EC2 ........ 45
Table 2. IaaS Storage APIs: AWSS3Storage.java for AmazonS3........................................... 45
Table 3. DEVS VM kernel vs. MS-Image .................................................................................. 46
Table 4. Comparison among DEVS/SOA, RISE and CloudRISE ........................................ 55
Table 5. Mapping WADL and Resource signature ................................................................... 92
Table 6. Cell states for Crowd model ...................................................................................... 124

List of Figures

Figure 1. Categories of M&S resources: basic M&S entities and supported resources ... 13
Figure 2. Modeling and Simulation Framework, adapted from [Tra06] ......................... 15
Figure 3. Web APIs protocol usage (from ProgrammableWeb.com)............................... 24
Figure 4. Life-cycle models of (a) current web applications and (b) mashups, adapted from [Dan11] ................................................................................................................................. 26
Figure 5. Relationship of MSaaS and Cloud services ............................................................. 35
Figure 6. MAMSaaS (Mashup Architecture with Modeling & Simulation as a Service) 36
Figure 7. Users in the integrated layered architecture ............................................................ 38
Figure 8. Basic Structure of the Cloud Layer ......................................................................... 40
Figure 9. MS-Image .................................................................................................................... 41
Figure 10. Amazon EC2 instances for MAMSaaS ................................................................. 43
Figure 11. Amazon S3 resource folders for MAMSaaS ......................................................... 44
Figure 12. The CloudRISE URL hierarchy ........................................................................... 48
Figure 13. Structure of the Model Description file ................................................................. 49
Figure 14. An example of Instance Description file ............................................................... 50
Figure 15. URL hierarchy of the EF Template for Simulation ............................................. 50
Figure 16. Structure of the EF Description for Simulation .................................................... 52
Figure 17. Structure of the Experiment Description for Simulation ...................................... 53
Figure 18. Lifecycle of the experiment in CloudRISE .......................................................... 53
Figure 19. The resources involved in the Crowd M&S ......................................................... 56
Figure 20. Model Description file for the crowd model ....................................................... 58
Figure 21. IFC loaded in Autodesk Revit for crowd behavior study .................................. 58
Figure 22. EF Description for the Crowd Simulations ........................................... 59
Figure 23. Experiment Description for the Crowd Simulations ................................ 60
Figure 24. Performance test of reproducing simulation experiments using CloudRISE .. 61
Figure 25. 3D visualization of parsed results of crowd simulation ............................ 63
Figure 26. M&S Mashup Methodology .................................................................. 65
Figure 27. Mappings of M&S mashup process and the standard mashup life cycle ...... 66
Figure 28. Event-driven pattern among boxes ...................................................... 71
Figure 29. Box/Wiring/Mashup example ............................................................... 72
Figure 30. Class Diagram of Box Development Tool ............................................. 73
Figure 31. Box Development Tool menus ............................................................. 75
Figure 32. Extraction Box Signature from Facebook REST WebAPIs ....................... 76
Figure 33. Generating Box configuration and suggesting similar Box Packages ......... 76
Figure 34. Class diagram of M&S Mashup Platform (powered by WireCloud) ....... 77
Figure 35. Box uploading .................................................................................... 79
Figure 36. Box selection ...................................................................................... 79
Figure 37. Adding box resources into user’s workspace ......................................... 79
Figure 38. Box wiring .......................................................................................... 80
Figure 39. Mashup execution .............................................................................. 80
Figure 40. Web Search Box configuration example ................................................. 83
Figure 41. Web Search Box HTML view example ................................................... 84
Figure 42. Web Search Box JS function example ................................................... 85
Figure 43. Uploaded boxes for Fire Spreading M&S mashup ................................ 85
Figure 44. Boxes wiring for Fire Spreading M&S mashup ...................................... 86
Figure 45. Executing the Fire Spreading M&S mashup application ....................... 86
Figure 46. Procedure of the Tag-tree based semantic selection method ................. 89
Figure 47. WADL Example from RISE .............................................................. 91
Figure 48. Co-occurrence graph (a) & Tag tree ontology (b) .................................. 94
Figure 49. Modified closeness centrality ............................................................ 99
Figure 50. An example of resource composition based on a tag-tree ontology .......... 102
Figure 51. PyCom Class Diagram for semantic selection ...................................... 104
Figure 52. Part of learnt tag-tree ontology from simulation services ....................... 106
Figure 53. Tag-tree learning algorithm vs. Heymann’s algorithm ......................... 106
Figure 54. Model composition, repository and simulation .................................... 108
Figure 55. Model composition success rates ....................................................... 111
Figure 56. A part of the learnt tag-tree ontology .................................................. 112
Figure 57. Semantic model composition for an occupancy case ............................. 112
Figure 58. Experiment Description for fire spreading model ............................... 116
Figure 59. Uploaded boxes for general Cell-DEVS mashup ......................................................... 117
Figure 60. Boxes wiring for general Cell-DEVS mashup ............................................................... 118
Figure 61. Part of Tag-tree used in Cell-DEVS mashup ................................................................. 119
Figure 62. Fire Spreading mashup using the general Cell-DEVS mashup ................................. 120
Figure 63. Snow growing mashup using the general Cell-DEVS mashup ................................. 121
Figure 64. Cancer propagation mashup using the general Cell-DEVS mashup ....................... 121
Figure 65. Wiring boxes of Crowd M&S mashup .................................................................... 125
Figure 66. Part of Tag-tree used in Crowd M&S mashup ...................................................... 126
Figure 67. Crowd M&S mashup ............................................................................................... 127
Figure 68. Building Occupancy vs. Evacuation Time, adapted from [Wan12a] ................. 128
Figure 69. Boxes developed for GIS M&S mashup ................................................................. 130
Figure 70. Boxes wiring for GIS M&S mashup ...................................................................... 131
Figure 71. Part of Tag-tree used in GIS M&S mashup ............................................................. 132
Figure 72. Executing the GIS M&S mashup application ......................................................... 133
Figure 73. Boxes wiring for Barber DEVS M&S mashup ........................................................... 135
Figure 74. Executing the Barber M&S mashup application ....................................................... 136
Figure 75. Three-layer Workflow Architecture for Crowd M&S ........................................... 146
Figure 76. The Composition Layer in the Cloud ..................................................................... 148
Figure 77. Overall workflow in Taverna .................................................................................. 149
Figure 78. Sub-task workflow for Cloud-based Simulation using SimaaS ....................... 150
Figure 79. 3D visualization results in different perspectives .............................................. 151
### List of Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>3-Dimensional</td>
</tr>
<tr>
<td>AMI</td>
<td>Amazon Machine Images</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARSlab</td>
<td>Advanced Real-time Simulation Laboratory</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>BIM</td>
<td>Building Information Modeling</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer-aided Design</td>
</tr>
<tr>
<td>CBS</td>
<td>Cloud-based Simulation</td>
</tr>
<tr>
<td>CBSE</td>
<td>Component-based Software Engineering</td>
</tr>
<tr>
<td>Cell-DEVS</td>
<td>Cellular Discrete Event Simulation</td>
</tr>
<tr>
<td>CFE</td>
<td>Context-Frame-Experimenter</td>
</tr>
<tr>
<td>CloudRISE</td>
<td>Cloud RESTful Interoperability Simulation Environment</td>
</tr>
<tr>
<td>COBS</td>
<td>Component-based simulation</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DEVS</td>
<td>Discrete Event Simulation</td>
</tr>
<tr>
<td>DSM</td>
<td>Domain Specific Modeling</td>
</tr>
<tr>
<td>EC2</td>
<td>Amazon Elastic Compute Cloud</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographical Information System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>LCIM</td>
<td>Level of Conceptual Interoperability Model</td>
</tr>
<tr>
<td>IFC</td>
<td>Industry Foundation Class</td>
</tr>
<tr>
<td>MSaaS</td>
<td>Modeling and Simulation as a Service</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modeling and Simulation</td>
</tr>
<tr>
<td>EF</td>
<td>Experimental Framework</td>
</tr>
<tr>
<td>MAMSaaS</td>
<td>Mashup Architecture with Modeling and Simulation as a Service</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>POS</td>
<td>Part of Speech</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer WS</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>RISE</td>
<td>RESTful Interoperability Simulation Environment</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SaaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SDLC</td>
<td>System Development Life Cycle</td>
</tr>
<tr>
<td>SimaaS</td>
<td>Simulation as a Service</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Reference Locator</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WADL</td>
<td>Web Application Description Language</td>
</tr>
<tr>
<td>WBS</td>
<td>Web-based Simulation</td>
</tr>
<tr>
<td>WebAPI</td>
<td>Web Application Programming Interface</td>
</tr>
<tr>
<td>WS</td>
<td>Web Services</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

Nowadays, Modeling and Simulation (M&S) has become a discipline that has its own knowledge, formalisms and methodologies [Ban10]. The M&S related technologies have been widely applied in almost every aspect of life [Pap10].

A key concept in M&S is the model, which is a representation to understand better a given system. We abstract what we learned about the system into a model that represents the system of interest. Although abstraction implies losing information, it allows us to describe the behaviors of the system, analyze it, and it makes easier to prove properties of the model [Wai09]. A second concept to consider is simulation, which is the execution of models with particular sets of data using some kind of (in general computing) device [Wai09].

M&S allows solving complex problems that traditional methods could not answer, and M&S applications are becoming more and more popular. Several development processes have been proposed [Sal09] [Zei15] to support the development of such M&S applications, (e.g. System Development Life Cycle [Sal09], M&S Life Cycle [Bal12] and Component-based Simulation Process [Dal07].

As it will be discussed in Section 2.2, a common issue of the above processes is that they make developing applications more and more complex [Zha14]. Each process involves different resources and activities from different people, which usually is costly and needs years of development and maintenance. In particular, as the fast development of technologies, current M&S applications need support that most of development processes cannot address:

1) **Varied M&S resources**: The quantity and variety of the simulation resources are increasing, such as source systems, models, simulators, experimental frameworks and experiments [Zei00] [Che13]. In addition, more and more data (e.g. text, database) and functions resources related to simulation (e.g. data collection, result analysis, visualization) are produced and available [Loz07].

2) **Environment configuration**: it is complex to configure and maintain simulation environments in order to run varied model experiments under different scenarios.
Likewise, the CPU and memory resources of a single computer can be insufficient to execute complex models for advanced applications [Dan11].

3) **Increasing number of users**: In addition, more and more people who own these varied resources are involved in M&S activities [Bal12]. They can be specialists or casual users who want to collaborate across organizational boundaries.

In order to address these issues, in recent years, people started using Web services (WS) and Cloud computing for M&S. New approaches were proposed, including Web-based Simulation (WBS, which exposes M&S functions as web services) and Cloud-based Simulation (CBS, which integrates WBS and Cloud Computing).

Web services with varied technologies (e.g. SOAP Web services, REST Web services) have been used in M&S for around 20 years [Jun13]. This method, usually called *Web-based Simulation* (WBS), uses existing simulation functions and it exposes them as Web services [Byr10]. WBS shares the simulators and their environments (which are originally accessible on a single computer) through the Internet. This method has been successful, and a large number of M&S Web services exist [Jun13].

At the same time, the emergence of Cloud Computing made Cloud-based Simulation (CBS) a feasible and attractive alternative to WBS [Ong14]. CBS goes beyond WBS, and it uses Cloud Computing technologies to reduce costs and to make it easier to develop M&S systems by taking advantages of the Cloud Computing services, e.g. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). CBS uses Cloud Computing to manage varied M&S resources and build different simulation environments [Cay13b]. Cloud Computing provides various IT resources to users in an easy-to-use, on-demand and pay-as-you-go manner. These advantages make simulation resources more accessible to users. In CBS, both web services and Cloud computing are used in M&S. The use of web services in CBS has received the name of Modeling and Simulation as a Service (MSaaS). MSaaS is a special form of Software-as-a-Service (SaaS), as it hides the underlying infrastructure, platform and software details from the users.

In fact, various experts in the area [Tay12] [Tay13] have mentioned CBS as one of the Grand Challenges in Modeling and Simulation. The use of CBS and MSaaS is still in a preliminary stage [Gar11].
In addition, there are many open Web APIs that could be useful for M&S applications (e.g. weather forecast, GIS information, and big data for simulation inputs), but little effort has been done to integrate simulation services with other services [Tay13]. They could improve user experience and make richer applications [Jun13].

In summary, the motivations for this work can be summarized as follows:

1) It is hard to develop web services for varied M&S resources, due to the Web framework constrains and the variety of M&S resources.

Web services have been used in M&S for many years; these efforts are mainly in WBS. After a peak period between 2000 and 2002, the research and publications about WBS dropped quickly [Tol14], because there was a mismatch between the WS Framework used by WBS and the main characteristics of the Web. Most WBS frameworks were based on SOAP, which has issues of structural constraints (e.g. mixed design and implementation, difficulties in development, exposing internal implementation details, etc.) [Wag12]. Any SOAP-based WS project takes at least 2 years in defining the SOAP WS layer, deploying those services, and standardizing the interfaces. Therefore, these SOAP-based WS failed to take full advantage of features of the Web (e.g. its universal interface, interoperability, ease of navigation and use, etc.). Many WBS efforts have merely focused on re-implementing existing standalone M&S systems by using SOAP-based WS [Tol14].

Current WBS only focuses on providing web services for simulation engines. However, as the M&S systems become more complex, the number of related resources increases very fast [Che13] [Sko12]. These services are either web-based M&S services (that expose models or simulations as services) or Web APIs (that are useful for M&S). These resources, which are varied in type and content, are complex to be developed as web services. This implies that all kinds of M&S resources can be used, reproduced or managed in a given environment [Rob04].

2) It is hard to deploy M&S resources in the Cloud, since the current technical solutions are not practical to reproduce and configure M&S experiments.
As discussed above, research in CBS is still in a preliminary stage. Johnson and Tolk [Joh13] have identified five challenges perspectives that have to be addressed in CBS: technical, governance, security, business and conceptual perspectives. Among them, we are interested in focusing on the technical perspective.

A related issue is reproducibility, i.e., the ability to repeat an entire simulation experiment or related functions. However, recently years, many studies questioned the reproducibility in M&S. In [Kur06], the author analyzed several papers from the ACM MobiHoc symposium between 2000 and 2005, and they have found that less than 15% of the simulation results were reproducible. To run a simulation model from others, a user needs to find right resources and web services to install a simulation environment, which can be complex (it needs to deal with underlying computing instance, operating system, and dependencies). Then we need the same model files, inputs, and experiments. Current WBS and CBS are still unable to simplify these processes [Byr10] [Guo11]. To the time of writing this thesis, there is no real implementation about how to deploy varied M&S resources on the Cloud in terms of reproducing simulation experiments.

In addition, it is hard to configure simulation environments in the Cloud. The life cycle of experiments should be easily manageable [Ove02]. However, to configure and manage the simulation resources, users need to pay for expensive supercomputers or additional storage hardware, as well as manually performing the migration and backup efforts.

3) It is hard to integrate with varied M&S services, and existing WebAPIs.

Web services have been used in M&S for many years for WBS and CBS. With the development of Web 2.0, there are numerous Web APIs available. For example, according to ProgrammableWeb, there are more than 11,000 APIs registered (REST WSs take 73% while SOAP-based APIs take 27% [Sir14]. Nevertheless, it is difficult to integrate them for complex M&S applications. In particular, experts from different domains cannot easily use existing M&S related resources for fast application development. There has been no research showing how to integrate these available M&S services and useful Web APIs in a simplified way for developing M&S applications.
4) **It is hard to identify and select resources and services based on their meaning.**

Another issue to be simplified during the integration phase of M&S applications is to select right resources. There could be hundreds or thousands of resources available. It is hard to understand what they are and how they are related to each other. When integrating these resources, those with similar meanings should be grouped together. However, it is hard to get the meaning and relationships among these resources. In [Men12], the author stresses that one of the biggest issues to compose a model from existing models is the semantic representation of behavior, labels and terminology. Similarly, for integrating services, it is also hard to get their semantic information.

In M&S, a semantic layer could be added to the M&S resources and annotate them using an external ontology for knowing their “meaning”. Ontologies represent knowledge as a set of concepts within a domain, with shared vocabulary agreed by specialists. The consideration of ontologies in M&S has been widely used [To10a] [Lac06] [Ze10]. However, the major problem of them is that these ontologies are predefined. In addition, their composition of simulation resources is limited to specific cases and rules.

### 1.1 Research Goals

The primary goal of this research is to address the issues above, in order to simplify the development and integration of M&S applications using web technologies. This thesis focuses on the issues mentioned above and introduces new methods to solve these issues. To achieve the goal, new concepts and techniques are needed:

1) In order to develop varied M&S resources as service by taking full advantages of the Web, we propose using RESTful WS [Fie00] as the *WS framework*. REST is simpler than SOAP; it is directly built upon the Web and it has a more concise style in terms of implementation and operation [Son10]. We use RESTful WS to expose every M&S resource as a unique URL that can be operated by the uniform HTTP methods (GET, POST, PUT and DELETE). In order to handle the *variety* of M&S resources, we use the concept of Modeling and Simulation as a Service (MSaaS) in the Cloud. MSaaS builds on Cloud services and it delivers web services related to M&S. We implemented an
MSaaS middleware, named CloudRISE, which allows users to expose and manage varied M&S resources as services in the Cloud.

2) In order to deploy M&S services on the Cloud, we propose a new method using Cloud computing in order to simplify the deployment of simulation environment and we use the concept of Experimental Frameworks concept defined in [Zei00] to configure and reproduce experiments. We use Cloud computing technology to store all M&S resources and compute experiments of simulation and functions on the Cloud. We developed a lightweight Experimental Framework Template using XML in the MSaaS middleware to configure and control the life cycle of experiments. Likewise, we used the Cloud as the underlying infrastructure to deploy the MSaaS middleware. In addition, we proposed an MS-Image concept, which is a VM-like approach that includes cloud middleware, simulation and function environment. We can easily launch and configure different computing instances from MS-Images.

3) In order to integrate available M&S services and existing WebAPIs, we present a new method based on mashup technologies in the Web 2.0 [Bal08]. Mashups have been widely used in different domains; however, these technologies have never been used to simplify the process of building M&S applications. We propose an M&S Mashup method to develop M&S mashup applications for integrating different M&S related WSs and resources. This novel method is a lightweight M&S application development. We implemented a Mashup platform to develop and identify M&S mashup components as well as link and execute mashups for quick M&S application development. This mashup platform supports universal identification and development for mashup component (named Boxes). Boxes consist of varied M&S resources (MSaaSs, WebAPIs, widgets and operators). In order to link and execute these components, this mashup platform supports component linking, development and search boxes; it can also link boxes by wiring their input/output ports, and then execute and visualize M&S at run-time.

4) In order to simplify the section process of M&S resources, we propose a new semantic selection approach with data mining and machine learning algorithms. We design an automatic identification way to add semantic to each resource using tagging mining techniques. Furthermore, we designed a new algorithm to learn tag-tree ontology from mined tags. The general idea is to automatically mine tag signatures from the user-
interested M&S-related resources, to learn and build the tag-tree ontology from the mined tag signatures, then to select right resources based on their tags and tag-tree ontology, avoiding the predefined ontology effort and increasing the automation of M&S application development.

We present an architecture to include all of the above efforts, named Mashup Architecture with Modeling and Simulation as a Service (MAMSaaS). MAMSaaS is a layered architecture with five layers: Cloud, Box, Wiring, Mashup, and Tag Ontology. MAMSaaS is a lightweight development approach of M&S application. It has a simplified life cycle to develop, deploy, identify, select, integrate and execute varied M&S resources as services in the Cloud.

In order to show how the proposed architecture can simplify the development of M&S mashup applications, we present different case studies. The first case study is for evacuation of a crowd in a shopping mall. We will discuss the M&S resources needed (e.g. evacuation model, simulator, data collection, and 3D visualization). We will show how to deploy them as MSaaS in the Cloud using CloudRISE, and then we will show how to use these generated MSaaS and the mashup platform to develop mashup application fast, by integrating M&S services like building information collection, simulation, visualization, as well as Web APIs like Web search, Flickr, and YouTube. Furthermore, we will also present another M&S Mashup for GIS landuse study, by integrating M&S services like GIS data collection, Cloud-based Simulation and visualization in Google Map, as well as Web APIs like weather forecast and location lookup. Moreover, we will discuss some general M&S Mashups for DEVS/Cell-DEVS models. In addition, we will show how to use the proposed semantic selection approach for identifying “meanings” of the M&S resources and selecting related “mashable” resources in these examples.

1.2 Contributions

We have published this research in different venues.

The general MAMSaaS architecture and its semantic approach has been first published the *Symposium on Theory of Modeling and Simulation (TMS), 2014*, which
focuses on the overall architecture and semantic selection part of varied M&S services [Wan14b]. Later, in the *Symposium on Theory of Modeling and Simulation (TMS) 2015*, we presented results on applying this semantic selection approach in model composition for identifying and selecting models [Wan15d]. We have shown the success test rate of using our approach to simplify the selection process for DEVS models.

The initial MSaaS idea was published in *Simulation: Transactions of the Society for Modeling and Simulation, 2015*, focusing on the design and implementation of MSaaS middleware, and its integration with other functions (building data collection, simulation result parsing and 3D visualization) in the crowd M&S [Wan15a]. Later, we submitted our latest research on MSaaS middleware and CloudRISE [Wan16] to the same Journal (accepted). CloudRISE has already been used by other students in ARSlab successfully. For example, CloudRISE was used as backend service in a mobile APP for running and visualizing simulation in Smartphones, which won a special mention for the Mobile App Competition during the *Spring Simulation Multi-Conference, 2015* [Etm15].

We also presented the whole mashup architecture and its mashup platform details in *Web Information System Engineering 2015*, as well as a prototype of GIS M&S Mashup [Wan15c]. Recently, we extended and submitted this work to the *Simulation: Transactions of the Society for Modeling and Simulation, 2015*. This work has also won the Silver Medal Award during the *Poster Competition on Data Day, 2015* [Wan15e].

The work has been successfully used with different M&S-related systems, allowing us to develop and integrate M&S resources for fast M&S application development in different areas, including Computer-aided Design (CAD), Building Information Modeling (BIM), Geographical Information System (GIS), 3Ds Max and Google Earth [Wan15a] [Wan16] [Wan14a] [Wan13a] [Wan13b] [Wan12a]. The application domains cover crowd M&S, emergency evacuation, occupancy analysis, and landuse analysis.

The scientific contributions of this research are as follows:

1) A novel MAMSaaS architecture and a lightweight development process of M&S application to simplify the whole process to develop, deploy, identify, selection, integrate and visualize varied M&S resources as services.
2) A new MSaaS middleware (implemented in CloudRISE) to develop and deploy varied M&S resources as RESTful services on the Cloud. It can generate MSaaS services dynamically and automatically in the cloud based on their EF specification. MSaaS middleware uses the experimental framework template to reproduce experiments quickly. MSaaS middleware uses MS-Image for fast cloud deployment for simulation and function environment.

3) An original mashup method to integrate M&S services and existing WebAPIs, with a process of building mashup components as boxes, wiring boxes as mashups, and running and visualizing mashups during the run-time. We implemented a Mashup platform to develop and identify M&S mashup components as well as link and execute mashups for quick M&S application development.

4) A new semantic selection method for M&S resource recommendation. This is used to identify varied M&S resources and services, get their meanings by proposed tag-mining approach, learn tag-tree ontology based on their meanings, and recommend potential links based on tags and tag-tree ontology, which facilitates the mashup development, resource discovery and composition.

5) Design and implementation tools, including the CloudRISE middleware on AWS, the box development tool, the M&S mashup platform, the resource recommendation tool based on the proposed semantic selection algorithms.

6) Mashups Applications that adopt MAMSaaS and developed tools, including the Crowd M&S mashup, GIS M&S Mashup, and general DEVs/Cell-DEVs M&S mashup.

Here is the list of publications related to this research:

In Journals


**Book Chapter**


**Refereed Conference Papers**


**Conference Posters**


### 1.3 Thesis Organization

The rest of this proposal is organized as follows. In Section 2, we will review the literature of the techniques and approaches of trying to solve the complexity issues when developing and integrating M&S applications. In Section 3, we will propose the overall MAMSaaS architecture. In Section 4, we will discuss the design details of MSaaS middleware and its implementation CloudRISE. In Section 5, we will discuss the design details of the M&S Mashup approach and the implemented mashup platform. In Section
6, we will discuss the details of design and implementation of semantic selection method and its algorithms. In Section 7, we will show different applications using MAMSaaS for rapid M&S mashup development. In the Section 8, we will conclude this research.
Chapter 2: Background

In this section, we will review the literature of the domains related to the techniques and approaches regarding the development and integration of M&S applications.

2.1 M&S Resources

As discussed before, one trend of M&S applications is that they involve varied M&S resources. We have classified and organized the variety of M&S resources in order to make them more manageable. Figure 1 shows how we can group the varied resources available.

![Diagram of M&S resources: basic M&S entities and supported resources]

**Figure 1. Categories of M&S resources: basic M&S entities and supported resources**

As we can see, an M&S resource can be considered as an *entity* (i.e., a basic resource that is directly related to M&S) or a *supported resource* (i.e., a related resource that is helpful for M&S).

The theory of modeling and simulation presented in [Zei00] provides a conceptual framework for M&S *entities*. M&S entities include the *source system, models, simulators, experimental frameworks* and *experiments* (seen in the left part of Figure 1). The source system is a real or virtual environment that we are interested to model. A model is a representation of a source system built to understand that system under a given experimental framework. A simulator is a device that executes a model in order to study
its behavior over time. The simulation of a model involves different simulation experimental frameworks and associated experiments. The experimental framework (EF) represents the context under which a system or a model is observed or experimented with. This kind of context is essential information to configure and reproduce the simulation experiments.

The **supported resources** can be grouped into two categories: supported *data* and supported *functions*, as seen on the right part of Figure 1. A supported data can be any resource that is not executable, like a system behavior database, scenario data and documentation. Supported data can be in a wide range of forms: text, file, picture and video [Sko12]. Supported functions are any resources executable, from small portions of code, through larger components, to complete executable programs [Rob04]. Examples of the supported functions are data collection tools, results analysis program, animation and visualization tools. A function is in general associated with an execution environment and provides interfaces for others to access.

As discussed before, one important issue to develop M&S application is to configure and reproduce a simulation environment. To do so, it is important to capture the specifications of the context in which a system is studied; also, it is generally agreed that this context must explicitly state the underlying objectives, assumptions, and constraints of the study [Tra06]. In addition, the context also should state the information of the experiments associated to that model [Chr09]. Without this kind of context, the model is not valid; thus, it becomes meaningless and impossible to reproduce the experiments associated to that model. As mentioned above, the EF concept has been introduced to capture this kind of context. There is also some research of tying to formalize EF specification. In [Bar04], the idea of a framework that takes into account different meanings of an EF was introduced. In [Guo11], a similar specification of experiment with a meta-model of a simulation is proposed. The Context-Frame-Experimenter (CFE) framework by [Tra06] further formalized these ideas, which is shown in Figure 2. CFE presents a clear distinction between the three abstraction levels: the context through which a real system is studied, the frame specification of this context as an EF, and the experimenter to implement this EF for executing a simulator. They
view EF as a circuit board with input and output ports, into which a source system or a model can be “plugged-in”.

![Diagram](image)

**Figure 2. Modeling and Simulation Framework, adapted from [Tra06]**

An EF can help to configure and reproduce simulation environment. However, to our best knowledge, the current EF research in M&S is very limited. There is little research on the M&S about trying to provide both clear context and configurable experiments related to the simulation models. The CFE framework [Tra06] presents a clear distinction and specification of the context, frame and experiment, but it is neither widely used nor easy to adopt; besides, there is no real implementation and applications using the CFE framework to configure and reproduce experiments. Furthermore, the CFE specification is complex that requires more efforts of a modeller to maintain the ports between the EF and model. In addition, the EF does not take advantages of web technology. Many existing M&S web services are very beneficial for solving the issues of environment configuration and reproducibility.

### 2.2 M&S Application Development

We are interested to simplify developing and integrating M&S applications. Here we discuss the traditional development processes of M&S applications.

Initially, in the early days of system development, like other type of applications, a M&S application is developed using the System Development Life Cycle (SDLC)
approach. SDLC provides a set of models or methodologies such as waterfall, v-shaped, incremental, and spiral [Sal09]. The waterfall model defines a sequence of phases; the output of each phase becomes the input for the next phase. The v-shaped model is similar to the waterfall model with the difference that testing phases are developed before implementation. The incremental model consists of dividing the waterfall model into smaller iterations. The spiral model is widely used in the development of complex systems. This model is similar to the incremental model, with more efforts on risk analysis.

In recent years, the M&S community has become conscious of the foundational theory on the M&S application development. As a result, research on the M&S life cycle management is gradually attracting attention from academic circles. Radeski and Parr [Rad02] proposed the modeling simulation life cycle framework, which defined an organization structure of the modeling simulation process, work products, quality assurance, and project management. This framework also described the features and requirements of the life cycle phases such as development, use, maintenance, and reuse of the modeling simulation system. In [Abd10], the authors achieved some valuable results in the model life cycle management and developed a prototype for model management system. In [Zha11], the author proposed that a model life cycle includes model analysis, model design, model construction, model Verification and Validation (V&V), model application, and model maintenance. In [Zei15], the authors proposed the concept of Model Engineering, which is defined as a general engineering methodology for theories, methods, technologies, standards, and tools that guarantees the credibility of the full life cycle of a model.

Another development process is the M&S life cycle, proposed by [Bal12]. It represents framework for organization of the processes, work products, QA activities, and project management activities required to develop, use, maintain, and reuse an M&S application from birth to retirement. The M&S life cycle is created to modularize and structure an M&S application development and to provide guidance to an M&S developer, engineer, manager, and organization. The M&S life cycle contains different processes, which are Problem Formulation, Requirements Specification, Conceptual Model, Architecture Specification, Design Specification, Executable Sub models,
Simulation Model, Simulation Results, Presented Results, Certified Simulation Model, and V&V. The life cycle is iterative in nature. It is typically bounce back and forth between the processes.

One particular way to develop M&S application is using Component-based simulation (COBS), which is developed based on Component-based Software Engineering (CBSE) [Szy03] [Lau07]. COBS tried to model complex systems by dividing the initial system into components [Dal07]. These components are independent and connected with each other by the interfaces. Although there are many different views on what a component is and what its features are, it is general accepted that a component is like a black-box entity with well-defined interfaces and behaviors. This kind of component-based simulation may give gains in productivity and quality [Kul01]. It can help to develop complex and distributed systems deployed on a wide range of platforms, by plugging pre-defined components, instead of building these applications separately. It can reuse already tested and validated components [Ber06].

However, the common issue of all development processes mentioned above is as more and more resources are involved, the developing process become more and more complex. Each method is composed of different sub processes, which make a long life cycle. As a result, developing an M&S application is costly and requires years of development. In addition, different steps in each life cycle need different people from multiple domains. Collaboration and M&S related expertise among those people are required. Furthermore, most methods are difficult to balance the semantics of their advanced features, which makes them sometime ambiguous to use correctly and flawlessly in practice [Bur06]. In addition, most of the methods communicate only by defined means, not using web technology. It also does not support configurable EF development and varied resources management. Therefore, a way of fast developing these M&S resources as services as well as rapid integrating these services as M&S applications is highly needed.

2.3 Web-based Simulation

In order to facilitate the development of M&S applications, the M&S community has been interested in web technologies for many years. These efforts include WBS
(exposing M&S functions as web services) and CBS (the Cloud-based Simulation that integrates WBS and Cloud Computing). The basic idea of WBS is to put M&S resources on the Web, and to provide users with new information standards and communication protocols [Wan10]. As reported in [For14], web-based technologies are beneficial to non-experts, who can share M&S resources and reduce the time required for users to learn about the M&S tools. In recent years, CBS has started to become popular to facilitate the use of these varied M&S resources [Smi13]. CBS and WBS have some similarities. CBS is derived from WBS and it has been identified as a challenge in M&S [Tay12] [Tay13]. However, the work on WBS is not new. The early efforts in using WBS to support modeling, simulation and simulation results analysis can be traced back to Fishwick’s paper in 1996 [Fis96]. Between 1996 and 2000, the number of publications on WBS grew explosively. After 2002, the number of publications dropped off very quickly [Ong14]. Since WBS and CBS are similar, it is important to study what has happened to WBS.

In WBS, the simulator and its simulation environment are located remotely on a server [Ben04]. Users can submit their requests (with specified message/parameters) to the simulator through web servers, then simulation experiment runs remotely, and the results are returned to the user. WBS eases the sharing of the simulation resources: for instance, the simulation engines available on the server (without worrying about simulation environment setup and other software dependencies issues). WBS improves data accessibility, interoperability and user experience (non-expert users can access these services easily), allowing them increase productivity [For14].

Onggo and Taylor [Ong14] analyzed the problem of the decline of WBS. They stated that there was a mismatch between the main characteristics of the Web and the web service framework taken by the domain of WBS (which mainly uses SOAP-based WS). This mismatch resulted in that WBS cannot take full advantage of the features of the Web, including common standards, interoperability, ease of navigation and use, etc. Kuljis and Paul [Kul03] also supported this argument. They also argued that developing M&S applications using Cloud computing technologies must not simply re-implement existing simulation software.

Web services can be categorized into two main frameworks: SOAP-based [Pap07] (in which the service exposes an arbitrary set of operations), and RESTful [Ric07] (in
which we manipulate XML representations of Web resources using a uniform set of “stateless” operations. In WBS, SOAP-based WS are widely used in M&S systems, while few systems use REST-based WS as the web service framework.

SOAP-based WS are exposed as in Remote Procedure Calls (RPC). For simulation, the simulation functions are available as procedures on the server, and they are described in XML Web Service Description Language (WSDL) documents. A client can compile the WSDL into procedure stubs, and at runtime, the SOAP messages are wrapped in HTTP, and POSTs the SOAP message to the server. When the server receives the message, it extracts it using a procedure call and responds to the client in a similar way. SOAP has been widely used in both non-DEVS and DEVS systems. DDSOS [Tsa06] is a simulation framework by using SOAP-based WS to support the evaluation of large-scale distributed systems. SOPM [Bre09] is a SOAP-based modeling framework to build performance models of service-oriented architectures. SASF [Smi13] is a SOAP-based simulation framework for predicting the behavior of service-oriented systems under different configurations and loads. There are many SOAP-based WS used in DEVS simulators. In [Mad06] [Mad07] [Wai08], the authors defined a distributed DEVS simulation environment over SOAP. DEVS/SOA [Mit07] [Mit13] implements DEVS over a SOAP-based SOA framework, supporting a development and testing environment known as DEVS Unified Process. A similar work was proposed in SOAD (SOA-compliant DEVS), a simulation framework for modeling service-oriented computing systems [Muq10]. They developed a set of abstract DEVS models that conform to the SOA principles. In [Roc10], the authors developed a tool using SOAP-based web services to support the creation of interoperable and composable training simulations.

Instead, RESTful WS imitate the Web interoperability style, directly taking full advantage of the Web. They use universal accepted standards, uniform channels, are resource-oriented, message-oriented, and hide implementation. REST represents resources as URIs; any user can request the resource by HTTP method, and communicate with the resource via standard representation (like XML). Resources are manipulated using a fixed set of four operations: POST, GET, PUT, and DELETE. However, using REST for WBS is not as popular as SOAP. In DEVS systems, the RESTful Interoperability Simulation Environment (RISE) [Alz13] was the first RESTful
distributed DEVS simulator in the Cloud. The main objective of RISE is to support interoperability of distributed and heterogeneous simulations regardless of the model formalisms, model languages or simulation engines. In non-DEVS systems, Arroqui et al. [Arr12] developed an agricultural information system by using both RESTful WS and SOAP-based WS. From the comparison, they found that RESTful WS requires 24% less bandwidth for transferring data than SOAP-based WS do.

SOAP WS have the following shortcomings when compared to RESTful WS:

1) Client-Server interaction in SOAP WS is tightly coupled, and changes on the server result in complex code changes on the client [Wag12] [Alz13].

2) SOAP messages require XML wrappers on all messages, whereas REST does not. Therefore, SOAP messages are bigger than HTTP messages used in RESTful WS; hence, SOAP consumes more bandwidth [Wag12] [Alz09].

3) SOAP exposes details of the internal implementation. Developing SOAP WS needs the understanding of the detail implementation of the procedures, which makes them harder to develop [Wag12] [Wai10].

4) SOAP is based on RPC. Each RPC interface defines its own services with its own syntax and semantics, which limits the scalability of the framework while REST has a uniform interface because of its use of HTTP methods [Mul09] [Wai10].

5) SOAP WS also has security issue. SOAP/RPC messages are wrapped in an HTTP header; the server only understands the messages once they have been parsed. This allows malicious commands to get through the firewall undetected. In contrast, each resource in REST is assigned a unique URL, and it is handled on the HTTP/HTTPS firewall (making REST more secure) [Fen09].

In fact, a large number of RESTful WS have been designed as replace inefficient SOAP WS. According to more than 11,000 APIs registered by ProgrammableWeb, the trend toward SOAP is dying: today 73% of the APIs use REST, while SOAP is only 27% [Sir14]. These new services in REST are more scalable, interoperable and simpler [Mul09]. These shortcomings of SOAP mentioned above have made WBS unable to take full advantages of Web, which further made WBS decline in recent years. Therefore, when developing M&S applications, RESTful WS should be adopted as Web Service framework instead of SOAP WS.
Besides the Web framework constrains, WBS has other shortcomings. Current research about WBS focuses on only exposing simulation services. They lack the support to EF specification for covering the context under which a system is observed or experimented with. The simulators are usually pre-existed on the server; they do not support the configurable EF development. For example, if we want to add another simulator with a different environment, additional servers may be needed and the users have to manually configure the server again (e.g. dependencies, libraries, etc.). Besides, they only expose selective simulation functions as services, but do not consider the other types of resources; for example, they cannot manage models, tools, related files and functions. Therefore, a dynamic way with a configurable EF specification to develop the simulation resources as services is highly needed.

2.4 Cloud-based Simulation

CBS is a term used to capture the intersection between M&S and Cloud Computing. Cloud Computing can be used for building a simulation environment in a pay-as-you-go manner, reducing the costs in the development of M&S applications [Cay13b], and computing power can be increased or decreased according to usage. Cloud computing also allows the users to scale up or down the underlying infrastructure [Liu12]. We can use Cloud Computing to manage varied M&S resources and build different simulation environments according to the actual demands. Cloud Computing for WBS also uses advanced technologies such as load balancing, fault tolerance and security [Fuj10]. In [Ong14], the authors have pointed out the following Cloud-based scenarios for M&S: 1) run simulation on Cloud infrastructure; 2) create and control models in the Cloud; 3) run experiments of simulation models on Cloud infrastructure; 4) manage and control the simulation development lifecycle; and 5) control over storage and the execution simulation engine.

Onggo and Taylor [Ong14] argued that developing M&S resources in the Cloud is not simply done the same way as WBS does: reimplementation of existing M&S resources using Cloud Computing. Johnson and Tolk [Joh13] identified five challenges that have to be addressed in CBS (i.e. technical, governance, security, business and
conceptual). Cayirci [Cay13b] showed how various technical perspectives have been addressed, but Liu et al. [Liu12] argue that the current technical solutions are neither widely used nor practical. In fact, there is no implementation trying to develop and deploy varied M&S resources as services in the Cloud.

In this research, we will show how Cloud Computing can help us to develop MSaaS (Modeling and Simulation as a Service), a concept that received some attention recently. MSaaS focuses on delivering services related to M&S using Cloud Computing [Cay13a], exposing models and simulation functions as services. MSaaS takes advantages of Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [Tsa11], hiding the underlying infrastructure, platform and software details from the users.

Recent research focused on how to use Cloud and MSaaS for developing M&S applications. There are three groups of researchers, which focus on different aspects:

1) **Managing M&S resources**: they use the Cloud to share and manage M&S resources. Sliman et al. [Sli13] proposed an MSaaS platform called RunMyCode, which allows scientists to share their code associated with their research publications in the Cloud. In [Liu12], the authors presented the work on deploying existing M&S software into the Cloud.

2) **Supporting Parallel and Distributed Simulation (PADS)**: Li et al. [Lit09] proposed a Grid-based platform called Cloud Simulation Platform, and summarized twelve key technologies for the development of CSim. Fujimoto et al. [Fuj10] discussed the benefits and challenges associated with executing PADS in the Cloud. Taylor et al. [Tay14] introduces the CloudSME simulation platform to transplant legacy Grid-based simulation software in the Cloud.

3) **Building applications in the Cloud**: they focus on the development of M&S applications using the Cloud. Bruneliere et al. [Bru10] presented an approach to build applications by using model-driven engineering and Modeling as a Service (MaaS) in the Cloud. The Lanner group [Lan14] designed a system called L-SIM 2.0 in the Cloud to simulate Business Process Management systems. CIMdata [Cim15] proposed a simulation application using the Cloud in the area of commercial product design.
However, the use of Cloud Computing and MSaaS is still in a preliminary stage and a universal definition of MSaaS does not exist [Cay13b] [Gar11]. In [Cay13b], the author discussed the differences and relations among MSaaS, IaaS, PaaS and SaaS. In particular, the author views MSaaS as a special form of SaaS. The author classified three types of MSaaS: modelling as a service (services to develop models), model as a service (services to access models) and simulation as a service (services to run simulations). However, this kind of classification of MSaaS only considers models and simulations; it does not consider other kinds of M&S activities (e.g. sharing models and experiments, storing supported data and reproducing supported functions). For example, in order to complete a simulation process, some supported functions are important (e.g., data collection for the model inputs, simulation results analysis). However, the current WBS and CBS lack the support for users to expose these supported functions and reproduce the experiments related to these functions.

Our research effort can also help in dealing with the reproducibility and environment configuration crisis: in [Kur06], the author analyzed many papers from the ACM MobiHoc symposium between 2000 and 2005, and found that less than 15% of the simulation results were reproducible. In [Sli13], the authors argue that research community cannot publish simulation models along with the published papers; as a result, they cannot reproduce these simulation results easily as stated in the published papers. There are several reasons for this. Then one must reproduce the simulation in the same way as the original, which requires that one must have the access of simulation services for the same simulation files, inputs, etc. In this sense, existing WBS and CBS lack in simplicity in terms of environment configuration, which makes it hard to simplify the simulation process [Byr10]. Most services provided by WBS are delivered on local servers, so these services rely on heavily the capacity of the servers. Due to the increasing demands of underlying infrastructure for simulation experiment, it is hard to set up new servers efficiently with a higher compute power or a larger storage capacity. Users have to migrate and backup by themselves, as well as purchasing infrastructure and IT services. In CBS, even if users can use Cloud services to get scalable computer power or storage capacity, carrying out an experiment can be still complicated. Users still need to select correct services to set up the experiment environment, prepare input, control the
experiment lifecycle and analyze results [Guo11]. It is even harder when there are varied M&S resources that cannot be easily exposed as services. Therefore, a simplified way to develop these resources as services in the Cloud is needed.

2.5 Mashups

Besides the different M&S services generated by WBS and CBS, there are also many open Web APIs that may be helpful for the M&S application. These APIs have the potential to be used for M&S purposes. We can compose them for better user experience and richer applications (e.g. Google Maps, YouTube, weather forecast, GIS information, etc.). Due to the fast development of web technologies, there are emerging various kinds of Web APIs (like REST, SOAP, JS, and XML-RPC). Figure 3 shows the protocol usage of the current Web APIs based on the 10310 APIs available on ProgrammableWeb.com [Pro14].

![API Protocol Usage](image)

**Figure 3.** Web APIs protocol usage (from ProgrammableWeb.com)

From this figure we can see, the portion of Web APIs based on REST is much bigger than the other protocols (69%). This is due to the convenience in handling the service resources with URL and HTTP, so its utility for mash-up is very high among others. However, there is no standard description language for RESTful web services. WADL is a popular language to describe the syntax of RESTful web services. Other formats like Swagger, WSDL 2.0. Swagger is a specification to document and visualize RESTful APIs. WSDL is originally designed for describing SOAP web services; WSDL 2.0 is its latest version that is extended to allow RESTful web services. Besides, many IT companies (i.e., Google, YouTube, Flickr, etc.) provide HTML pages for describing their
RESTful APIs, like (Google, YouTube, and Fliker) [Lee11]. As discussed above, due to the limitations of SOAP compared to REST (e.g. tightly coupled, bigger payload, exposes internal implementation, limited scalability, security issue), developing RESTful API is more and more popular.

However, there has been no research showing how to integrate these useful web services (e.g. WBS, CBS and Web APIs) when developing M&S applications. Each type of web service APIs focus on different logic and use different web frameworks (e.g. REST, SOAP). In a result, it may require specific techniques to describe and invoke the APIs, which makes it difficult to provide support for the integrating of APIs. In addition, in order to simplify the developing process, this kind of service integration should support the service execution and visualize service results.

Mashup technologies in Web 2.0 can be used to solve this integration issue and simplify the M&S application development. Mashups integrate different services from the web, using content from more than one existing source to create a new value-adding application [Bal08]. Mashups integrate heterogeneous data, application logic (exposed as services in general), and UI components (e.g. widgets). A large number of mashup techniques and tools have been developed in both industrial development and academic research [Geb12]. Many industrial companies have developed their own commercial mashup tools, like IGoogle (http://www.igoogleportal.com), and Yahoo! Pipes (http://www.pipes.yahoo.com). They are based on the visual connection of components of heterogeneous data at the enterprise level, offering Do-It-Yourself guidance to meet user requirements [Liz09]. In addition, many academic efforts focus on mashups. Many of them use End-User Programming, focusing on the composition and integration of web sources for new purposes. Mashroom [Wan09] uses nested relational models and provides mashup operations like merge and filter over tables. In [Agh12], the authors use native language programming in mashup components, linking different logic together.

The fundamental element of current mashup technologies is the widget, a small processing unit for performing single purpose task such as fetching, parsing, formatting and visualizing data [Tri14]. For instances, DERI Pipes [Lep09] enables users to build widgets to process data from different sources (e.g. RDF, SPARQL, XML, HTML). In [Tri14], the authors proposed an open mashup platform with linked widgets created
freely by users that can be discovered and combined easily. WireCloud [Zah14] is an open source mashup platform provided by the FI-WARE project, which can implement widgets in JavaScript (JS) and HTML5 and build mashups by “wiring” widgets.

Figure 4. Life-cycle models of (a) current web applications and (b) mashups, adapted from [Dan11]

As discussed earlier, the common issue of the traditional development processes is that these processes make developing applications more and more complex. Traditional development methodologies contain different sub processes, which makes a long life cycle. In contrast, Mashup provides a lightweight development process; it tries to compose an application from existing content and functionality. In [Dan11], the authors have compared the traditional development process and mashup process, as shown in Figure 4. Figure 4(a) shows the traditional development process, which contains much more sub processes, like Requirements analysis, Design, Implementation, Testing and evaluation, Usage and maintenance, which is time-consuming and hard to maintain.

Mashup development process shown in Figure 4(b) contains fewer sub processes, namely Discovery and selection, Mashup composition, Usage and maintenance. Mashup is a lightweight development process and allows continuous improvement and evolution. Mashup is prototype-centric and iterative approach: the user mashes up exiting resources and runs the result to check whether it works; in case of unsatisfactory results, the user fixes the problems and runs the mashup again, until it is satisfactory. The author also mentioned that the key of mashup development is a mashup platform to select and compose existing resources, along with a set of open Web APIs that support features and available data.
However, no one has ever tried to develop such widgets and mashup applications in M&S by integrating available services (e.g. available services from WBS, CBS and Web APIs). Many of the current mashup techniques and tools cannot work directly for M&S because: 1) they are domain specific (i.e., they are useful only for single or limited problems in specific domains); 2) they have been discontinued; 3) they are limited (the widgets do not support MSaaS and different kinds of Web APIs like SOAP and REST). Based on this, in this research, we propose a new mashup method focused on the process of developing M&S and integrating varied M&S resources as services, including MSaaS services and open Web APIs.

2.6 Semantic Mashup and Ontology

M&S mashup can simplify the development of M&S applications by integrating existing resources and services. However, due to the variety of M&S resources, there may be thousands of M&S resources and services, which make it hard to select the right ones based on their “meanings”. M&S usually adds a semantic layer to the M&S resources and annotates them to an external ontology. In this section, we are going to survey the current background in this area.

2.6.1 Semantic Mashup

In order to know the meaning of the resources to be integrated, there is a new concept called “semantic mashup”. Semantic Mashup is for the semantic selection and composition purpose [Mal12]. A Semantic Mashup is a mashup whose combined APIs are supported by a semantic layer that allows to select and compose them in an automatic and unambiguous way. As summarized in [Yuj08] [End13], a number of mashups efforts, such as approaches, techniques and tools have emerged. However, there is no research using semantic mashup to select M&S related resources and services in M&S communities.

There are two general approaches to do the Semantic Mashup: semantic web language mashups and semantic annotation mashups. The first type (i.e., the semantic web language mashups) is to use a semantic web language to define a complete ontology
for describing web services and their semantics in a single block. The basic idea of this type is to use the defined ontology of web service APIs as an upper level standard, then follow the rules of the ontology to define web services. This type of mashup uses the reasoning power and querying method of the chosen semantic web language for web services composition and integration. The examples of this type are RDF [Rdf15], OWL [Owl15a], OWL-S [Owl15b], WSMO [Wsm15], SWSF [Sws15], SSWAP [Ssw15], Jena [Apa14].

The second type (i.e., the semantic annotation mashups) is to annotate web services with semantic information to some existing domain ontology. This type is also very active in semantic mashup, assisting developers in quickly discovering and matching mashup components. The examples of this type are WSDL-S [Wsd15], SAWSDL [Saw15], SBWS [Bat08], SA-REST [She07], SAWADL [Mal12], SWEET [Mal09].

However, both of the mashup techniques need to pre-define ontology (either semantic web or domain) before performing the mashups, which is a major drawback. Most ontologies are developed manually through the collaboration of highly skilled domain engineers and ontology experts. Ontology building is a complex and time-consuming task. Besides, the ontology under use may not cover all the concepts of the use case. People using different ontologies could generate different results due to the mismatch of the ontologies. In addition, there is no research using semantic mashup in M&S. Users still need to learn how to use the ontologies in order to annotate their mashups.

2.6.2 Ontology in M&S

The consideration of semantic in M&S has been in use for many years. In [Tol03], the authors introduced the Levels of Conceptual Interoperability Model (LCIM), which identifies seven levels of interoperability. In LCIM, the semantic level was defined as the meaning of models, requiring a reference model for common information exchange. In [Zei07], the authors use pragmatics (data use in relation to data structure), semantics (meaning of terms) and syntax (data structure and rules). In [Mit09], the authors proposed a layered M&S architecture. A modeling layer is related to semantic composition (focusing on independent model representation) and a collaboration layer (focuses on common understanding). In [Tol05b], the authors used a Common Reference Model
(CRM) as ontology for model composition. In [Sil07] the authors presented a method for using the knowledge encoded in ontologies to facilitate the development of simulation models. Discrete-event Modeling Ontology (DeMO) [Mil04] providing a precise description of simulation models with hard semantics, is an upper ontology that details events, activities and processes. PIMODES [Lac06] is the development of the Process Interaction Modeling Ontology for Discrete-event Simulations. COSMO [Teo08] is the development of the Component Simulation and Modeling Ontology. In [Zei08], the authors focus on Integration of Modeling and Simulation. They propose a standard for interoperability based on linguistic categories along with the DEVS. The standard contains three levels: Syntactic, Semantic and Pragmatic, taking into consideration of semantic and ontology in M&S. However, the ontologies that they use are domain specific and are not for selecting M&S resources.

There is also semantic based research on managing M&S services. In [Rab12], the authors propose an approach that builds on an existing Semantic Web Platform for Modeling and Simulation that supports planning (especially during the information preparation and results evaluation). These constraints are built as semantic rules utilizing Semantic Web technologies. In [Loz07], the authors describe an initial work designing and developing a semantic based distributed repository for secure sharing of simulation models as services, components and related resources such as computer resources. However, the ontology and matching method is immature and needs further improvements. In [Riz08], the authors share an idea of using semantic links in their integrated modeling framework. In [Sil07], the authors present a method for using the knowledge encoded in ontologies to facilitate the development of simulation models. It suggests a technique that establishes relationships between domain ontologies and a modeling ontology and then uses the relationships to instantiate a simulation model as ontology instances. In [Tsa11], P4-SIMSSA comes with an ontology system and an innovative tenant related policy specification for Simulation SaaS. In [Lit13], the author use OWL-S to define layered ontology (resource, function, simulation) for compositing their SOAP-based WS in the Cloud simulation platform.

However, the ontologies used in these approaches are fixed beforehand, thus lacking the automatic development of ontologies. Defining a single conceptual model to
represent all the simulation models is not practical [Alp12]. Most ontologies are
developed manually by skilled ontology experts, and ontology building is complex and
time consuming. In addition, pre-defined ontologies are limited to specific cases and
rules, which are hard to understand and they may not cover all concepts for the users.

2.6.3 Ontology Learning

Ontology construction is an expensive and complex task that needs many skilled
domain experts and ontology engineers. A number of ontology-learning methods have
been proposed for automatic acquisition of semantic information. Hess and Kushmerick
[Hes04] employ Naive Bayes and machine learning methods to classify WSDL files in
manually defined task hierarchies. Dong et al. [Don04] employ a clustering method that
clusters parameters present in inputs and outputs of operations into parameter concepts.
However, these clusters are not a semantic ontology but just a set of simple synonym
words. Sabou et al. [Sab05] propose an automatic extraction method that learns domain
ontologies from textual documentation attached to web services. The limitation of this
method is that it is limited to human-readable documentation, which is not common
means of web service descriptions. Guo et al. [Guo07] leverage relations between words
in phrases to establish ontological relationships between acquired concepts. However, the
authors tackle only a one-to-one mapping solution by aligning the generated ontology
fragments, and take advantage of active domain experts. Lee and Kim [Lee11] provide an
ontology-learning method to link web services (REST, JS, XML-RPC, Atom). The idea
of this paper is to learn and build an ontology of API input/output operations
automatically, based on predefined relationship rules; then for given an API, it proposes a
similar search method to match it with the registered ones.

However, these learning processes are still hard to operate and are restricted to
some particular domains; in addition, they are usually based on some predefined rules
that do not consider all cases of semantic, syntactic and structure issues when finding the
similarity of two resources.

2.6.4 Tagging System

Tagging system can benefit the selection process of M&S resources for semantic
mashup. In Web 2.0, tagging systems (also termed as Folksonomies) [Wal14] can be seen
as large collections of informal semantics - many casual users cooperate to label objects with free-form tags of their choosing. Tags are used by the users to characterize the content of the resource and for communication of the content to both the user who defined the tag and to other users. The more common a tag is used for a resource by users, the more accurate the tag will represent the concepts of the resource; thus, tags will ultimately cover domain concepts [Hey06]. A number of Web 2.0 applications like Flickr, del.icio.us and Twitter now employ tag-based descriptions to facilitate search and retrieval of resources and users. Tagging lowers the entry barrier to user participation and cooperation according to their own vocabularies, and largely enriches the ontology for the Web resources. Tagging systems are increasingly popular, simple, intuitive and powerful to label and organize the big data, which can be viewed as large collections of informal semantics [Sol11].

Such tag-based service semantics will be more natively representative and provide easy-of-use for selecting resources. Based on this idea, iMashups [Liu13] proposes an approach for composing data-driven mashups. It explores semantics from popular tags. The data is described by tags, and the semantics are determined by the tag-based taxonomy. Another tag-based approach, like [Bou08], proposes a method for composition in flow-based information processing systems, where application design and component development are based on tag-based descriptions. The major problem of the traditional tagging system is the lack of structure/hierarchy. The tags in a normal tagging system are unstructured with same-level annotated labels, lacking the support of the concept abstractness of tags. This makes it difficult to navigate the broader (e.g. parent tag) or narrower (e.g. child) tags with better representation. More importantly, loosely related tags make it unable to find the relations among different resources, not to mention mashups [Hey06]. Furthermore, most of the tag-based approaches mentioned above depend on a manually built predefined tag hierarchy database. Furthermore, their similarity finding of two resources is very limited, ignoring most of semantic, syntactic and structure issues.

Knowing the main issues of traditional tagging systems is that the lack of semantic structure among tags, researchers try to use ontology-learning technologies to build structured folksonomy by identifying meanings of tags, find the relationships among
them and building a structure of them. Current research focused on learning the tag structure based on ontology-learning technologies. Existing methods for this can be organized into four categories: 1) *Semantic linguistic resource* approaches [Ber10]: they link tags to a concept in an existing ontology. 2) *Syntactic distance* approaches [Cat08]: they find relations of tags by checking their similarity based on the syntactic variations, not considering complex semantic issues [Sol11]. 3) *Clustering/co-occurrence* approaches: they use machine-learning techniques to cluster tags into different groups, but the coarse-grained classification is not suitable for the heterogeneous M&S resources. 4) *Network-based* approaches [Son11]: they use graph/network techniques with the probability and approximation techniques to build the structure, not considering the semantic issues comprehensively, such as the syntactic distance or linguistic resources.

However, these tag structure-learning methods are not directly suitable for the M&S resources mashups, because they mostly focus on grouping the tags rather than providing a tree-like hierarchy, which would be needed in the case of M&S resources and services. In brief, although semantics and ontologies are being used in M&S, it is still hard to get the meaning of the M&S resources. Little has been done about automatic learning domain-specific ontologies for these resources. Tagging systems can help with this issue, but there is no method on the use of tagging system and trying to learn tag hierarchy as ontologies in M&S.

### 2.7 Discrete Event System Specification and Related Resources

In the M&S application development process, we are interesting in the broad view of M&S resources, which refers to all of the resources related to M&S. In particular, we will focus on the Discrete Event System Specification (DEVS) formalism and its related resources.

DEVS [Zei00] is M&S specification that is aimed to study discrete event systems. DEVS is a well-defined formal methodology that allows one to define hierarchical modular models. As in any discrete-event simulation, the models change their state only at discrete points in time, upon the occurrence of an event. The models consist of components connected together through external input/output ports where events are exchanged among models. DEVS expresses a system as a number of connected
behavioral (atomic) and structural (coupled) components. An atomic model is the lowest level in which we can define behaviors of any entity, while a coupled DEVS model integrates different DEVS models together through well-defined input and output interfaces. A DEVS model is event-driven, by external or internal events. A coupled model allows one to convert the outputs of a model into inputs of events for the other models, and to handle inputs/outputs to and from external models.

Cell-DEVS [Wai09] is an extension to DEVS that defines each cell in a cellular model as an atomic DEVS model that changes state according to the values of its neighbors following the rules defined by a local computing function. Cell-DEVS describes n-dimensional cell spaces as discrete-event DEVS coupled models, where each cell is represented as a DEVS atomic model. Furthermore, it defines timing constructions rules for each cell, allowing explicit timing delays, asynchronous model execution, and integration with other DEVS models.

CD++ [Wai09] is an object-oriented modeling and simulation toolkit capable of executing DEVS and Cell-DEVS models. For each DEVS atomic model, users need to implement the various functions as required by the DEVS formalism in a C++ class, which is then integrated into the modeling hierarchy during compilation. For DEVS coupled models and Cell-DEVS models, users can specify the coupling information as well as other attributes of cell spaces in a model configuration file using a built-in script specification language. CD++ has been successfully applied to model complex systems and it is widely used in different domains (e.g. environmental systems, architecture and construction, networking and communications, biology and medicine, hybrid and continuous systems, traffic systems) [Wai09]. There are many efforts trying to integrate CD++ with other related simulation resources (e.g. data collection, Web-based Simulation and visualization). In [Wan15a], a united approach to integrate CAD/BIM tools, Crowd models using Cell-DEVS modeling, Web-based Simulation, and 3D visualization is presented. In [Wan14a], an integrated architecture to GIS data collection, Cell-DEVS modeling, WBS and Google Earth visualization is presented.

DEVS supports the development and integration of M&S applications. However, DEVS shares the issues of developing M&S applications using new technologies like web services and cloud computing. First, most WBS research in DEVS is based on SOAP
WS, which has the issue of Web Framework constrains compared to REST WS; in addition, these SOAP WS only focus on simulation functions, but not for varied M&S resources. Second, since CBS is infrequently studied in M&S, using cloud computing in DEVS is also not practical to configure a simulation environment; in particular, no actual implementation has been made for developing configurable EF for running DEVS simulations in the Cloud. Third, there are varied DEVS-related resources and services, however, no effort has been made to simplify integration of these resources; in particular, no mashup research has been studied in DEVS community to integrate, execute and show varied DEVS resources. Fourth, in regard to the meanings of DEVS resources, the current semantic efforts depend heavily on pre-defined ontology, there is no effort trying to learn domain-specific ontology based on given DEVS resources.

2.8 Summary

In summary, though web technologies like web services and cloud computing have been used in M&S for many years, it is still a complicated process to develop M&S applications. In order to simplify the development process of M&S applications, we still need to handle the issues caused by WS frameworks like SOAP WS, the variety of M&S resources, the difficulty of configuring experiment frameworks in the Cloud, the integration of varied M&S services and resources, and selection of resources based on their semantics. In the following chapters, we will introduce our approach and detailed solutions for solving these issues.
Chapter 3: The MAMSaaS Architecture

In order to simplify the development and integration of M&S applications and solve the new issues when using web technologies, we propose a new methodology and architecture, to deploy, identify, select, compose and invoke varied simulation resources as services. To do so, we propose using MSaaS to share the varied M&S resources as services. We regard MSaaS in a broad view that it delivers everything as a service using RESTful principles. In other words, MSaaS is able to deliver everything related to M&S as a RESTful WS by using Cloud Computing. MSaaS are built on top of the Cloud services (i.e. SaaS, PaaS, and IaaS), considering that MSaaS is a special form of SaaS [Cay13b]. Figure 5 shows the relationship between MSaaS and the Cloud services.

![Figure 5. Relationship of MSaaS and Cloud services](image)

Users can share a model by using Models as a Service, reproduce experiments by using Experiments as a Service, reconfigure Experimental Frameworks by using EF as a Service, run a simulation by using the corresponding Simulator as a Service, share the supported data by using Data as a Service, and execute supported functions by using Function as a Service. These MSaaS services take advantages of SaaS, PaaS and IaaS. SaaS provides software on-demand and application services to be used for MSaaS. PaaS provides a computing platform that facilitates the development, deployment and management of the applications for MSaaS. IaaS delivers computer and storage infrastructure as a service for MSaaS to access.

Based on these ideas, we propose a novel architecture, named Mashups Architecture with Modeling and Simulation as a Service (MAMSaaS) to develop,
identify, select, integrate, and execute varied M&S resources as MSaaS in the Cloud. Figure 6 shows its layered architecture, as follows:

![Diagram of MAMSAAS (Mashup Architecture with Modeling and Simulation as a Service)](image)

**Figure 6. MAMSAAS (Mashup Architecture with Modeling and Simulation as a Service)**

1) **Cloud Layer**: it is responsible for supporting Cloud infrastructure and deploying MSaaS. The Cloud infrastructure includes Cloud *compute* units (for building and executing simulation experiments) and Cloud *storage* units (for sharing M&S resources). This layer is also responsible for deploying user-provided M&S resources as MSaaS in the Cloud by using the *MSaaS middleware* on-demand, which is a platform for development, deployment and management of MSaaS. In addition, this layer supports a VM-like *MS-Image* with EF configuration for easy launching and maintaining simulation environments.

2) **Box Layer**: it is responsible for developing user-provided M&S resources as mashup components (termed Boxes). Boxes can have different categories (e.g. MSaaS from the Cloud Layer, existing open WebAPIs, widgets, and operators). Each box is
identified by a uniform box signature; also, each box has its own function for handling input messages and has its own visual form.

3) **Tag Ontology Layer**: is responsible for maintaining and learning the tag-tree ontology according to the boxes components. It provides new tag-mining and ontology-learning algorithms. The tag-mining algorithm can mine tag signature from boxes that best represent their meanings. The ontology-learning algorithm can construct domain-specific tag-tree ontology from tag signatures of boxes, without the need of pre-defined ontologies.

4) **Wiring Layer**: it is responsible for connecting boxes into a mashup. Boxes can be linked with each other by their inputs/outputs that are identified in their box signatures. A same box can be reused and re-wired in different mashups for new purposes. This layer plays as a link between boxes and mashup applications. This layer can also work with Tag Ontology Layer for semantic box selection. The idea is learnt tag-tree can suggest “wirable” (composable) boxes to a given box. In a result, boxes can have “dataflow” fashion with much more meaningful connections based on their tags.

5) **Mashup Application Layer**: it is responsible for selecting and wiring boxes as mashups in user workspaces for particular M&S application requirements, and running mashups at run time. This layer provides a user-friendly GUI to allow user to manage boxes, select boxes and wire them as mashups, execute and visualize mashups at run time.

This architecture supports collaborative development. It has different kinds of users (shown in Figure 7): Cloud Administrators, Box Providers, Ontology Managers, Wiring Engineers, and Clients.

1. The **Cloud Administrators** are the workers who manage the cloud computing and cloud storage units. They support the management of the underlying cloud infrastructure for the upper layers. They can use MS-Images to launch MSaaS middleware on different cloud computing units. They can also monitor the health of the cloud infrastructure.

2. The **Box Providers** are specialists from different fields who can provide a variety of box components. The boxes can be in different categories, like MSaaS (using MSaaS services), WebAPI (using Web APIs), Widget (visualizing specific data) and
Operators (handling mismatches when wiring). In addition, each box is packaged in its own signature, function and view. Anyone with minimal web development knowledge can easily develop such boxes.

3. The **Ontology Managers** are workers to run the tag-mining algorithm for adding semantics to boxes. They are also responsible for running the tag-learning algorithm to obtain the tag-tree ontology. Note that they do not have to be the experts of the ontology; the tag-tree is learnt automatically. Their tasks mainly focus on the execution and management of these ontologies; e.g. learning a new tag-tree, or saving them in the MSaaS middleware.

4. The **Wiring Engineers** are experts for designing and managing the data flows of the boxes. These wiring engineers can benefit from the learnt tag-tree for semantic resource selection. Note they can use Mashup Platform to simplify the wiring activity by simply dragging and dropping selected boxes.

5. The **Clients** are the end users, who can use the mashup platform to select boxes, modify wirings, execute mashup application and see the results on the web browser. They can easily modify the mashup by adding new boxes or rewiring their connections.

Figure 7. **Users in the integrated layered architecture**

The MAMSaaS architecture divides the responsibilities of the mashups development task into different levels. It allows users to focus on their own tasks, and it
makes it easy for the users of different layers communicate and collaborate with each other. At the beginning, it starts with the *Cloud Administrator*, who configures the cloud infrastructure. These administrators add some basic dependencies and configurations on these infrastructures. To do so, they can take advantages of many cloud platforms, like the most popular ones (Amazon EC2, Google App Engine, and Microsoft Azure). After that, the *Box Providers* can deploy their box components (from either MSaaS or Web APIs) on the cloud. For the MSaaS services, the provider only needs to provide their EF specification and related files. The MSaaS middleware then will deploy these resources as services on the Cloud automatically. The *Clients* can reuse these newly deployed MSaaS WSs to reproduce the experiments. For other types of box, the Box Providers can quickly package the box using basic web development knowledge (e.g. XML for its signature, JavaScript (JS) for its function and HTML/CSS for its view). Then, the *Ontology Manager* can select interested boxes, mine tags for them in order to get their tag signatures and learn tag-tree from them for the further semantic selection purpose. Next, the *Wiring Engineer* will build mashup by dragging and dropping boxes; in addition, they can get wiring suggesting based on the learnt tag-tree. Finally, the *Client* can use the Mashups Platform to see the mashups application; they can also manage the mashups or execute it the Mashups Platform.
Chapter 4: Cloud Middleware and MSaaS

The Cloud Layer and the MSaaS middleware play fundamental roles in the MAMSaaS architecture presented in Chapter 3. They are used to develop and deploy M&S resources as MSaaS in the Cloud. The Cloud Layer supports Cloud infrastructure and it contains MSaaS middleware. The MSaaS middleware is responsible for developing MSaaS. This middleware supports a variety of resources and can solve the structure constraints caused by the web service framework. In this chapter, we will discuss their detailed design and implementation.

4.1 Cloud Layer

![Figure 8. Basic Structure of the Cloud Layer](image)

The Cloud Layer, presented in Figure 8, provides IaaS services for the upper layers of the MAMSaaS. Physically, all the resources will be stored and all the experiments will be executed in this layer. In general, this layer supports the Cloud infrastructure, and it provides IaaS APIs. We consider two kinds of Cloud infrastructures: Compute Units and Storage Units. The Compute Units are used for executing experiments (either for simulations or for supported functions), while the Storage Units store M&S resources (e.g. the models, the experiments, the EFs, the supported data, and the supported functions). The IaaS APIs (divided in IaaS compute APIs and IaaS storage APIs) are the
APIs for these Cloud infrastructures. They can manage the underlying Cloud
infrastructures (e.g. launching/releasing new Compute Units, saving/retrieving resources
from a Storage Unit).

Compute Units contain experiment environments (runtime environments for the
simulations or supported functions). Each Compute Unit can get an experiment from a
Storage Unit, and reproduce it in the corresponding environment. After the experiment
finishes, the Compute Unit saves the outputs back to the storage.

Storage Units are storage volumes in the Cloud. Each M&S resource is stored in a
Storage Unit, which can be replicated to protect the data from failure, providing high
availability and accessibility.

IaaS APIs allow upper layers of MAMSaaS to access the Cloud. In recent years,
many Cloud providers support IaaS (e.g. Amazon Web Service – AWS, Google Compute
Engine, or Microsoft Windows Azure). However, it is still hard to use these IaaS
services. For instance, users need to specify the IaaS provider, deal with the credentials,
configure the security group, and set up the connection with the Cloud. The IaaS APIs
here provide an easy way to use these services. There APIs include the IaaS Compute
APIs (to manage the Compute Units) and IaaS Storage APIs (to manage the Storage
Units).

![Figure 9. MS-Image](image)

In order to provide a simple way to configure environments, we define the concept
of **MS-Image** (Figure 9). An MS-Image is a copy of a Compute Unit. It consists of the
MSaaS middleware, the simulators and the function environments. An MS-Image is initially saved in a Storage Unit, and it can be launched as any number of Compute Units. An MS-Image is a special version of a Virtual Machine (VM) image in the Cloud. It can be viewed as a customized VM.

As seen in Figure 9, an MS-Image consists of:

1) **MSaaS middleware**: it uses the IaaS APIs to develop varied M&S resources as services. It separates the underlying infrastructure and MSaaS services. It wraps the simulators and functions provided by users.

2) **Simulators**: the MS-Images configure the environments for simulators. Different MS-Images can configure different simulators. MS-images expose these simulators as MSaaS services.

3) **Environments for supported function**: MS-Images can hold environments for different supported functions (e.g. Java, C++, and Python). Users can deploy their own functions at run-time. MS-images expose these functions as MSaaS services.

The MS-Image simplifies the deployment of environments. Users do not need to worry about the installation and configuration of the environments. To run experiments, they find an MS-Image, and then launch Compute Units from it. After that, they can use the MSaaS middleware. For instance, let us assume that we need to run a simulation 50 times. We can find an MS-image that contains the needed simulator, and launch 10 Compute Units of that image. After that, we can execute 5 simulations on each Compute Unit. When the simulations complete, we can release these 10 Compute Units to reduce costs.

In MAMSaaS, the Cloud administrators are responsible to manage the underlying infrastructures. To do so, they can use the IaaS APIs and MS-Images. In addition, they can use other Cloud services (e.g. PaaS and SaaS). For instance, the PaaS load balancing service can evenly distribute messages among Compute Units. The SaaS Simple Notification Service can send notifications via email when infrastructure is unhealthy. Other MAMSaaS users (e.g. Box providers, Clients and Wiring Engineers) do not operate this layer; instead, they use the MSaaS middleware directly.

We implemented a prototype of this Cloud Layer using Amazon Web Services (AWS). We chose AWS as it supports lower level operations, so we can completely
control the underlying infrastructure. In particular, we use *Amazon EC2 Instances* for Compute Units, *Amazon S3* for Storage Units, *Amazon Machine Images (AMI)* for MS-Image, and *AWS SDK* for IaaS. Amazon EC2 Instances provide elastic compute capacity in the Cloud. Amazon S3 is a Cloud storage service. AMI allows launching Amazon EC2 Instances, and AWS SDK provides IaaS for Amazon EC2 and S3.

We launch EC2 Instances by specifying the *instance type* and *AMI*. Instance types have different combinations of CPU/GPU, memory, storage, and networking capacities. For example, *Amazon Micro* is a low-cost instance with a small amount of CPU. Figure 10 shows a screenshot of the current implementation in the AWS Management Console. We have two instances (shown in the top-left of Figure 10): *MSaaS Instance 1* and 2, each with its own IP address. Each instance is launched by the same customized AMI (*MS-Image1*), shown in the bottom-right of Figure 10. This AMI contains CloudRISE (the implementation of MSaaS middleware) and different versions of CD++ (e.g. CD++ and CD++3.0). For the function experiments, this AMI uses Amazon Linux AMI 2013, which covers environments for most functions (e.g. Java, C++, Python, PHP, and Tomcat).

![Figure 10. Amazon EC2 instances for MAMSaaS](image)

We use Amazon S3 to store varied M&S resources. Amazon S3 is able to store any amount of data at any time from anywhere on the web. It is simple and highly scalable.
Amazon S3 can be seen as a Cloud-based file system. It stores resources within buckets. Each resource is identified by a URL. For example, in a URL like http://s3.amazonaws.com/MSaaS/Instance/.../ms-image1.ami, MSaaS is the name of the bucket and Instance/.../ms-image1.ami is the key of the resource. In the current implementation, we use one bucket for all resources, named MSaaS (the number of buckets does not matter; we can also use different buckets). The structure of the resources is hierarchical (shown in Figure 11). The first level of the folders are /Model, /Simulator, /Experiments, /SupportedData, /SupportedFunctions, and /Instance. Each folder has sub folders.

![Figure 11. Amazon S3 resource folders for MAMSaaS](image)

The Cloud Layer provides IaaS APIs. In the current implementation, we use AWS SDK as IaaS services, and implemented IaaS APIs. For example, we implemented the IaaS Compute APIs in the class AWSEC2Instances.java and the IaaS Storage APIs in class AWSS3Storage.java.

The class AWSEC2Instances.java provides IaaS Compute APIs for AMI and EC2 instances (see Table 1). It is able to set up configuration (setUpConnection), create a customized AMI from an instance (createImage), launch new EC2 instances (runInstances), and release EC2 instances (terminateInstances). For example, when we want to launch an instance, we can call the function runInstances (String imageld, String instanceType, int count), which will create number of instances (count) with required AMI (imageld) and instance type (instanceType).
Table 1. IaaS Compute APIs: AWSEC2Instances.java for Amazon AMI and EC2

<table>
<thead>
<tr>
<th>IaaS Compute APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean setUpConnection (String accessKey, String secretKey, String securityGroup)</td>
<td>Sets up connection to an EC2 instance.</td>
</tr>
<tr>
<td>Boolean createImage (String instanceId, String imageName)</td>
<td>Creates a customized AMI from an existing instance that is either running or stopped.</td>
</tr>
<tr>
<td>Boolean runInstances (String imageId, String instanceType, int count)</td>
<td>Launches specified number of instances using an AMI.</td>
</tr>
<tr>
<td>Boolean terminateInstances (ArrayList instanceIDs)</td>
<td>Shuts down one or more instances.</td>
</tr>
</tbody>
</table>

Table 2. IaaS Storage APIs: AWSS3Storage.java for AmazonS3

<table>
<thead>
<tr>
<th>IaaS Storage APIs</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean setUpConnection (String accessKey, String secretKey, String securityGroup)</td>
<td>Sets up a connection to the MSaaS S3 repository.</td>
</tr>
<tr>
<td>Boolean createResource (String resourceType, String resourceName, File file)</td>
<td>Creates a new resource to the MSaaS S3 repository.</td>
</tr>
<tr>
<td>Boolean updateResource (String resourceType, String resourceName, File file)</td>
<td>Updates an existing resource in the MSaaS S3 repository.</td>
</tr>
<tr>
<td>Boolean deleteResource (String resourceType, String resourceName)</td>
<td>Deletes an existing resource repository.</td>
</tr>
<tr>
<td>Boolean downloadResource (String resourceType, String resourceName, File outputFile)</td>
<td>Downloads a resource from S3.</td>
</tr>
</tbody>
</table>

The class AWSS3Storage.java provides IaaS Storage APIs (see Table 2). It is able to set up configuration (setUpConnection), and create/update/delete/download (createResource/deleteResource/deleteResource/downloadResource) varied resources in S3. For example, when a new model /evacuation.ma needs to be saved, we use createResource (String resourceType, String resourceName, File file), which will create a new resource with required Cell-DEVS (resourceType), Evacuation (resourceName), and evacuation.ma (file). After that, this model is saved in https://s3.amazonaws.com/MSaaS/Models/Cell-DEVS/Evacuation/evacuation.ma.

Please note that the MS-Image is conceptually similar to the DEVS VM kernel proposed by [Mit13]. A DEVS VM kernel is a DEVS kernel embedded in a VM. It can
be executed in local, distributed and real-time environments. The MS-Image and DEVS VM kernel are similar: both of them provide customized VM for simulators. However, there are many differences between them, listed in Table 3.

### Table 3. DEVS VM kernel vs. MS-Image

<table>
<thead>
<tr>
<th>Aspect</th>
<th>DEVS VM kernel</th>
<th>MS-Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content</strong></td>
<td>DEVS simulator environment</td>
<td>Environment for DEVS simulators, supported functions, and Cloud middleware</td>
</tr>
<tr>
<td><strong>On-promise deployment</strong></td>
<td>Yes</td>
<td>Potential</td>
</tr>
<tr>
<td><strong>Implemented with Cloud Provider</strong></td>
<td>No</td>
<td>Yes (AWS)</td>
</tr>
<tr>
<td><strong>Simplicity of deployment on Cloud</strong></td>
<td>Unknown</td>
<td>Single API call by MSaaS middleware</td>
</tr>
<tr>
<td><strong>Overall controlled</strong></td>
<td>Unknown</td>
<td>MSaaS middleware</td>
</tr>
</tbody>
</table>

The MS-Image differs from DEVS VM kernel in the following aspects:

1) The DEVS VM kernel aims to provide a general DEVS simulation environment while the MS-Image does not only focus on simulators, it also supports functions and middleware in the Cloud.

2) The DEVS VM kernel is deployed in local, distributed or real-time environments while the MS-Image is designed for the Cloud.

3) The DEVS VM kernel provides DEVS simulation protocol and it is originally designed for Web-based Simulation. As mentioned in [Tol14], it has the potential to be used in a Cloud environment; however, at this time there is no implementation of DEVS VM in the Cloud. In contrast, MS-image is designed for Cloud deployment and it is already implemented using AWS.

4) Since there is no actual implementation, it is unknown how easy to deploy DEVS VM kernel in the Cloud. For MS-Image, it is simple. MS-Image is highly available in AWS S3, and we can use single API to launch Compute Units. MS-Image is able to automatically deploy environments, install packages and dependencies, set up workspace, and start middleware service.
5) The MS-Image is fully controlled by the MSaaS middleware. Each launched Compute Unit has a running MSaaS middleware. This middleware acts as a VM manager, which can manage and control all resources. This kind of middleware is not supported by DEVS VM kernel.

In MAMSaaS architecture, the Cloud Layer focuses on providing IaaS APIs and deploying MSaaS in the Cloud. In the next section, we will discuss the details of the MSaaS middleware and its implementation.

4.2 Implementing an MSaaS: the CloudRISE Middleware

As discussed earlier, MSaaS middleware is for developing M&S resources as MSaaS services. It solves the issues of WS framework constrains and resource variety. In the current implementation, we implemented the concept of MSaaS middleware in CloudRISE.

CloudRISE is an extended version of RISE [Alz13]. Originally, RISE exposes simulation functions as a RESTful WS. The main objective of RISE is to support interoperability of distributed simulations. However, RISE has some restrictions. It was not designed to support varied M&S resources in the Cloud. RISE supports an XML configuration file for each simulation; however, this XML file is specific to DEVS models, it does not provide model context (e.g. initial state, termination conditions, inputs/output variables, objectives, assumptions, and constraints). CloudRISE reuses RISE functions and deals with the above issue. It uses Cloud services to share resources and reproduce experiments. It uses Experimental Frames to cover the model context.

CloudRISE is resource-oriented and based on RESTful WS. M&S resources are identified through URIs, and they exchange information through standard XML messages via HTTP methods (GET/PUT/POST/DELETE). CloudRISE works as a repository interface for users to share resources. More importantly, each Compute Unit (e.g., EC2 instances) has an associated CloudRISE. It lists all the experiments and their environments. For instance, an URL starts with the endpoint like http://www.amazon-ec2ip.com:8080/msaas/..., which indicates the public IP and port of an EC2 instance on which a CloudRISE is running.
Figure 12. The CloudRISE URL hierarchy

Figure 12 shows how resources are organized. There are six main branches: models, data, semantics, instances, simulations, and functions. Each branch has its own hierarchy: models manages all the models; data manages all the data; semantics manages tag-tree ontology to help selecting resources; instances manages all available Compute Unites; simulations provides environments of simulations, and functions provides environments of functions. Each branch applies URL templates. A URL template [Gre12] uses variables placed between braces ‘{}’. For instance, approach and model in the template .../models/{approach}/{model} can be substituted with any string (i.e., .../models/devs/queue).

Let us see more on branch models. It lists modeling approaches. .../models/{approach} specifies a particular modeling approach (e.g. DEVS, Modelica, Cell-DEVS). .../models/{approach}/{model} specifies any model that belongs to a given approach. Users can use HTTP methods on URL .../models/{approach}/{model}. For instance, PUT to create an XML description; DELETE to remove a model; GET to retrieve it from the Cloud; or POST .../files to append model files. Similarly, for other supported data (e.g. system behavior files, scenario data and documentation), users can use /data. They can specify the data type in ...data/{type} and the data file in .../data/{type}/{data}.

In order to describe resources better, CloudRISE supports description files in XML. For instance, it supports Model Description (an XML description for models). Its
structure is shown in Figure 13. It keeps basic information in <Model>, which includes model name, type (e.g. atomic or coupled), a description (to describe this model), and its location (e.g. the S3 URL). If the model contains input/output ports, the XML also keeps these <Ports> (<Inputs> or <Outputs>), including their name, type, and a description. In addition, it can store associated files in <Files>, including file name, type and location (the URL in S3). A model can have multiple files, e.g. CPP classes, headers, and model configuration files.

```xml
<ModelDescription>
  <Model name="" type="" description="" location=""/>
  <Inputs>
    <Port name="" type="" description=""/>
    ...
  </Inputs>
  <Outputs>
    <Port name="" type="" description=""/>
    ...
  </Outputs>
  <Files>
    <File name="" type="" location=""/>
    ...
  </Files>
</ModelDescription>
```

**Figure 13. Structure of the Model Description file**

The branch.../msaas/instances maintains MS-Images and instances. ../instances is used to list available MS-Images; ../instances/{image} is used to show instances that are launched from a specific image; and ../instances/{image}/{instance} is used to operate a specific instance. Users can PUT to launch a new instance with an XML description file (Instance Description); GET to get the XML file; and DELETE to release an existing instance. The structure of the Instance Description file is shown in Figure 14. The file includes the instance name, ID (the EC2 instance ID in the AWS), Image (where it is launched from); InstanceType (computing power, different CPU, memory, storage,
networking); \textit{IP} (the IP address of the image); and \textit{Status} (e.g. running, stopped, terminated, initializing).

\begin{verbatim}
<Instance>
  <Name>MSaaS Instance 1</Name>
  <ID>i-b663c29c</ID>
  <Image>MS-Image 1</Image>
  <InstanceType>t1.micro</InstanceType>
  <IP>54.145.233.104</IP>
  <Status>running</Status>
</Instance>
\end{verbatim}

\textbf{Figure 14. An example of Instance Description file}

CloudRISE can simplify the configuration of experiments. It is done by keeping an \textit{EF Template} (i.e. \textit{EF Template for Simulation} for models, \textit{EF Template for Function} for supported functions). Each EF Template maintains a URL structure. Users can use it to reproduce experiments. The related EF information is specified in a XML file.

\begin{verbatim}
.../msaas/simulations

/{framework}

/{experiment}
  /execution
  /results
  /debug

/web interface to list all the models that have experiments
/web interface to specify parameter values of an experiment
/web interface to reproduce the experiment
/web interface to retrieve experiment results
/web interface to retrieve debug results
\end{verbatim}

\textbf{Figure 15. URL hierarchy of the EF Template for Simulation}

The \texttt{.../msaas/simulations} uses the EF Template for Simulation (Figure 15). This template is hierarchical. It contains the EF information and corresponding experiments. The \texttt{.../msaas/simulations} is used to list all the models that have experiments. The
.../simulations/{framework} provides the EF Description. The
.../{framework}/{experiment} specifies parameter values for an experiment. The
.../{experiment}/execution reproduces an experiment. The .../{experiment}/results is used
to retrieve experiment outputs, and .../{experiment}/debug is for debug information.

The EF Description for Simulation is presented in Figure 16. It is an XML file to
represent the context for observing or experimenting with a model. It contains the
following information:

1) The framework <Name>;
2) The <Model> to be simulated (its URL is in CloudRISE);
3) The <Context> of the model, including a <Description>, its <Objective>, any
   <Assumption> made by the designer, and the <Constrains> under which the model was
designed.
4) The <Input> and <Output> parameters for the experiment. Each parameter
   includes a parameter name, type and a short description.
5) The <TerminateTime>,
6) The simulation <Environment>: an instance to specify the Compute Unit, a
   simulator (e.g. CD++), the IP address and Port CloudRISE running on (e.g.
   http://www.amazon-ec2ip.com:8080), and a group to which that Compute Unit belongs
to.

<Framework>
  <Name>...</Name>
  <Model>...</Model>
  <Context>
    <Description>...</Description>
    <Objective>...</Objective>
    <Assumption>...</Assumption>
    <Constrain>...</Constrain>
  </Context>
  <Inputs>
    <Parameter name="" type="" description=""/>
    ...
  </Inputs>
Figure 16. Structure of the EF Description for Simulation

A group can have different instances. If a group has only one instance, then during the run-time, this instance runs complete experiments. If a group has more than one instance, then these instances run a distributed simulation for the model. In this case, the model is partitioned into different sub-models, and each instance executes one sub-model. CloudRISE reuses the simulation protocols in RISE for this kind of distributed simulation. RISE can transmit messages and synchronizes the simulation time among instances [Alz13].

Each EF can have experiments with different parameter values. The ...

...simulation/[framework]/[experiment] is used for specifying actual parameter values of an experiment. It uses the Experiment Description for Simulation. Users can use PUT to create this URL; POST to append files; GET to retrieve this description file. This description contains the following information (see Figure 17): the experiment <Name>, a short <Description>, the corresponding EF Description for Simulation in <EF>, the group number of the environment, input/output parameter values in <Parameter>, and number of runs in <NumOfRuns>. Each instance runs experiments in threads (with the number of <NumOfRuns>).

<Experiment>
  <Name>...</Name>
  <Description>...</Description>
  <EF>...</EF>
<Environment group=""/>
<NumOfRuns>…</NumOfRuns>
<Inputs>
   <Parameter name="" value=""/>
   ...
</Inputs>
<Outputs>
   <Parameter name="" value=""/>
   ...
</Outputs>
</Experiment>

Figure 17. Structure of the Experiment Description for Simulation

![Diagram showing the lifecycle of an experiment in CloudRISE]

Figure 18. Lifecycle of the experiment in CloudRISE

The EF Template for Simulation allows users to control the lifecycle of experiments (Figure 18). Users can provide the EF Description for Simulation to ../msaas/simulations/{framework}, and specify its actual parameters value in ../simulations/{framework}/{experiment}. Initially, the state of this experiment is Init. Users can check the state anytime by using GET ../{experiment}/execution. CloudRISE will copy all the existing files specified to the workspace of the experiment. Users can also POST inputs for the experiment. After all of the required inputs are prepared, the state is Ready. Then, users can PUT to ../{experiment}/execution, and simulation will start Running. During the execution, users can DELETE ../{framework}/execution to abort the simulation. After the experiment completes successfully, it turns to Finished.
and the user can retrieve output from \( {.../\{ \text{experiment} \}/\text{results} } \). If an error happens, it changes to state \textit{Error}.

Using these MSaaS services, users can reproduce experiments easily. If the model and data already exist, and the simulators are already configured, users only need to provide the description files. Then, they can reproduce the experiment and control its lifecycle.

The experiments with the supported functions are managed by \texttt{/functions}. The EF Template for Function is similar to the EF Template for Simulation (shown in Figure 15). For example, \texttt{.../msaas/functions} is used to list all functions that have experiments. The URL \texttt{.../msaas/functions/\{framework\}} provides the EF Description for a function. The URL \texttt{.../function/\{framework\}/\{experiment\}} is used to provide the Experiment Description for Function, which specifies actual parameter values.

The lifecycle of supported function is also similar to the ones for simulations (presented in Figure 13). There are two differences. The first one is that users can POST their function programs at run-time while in simulations we need predefined simulators in the MS-image. Another difference is in the EF Description for Function, the “\textit{group}” in \texttt{<Environment>} has only one Compute Unit, which means that each function is run completely in one instance. In addition, the \texttt{<environment>} contains a new \texttt{<Command> } to specify about how to execute this function. For example, in a results parser function, the \texttt{<Command> } is like \texttt{.../{Function} {SimulationResults} {ParsedSimulationResults} }. It means that during the runtime, it will do the following:

1) Replace the parameter values specified in the description file;
2) Call the command line to execute the experiments.

As discussed in Chapter 2, there are many efforts in WBS and CBS, but at this time, CloudRISE is the first one to deploy both DEVS simulations and other M&S resources as Cloud services. Table 4 lists the differences of CloudRISE and other WBS/CBS approaches (e.g. DEVS/SOA and RISE). In detail,

1) DEVS/SOA and RISE were originally designed for WBS, providing a DEVS-based simulation environment through web services while CloudRISE is designed for CBS, deploying different M&S resources as MSaaS in the Cloud.
### Table 4. Comparison among DEVS/SOA, RISE and CloudRISE

<table>
<thead>
<tr>
<th></th>
<th>DEVS/SOA</th>
<th>RISE</th>
<th>CloudRISE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>WBS</td>
<td>WBS</td>
<td>CBS</td>
</tr>
<tr>
<td><strong>Supported Simulator</strong></td>
<td>DEVS-based</td>
<td>DEVS-based, open to non-DEVS</td>
<td>DEVS-based, open to non-DEVS</td>
</tr>
<tr>
<td><strong>Web framework</strong></td>
<td>SOAP-based</td>
<td>RESTful WS</td>
<td>RESTful WS</td>
</tr>
<tr>
<td><strong>VM</strong></td>
<td>DEVS VM kernel</td>
<td>No</td>
<td>MS-Image</td>
</tr>
<tr>
<td><strong>M&amp;S resources</strong></td>
<td>DEVS distributed simulation</td>
<td>DEVS distributed simulation</td>
<td>Varied M&amp;S resources (MSaaS)</td>
</tr>
<tr>
<td><strong>Experimental-oriented framework</strong></td>
<td>Partial</td>
<td>Partial</td>
<td>Full</td>
</tr>
</tbody>
</table>

2) All of them provide DEVS simulation, but RISE and CloudRISE also have the potential to support non-DEVS simulators. As discussed in [Wan15a], in order to add a new simulator, we can upload this simulator to an MS-Image, and then add this image to branch .../instances/image).

3) DEVS/SOA provides simulation services as SOA-based WS while RISE and CloudRISE provide services as RESTful WS. As discussed before, compared to RESTful WS, SOAP WS have the issues of structure constraints.

4) As shown in Table 3, DEVS/SOA uses a DEVS VM kernel to provide a general DEVS simulation environment while CloudRISE uses MS-Images. In contrast, RISE does not have such a VM feature.

5) CloudRISE also allows users to deploy varied M&S resources as services, such as models, data, functions, images, instances, and experiment frameworks.

6) DEVS/SOA and RISE can use XML to configure experiments, while CloudRISE supports full control of experimental frameworks, such as model context, EF template and lifecycle control of experiments.

#### 4.3 Case Study: Crowd M&S using CloudRISE in the Cloud

In this section, we are going to present a complete case study of Crowd Modeling and Simulation (M&S). We will show how to use CloudRISE to develop services. We
will also show how to configure experiments involved in the Crowd M&S. The case study uses a crowd model in a shopping mall with multiple floors [Wan12a] [Wan13a]. In this model, the crowd tends to move forward, but they can change direction if the way is blocked [Hoo04].

Crowd M&S has been used to analyze the crowd behavior. It can be used to predict the impact of pedestrian movement [Hoo04]. Designers can use Crowd M&S to test their designs of buildings and find flaws before construction has begun. To study the behavior of a crowd in a building, we need basic information of the building (for instance, the floor plan of a building could be used as an input for a crowd simulation). At present, new Building Information Modeling (BIM) tools are widely used to manage the building data and improve the quality of building designs [Ham08]. The Industry Foundation Classes (IFC) is a standard used to represent BIM files. We can extract information from IFC files, and then use them as inputs for crowd models. In addition, BIM tools can also be used for 3D visualization, and in our case, for visualizing crowd simulation results [Fre13]. By doing this, the designers can efficiently view, analyze and refine the design with different scenarios. To do this, different kinds of M&S resources are involved (Figure 19).

![Diagram of Crowd M&S Process]

**Figure 19. The resources involved in the Crowd M&S**

In general the Crowd M&S process involves the following resources:

1) The *CrowdModel* that describes the crowd behavior. In our case, we will use a Cell-DEVS crowd model.

2) The supported data of the *IFC* file that contains the building information.
3) The supported data of Domain Specific Models (DSM) file that contains the information for crowd simulation.

4) The supported function of a BIM Data Collection. It can extract data from the IFC designs and DSM files. The extracted data can be used as the model input.

5) The experiments of BIM Data Collection. They can generate two outputs: the InitialState file that contains the layout of the building; and the ModelVariables that contains values for the crowd model;

6) The Simulation (in our case, a Cell-DEVS simulation). We can execute crowd simulations with the inputs of InitialState and ModelVariables.

7) The experiments of the crowd Simulation. They generate the SimulationResults.

8) The supported function called ResultsParser, which can parse the results so they can be visualized in 3D Visualization Tool.

In our case, all these resources are stored in Storage Units in the CloudRISE.

Next, we will briefly discuss how to use CloudRISE to execute the steps above. In [Wan12a] [Wan13a], we developed a crowd model in Cell-DEVS to study the crowd behavior in a multi-floor building. This model has walls, furniture, exits (on the first floor) and stairs (connecting floors). In the case of emergencies, people try to evacuate each floor, move down through stairs, and finally leave the building through the exits. The Cell-DEVS model was implemented in CD++ (with file name crowdModel.ma) [Wai09]. The behavior of each cell depends on its current state, neighbourhood and a set of rules. The rules can be categorized as follows: a) Someone enters a cell; b) Someone leaves a cell; c) Someone moves from a cell to a stairwell; d) Someone enters a stairwell; and e) Someone exits a stairwell.

We activated the API of ../models to share the crowd model. Users can POST the model to ../models/cell-devs/crowd/files. In addition, they can provide an XML description file for this model via PUT to ../models/cell-devs/crowd. This description is shown in Figure 20.

```
<ModelDescription>
  <Model name="CrowdModel" type="cell-devs" description="a cell-devs model to simulate crowd behavior in a multi-floor building" location="s3/MSaaS/Models/Cell-DEVS/crowd"/>
  <Files>
```
As we can see, it contains the model name, its type and description, and its location in the Storage Unit. The description file shown here keeps general information, which can be customized if specific information is available.

For other resources (e.g. the IFC file, the supported data of DSM, etc.), users can share them in CloudRISE in a similar way. For instance, we can use the IFC files to manage the layout of the area under study. Figure 21 shows the IFC file using Autodesk Revit (a BIM tool that can manage IFC files). Similarly, we can share this IFC file using .../data/ifc/buildingMall/files.

As discussed in the previous section, CloudRISE can simplify the experiment configuration by keeping the EF Templates of experiments.

Next, we show the EF Templates for crowd simulations (i.e. the Cell-DEVS Simulation component in Figure 19). CloudRISE uses the URL...
.../msaas/simulations/crowdEF to maintain an EF Template for crowd simulations. This template allows users to reproduce experiments. The URL .../msaas/simulations/crowdEF provides the EF Description file shown in Figure 23. Users can PUT and GET the EF Description via the URL of .../msaas/simulations/crowdEF.

<Framework>
  <Name>EF of the crowd model</Name>
  <Model>...msaas/model/cell-devs/crowd</Model>
  <Context>
    <Description>a model to simulate crowd behavior during evacuation</Description>
    <Objective>to study crowd behavior during evacuation</Objective>
    <Assumption>the crowds move at a constant speed</Assumption>
    <Constrain>cell sides should less than 500x500x10</Constrain>
  </Context>
  <Inputs>
    <Parameter name="InitialState" type="VAL" file="" description="the initial state of cells for the crowd model"/>
    <Parameter name="ModelVariables" type="MACRO" file="" description="dimension variables for the crowd model"/>
  </Inputs>
  <Outputs>
    <Parameter name="SimulationResults" type="LOG" file="" description="log of the crowd simulation results"/>
  </Outputs>
  <TerminateTime>30 mins</TerminateTime>
  <Environments>
    <Environment group="1" simulator="CD++" instance="MSaaS Instance 1" IP="54.167.165.82" port="8080"/>
    <Environment group="2" simulator="CD++3.0" instance="MSaaS Instance 2" IP="54.167.144.86" port="8080"/>
  </Environments>
</Framework>

Figure 22. EF Description for the Crowd Simulations

After the EF is set up, we can reproduce experiments. In particular, .../crowdEF/evacuation is used to specify actual parameter values. The URL .../evacuation/execution is used to reproduce the experiment, .../evacuation/results is
used to retrieve outputs, and .../evacuation/debug is to get debug information. The Experiment Description for Simulation is shown in Figure 24.

<Experiment>
  <Name>Evacuation experiment of a crowd model</Name>
  <Description>an evacuation experiment of Crowd M&S in a three floors building</Description>
  <EF>.../simulation/crowd ev acuation/</EF>
  <Environment group="1"/>
  <NumofRuns>5</NumofRuns>
  <Inputs>
    <Parameter name="InitialState" value="...s3/experiments/.../InitialState1.val"/>
    <Parameter name="ModelVariables" value="...s3/experiments/.../modelVariables.macro"/>
  </Inputs>
  <Outputs>
    <Parameter name="SimulationResults" value="...s3/experiments/.../SimulationResults"/>
  </Outputs>
</Experiment>

**Figure 23. Experiment Description for the Crowd Simulations**

The description contains the experiment <Name>, a short <Description>, the corresponding EF Template URL in <EF>, the group to run this experiment (i.e. it uses CD++ in MSaaS Instance 1), input/output parameter values in <Parameter>, and number of runs in <NumOfRuns>. Both inputs and outputs are AWS S3 URIs (during run-time, the output URL will save the experiment results).

CloudRISE allows users to control the lifecycle of experiments. As discussed in the previous section, uses can start by using .../msaas/instances. The users can PUT to .../instances/MS-Image1/{instance} to launch new instances, and DELETE to release an existing instance. After the instances are ready, the users can use the EF Template mentioned above to reproduce experiments. Users can also POST inputs during run-time. Then, the experiments are *Ready*. Users can PUT .../crowdEF/evacuation/execution to
start experiments. After the experiment completes, user can retrieve output from 

.../crowdEF/evacuation/output.

We conducted some tests to study the simplicity of CloudRISE (in terms of its environment configuration). The idea was to show how users could execute simulations easily using different instances and simulators. Let us assume that users need to run a number of experiments for crowd behavior analysis using the model mentioned. Users do not know which instance type and simulator they should use (they want to find a right combination to save money). Using CloudRISE, the users can easily run the experiments and find a right combination to use.

![Figure 24. Performance test of reproducing simulation experiments using CloudRISE](image)

Figure 24 shows the average execution time of different experiments launched by CloudRISE. In this test, the same crowd model was run in three types of instances and each instance used two different simulators (i.e., there are six different combinations of instance type and simulator). Each combination run five experiments and the average execution time of these experiments is presented in the figure. Users launched three types of instances from an MS-image: Micro Instances (low-cost with small amount of CPU resources), General Purpose Instances (a balance of compute, memory and network resources), and Compute Optimized Instances (higher CPU cores and memory, but more expensive). Each instance set up the CloudRISE middleware with two different simulators (CD++ and CD++3.0).
As we can see from Figure 24, different combinations of instance type and simulator result in different average execution time. For instance, for each instance type, the CD++3.0 always takes less time than CD++ (CD++ 3.0 has a faster way to collect simulation results and it does not use message queues as CD++ does). In addition, for the experiments on either CD++ or CD++ 3.0, Compute Optimized Instances always have shortest execution time (i.e. 467s in CD++ and 124 in CD++3.0). These numbers are about 1/3 of the time required by General Purpose Instances and 1/5 of the time required by Micro Instances. It implies that if users highly care about the execution time and they can afford some extra cost, they may choose Compute Optimized Instances; otherwise, if they want to save money and the performance of CD++ 3.0 in Micro Instances is acceptable, they may choose Micro instances.

Besides the simulations, the supported functions (in our case, BIM Data Collection and Results Parser) are managed by functions in CloudRISE. The BIM Data Collection can extract information from IFCBuildingFile and DSMInstance, and generate InitialState and ModelVariables as the model inputs. The Results Parser can parse SimulationResults to ParsedSimulationResults (i.e. the format that can be used for 3D visualization). As mentioned in previous sections, the EF Templates for these functions are similar to the EF Template for the crowd simulation. One difference is that use can provide a <Command> in the <Environment> of the EF Description. This Command can specify about how the function executes in the environment. For example, the command of BIM Data Collection is java –jar {Function} -initLayout {InitialState} -initParameters {ModelVariables} -IFCFile {IFCBuildingFile} -DSMInstance {DSMInstance} while the command of Results Parser is ./{Function} {SimulationResults} {ParsedSimulationResults}. At run-time, when the experiment starts, CloudRISE will replace the variables with the actual values specified in the Experiment Description, and then execute the command line.

Finally, the parsed simulation results can be visualized in a 3D visualization tool. In [Fre13], we developed a visualization tool in BIM (i.e. Autodesk 3ds Max). This tool can load the IFC file of the building and the parsed simulation results, and animate the crowd movements. Figure 25 shows how results look like: e.g., a) crowd movements in different floors; and b) crowd movements in a stairwell. The tool also allows the designer to filter
floors and focus on individual movement. The demo of this case study can be found at http://www.YouTube.com/watch?v=u5idq-PDLck.

Figure 25. 3D visualization of parsed results of crowd simulation

Now, a complete application using the CloudRISE is available for studying the behavior of the crowds. CloudRISE can help the users to deploy resources involved in the Crowd M&S as MSaaS (e.g. the crowd model and the IFC file, the crowd simulation, the data collection and the results parsing). More importantly, CloudRISE provide a scalable way to launch instances and configure environments. Users can reuse these services to test different designs of buildings and determine which one is best. For instance, they can provide different IFC files, configure instances, reproduce experiments, and observe the results in 3D visualization tool.

4.4 Summary

In the MAMSaaS architecture, Cloud Layer and MSaaS middleware are for developing and deploying varied M&S resources as services in the Cloud. This chapter described the details of the Cloud Layer (which focuses on environment configuration and deployment) and the MSaaS middleware (which deals with the structure constrains caused by the web service framework and the variety of resources). We also presented the design and implementation details of CloudRISE. CloudRISE extends from RISE and implements the concept of the MSaaS middleware. The chapter concludes with a complete case study of Crowd M&S using the MAMSaaS and its CloudRISE.
Chapter 5: M&S Mashup

The previous chapter has shown how we can develop and deploy MSaaS by simply using the Cloud layer of MAMSaaS. In order to simplify the development of M&S applications further, we need to integrate existing resources and services. To do so, we propose a new method (M&S mashup) in the MAMSaaS architecture. In this chapter, we will discuss its details.

There are two motivations for developing M&S mashups. At first, web services have been used in M&S for many years. In addition, with the development of Web 2.0, there are also numerous Web APIs available. Nevertheless, there is no research showing how to integrate these available services in a simple way. Second, even though there are different development processes for M&S application, the common issue for them is that as more and more resources are involved, the development process become more and more complex. As a result, developing an M&S application is costly and time-consuming. Users should be able to link and execute existing services easily, as well as visualize the results at run-time. Therefore, a simplified M&S application development process that can integrate M&S resources and services is highly needed.

In order to do so, we present a new method in MAMSaaS based on mashup technologies in the Web 2.0 [Bal08]. Mashups have been widely used in different domains; however, these technologies have never been used in M&S applications. Our objective is thus to develop M&S mashup applications to integrate different resources. In general, M&S mashup is a layered method. It can deploy and identify M&S mashup components as well as link and execute mashups for fast M&S application development.

M&S Mashup allows developing M&S mashup applications by integrating different M&S related WSs and resources. It supports universal identification and development for mashup component (named Boxes). A Box consists of varied M&S resources (i.e. MSaaSs, WebAPIs, widgets and operators). In addition, M&S Mashup also supports component development (i.e. boxes), component linking (i.e. wiring boxes), and execution and visualization of the mashup at run-time. In this chapter, we will discuss the M&S Mashup methodology and the tools that we implemented (i.e. the Box Development Tool and the MAMSaaS Mashup Platform). In addition, we will also
present an M&S Mashup prototype, which integrates a fire spreading model, a distributed M&S service, and a visualization tool.

### 5.1 M&S Mashup Methodology

The M&S Mashup methodology can simplify the process of developing M&S applications. It can be viewed as a sub-set of MAMSaaS (Figure 26). It has three layers to create and run M&S mashups, as follows:

- **Box**: it is responsible for identifying mashup components (termed *Boxes*). Boxes can have different categories (e.g. MSaaS from the Cloud Layer, existing Web APIs, widgets, and operators). Each box is identified by a uniform box signature. It has its own function for handling input messages. In addition, it has its own visual form.

- **Wiring**: it is responsible for connecting boxes into a mashup. Boxes can be linked with each other by their inputs/outputs. These inputs/outputs are identified in their box signatures. A same box can be reused and re-wired in different mashups for new purposes.

![Figure 26. M&S Mashup Methodology](image-url)
• **Mashup Application**: it is responsible for users to wire boxes and run mashup applications. They can add box, wires and visualize the mashup results. It supports an event-based data flow to show the mashup during run-time.

M&S Mashup supports a lightweight life cycle for developing M&S applications. In [Dan11], the authors have compared the traditional development process with the mashup process, as shown in Figure 4. They concluded that mashup has a lighter development process. A standard mashup development process includes sub-processes like Discovery and selection, Mashup composition, Usage and maintenance. Mashup is simple and allows continuous improvement: users mash up exiting resources and check whether the mashup works. If they do not like the results, they fix the problems and run the mashup again, until it is satisfactory.

Similarly, M&S Mashup belongs to this mashup development process. It can be viewed as a special form that applies to the M&S domain. As shown in Figure 27, M&S mashup has the following mappings compared to the standard Mashup life cycle:

![Diagram of M&S mashup process and the standard mashup life cycle](image)

**Figure 27. Mappings of M&S mashup process and the standard mashup life cycle**

1) **Box development and identification** supports the Discovery and selection process. A box is the fundamental element of M&S mashup. A box can be discovered and selected by its universal structure (i.e. the Box Signature), which identifies its
inputs, outputs and function. In addition, a box can be easily developed because it supports a standard package of XML, JS and HTML/CSS.

2) **Box wiring** supports the *Mashup composition* process.

The box wiring mechanism supports mashup composition. Boxes can be linked with each other by their inputs/outputs. After wiring, a mashup is developed and it can be used in the next mashup usage and maintenance process.

3) **Mashup execution and workspace management** supports the *Usage and maintenance*.

The mashup platform supports the mashup workspace management. Users can run the mashup in their workspace and visualize its results (e.g. simulation results). Users can easily modify the mashup by updating and re-wiring boxes, and then they can run the mashup again.

Here we will discuss the design details of these layers, and show how to develop boxes and build M&S mashups. Note that M&S mashup is built on top of the CloudRISE middleware provided by the Cloud Layer (see Chapter 4), which implements the concept of MSaaS (exposing all kinds of M&S resources as services).

### 5.1.1 Box – developing Mashup components

**Boxes** represent mashup components used for M&S. They are mini web applications for M&S-related scenarios (e.g. fetching a model, reproducing an experiment, visualizing results, etc.). Boxes receive heterogeneous data from varied M&S resources, and they modularize specific functions and send events to other boxes. Boxes are basic elements of M&S mashup. They are developed using basic web technologies. Users can publish and share boxes on the Web.

There are four basic types: *MSaaS, WebAPI, Widget*, and *Operator* Boxes.

- **MSaaS Box**: it uses MSaaS services from CloudRISE. As discussed in the previous chapter, CloudRISE exposes M&S resources as MSaaS. There are six types: simulations, models, supporting functions, supporting data, semantic data and instances. The ones closely related to the M&S mashup are simulations, functions, and models. Users can manage resources and control the lifecycle of experiments using MSaaS services. To simplify experiment building, MSaaS can combine several MSaaS into one box. For example, an MSaaS simulation box can generate a configuration file, create a
new experiment using that file, start the simulation and check its status; when it finishes, it sends simulation results to the output.

- **WebAPI Box**: it calls open Web APIs. Each API follows a WS principle (RESTful/SOAP-based). ProgrammableWeb.com [Pro14] is currently the most popular API directory. WADL is a popular language to describe the syntax of REST web services. In addition, many IT companies (i.e., Google, YouTube, Flickr, etc.) provide their own HTML pages for their REST APIs. For example, Mashape (http://www.mashape.com/explore) is the Cloud API Marketplace where developers can distribute and integrate Cloud APIs. SOAP web services are usually described in WSDL files. For example, WebServiceX (http://www.webservicex.net/ws/default.aspx) has over 70 SOAP WSs based on WSDL.

- **Widget Box**: it is a lightweight web application. It is used to render data on web browsers. It has at least one input so that it can receive and represent data. Widget is mostly for visualization. It can be developed by users with basic web development knowledge (e.g. HTML/CSS/JS). What’s more, they can be reused from existing web sites or other mashup platforms. For example, http://www.100widgets.com provides different widgets, supporting varied types of data. Examples of such widgets include forms, diagrams, tables, maps, photos and videos.

- **Operator Box**: it takes input and generates the output based on a customized process. The reason why we need operators is to address inconsistencies between boxes (i.e. boxes with similar ports that cannot be connected directly). Operators can be viewed as a converter between boxes. It can be a filter, aggregator, splitter, or adapter. For example, if one port of a box is *full name*, and one port in another box is *given name*, an operator can be a splitter (e.g. extract *given name* from *full name*), so these boxes can be connected. Operators do not need to be viewed in web browsers.

Though boxes have different types, they are managed in a similar way. Each box is packaged in a separated zip file, so it can be developed, downloaded and installed on different servers. Each zip file has three parts, as follows:

- **Box Signature**: To manage and identify boxes, we provide a uniform structure,
**Box Signature (B)** = \(<B_x, I, O>\) \hspace{1cm} \text{(Definition 1)}

\(B_x = <B_n, B_t, B_d, B_s, B_a, B_p, B_m>\) is the general information of the box,

\(I = \{p\}\) is a set of input ports of the box,

\(O = \{p\}\) is a set of output ports of the box,

\(p = <p_n, p_b, p_d>\) includes port name, type, and description.

The Box Signature identifies each box with its basic information \((B_x)\), input ports \((I)\) and output ports \((O)\). The basic information includes its name \(B_n\), type \(B_t\) (e.g. MSaaS, WebAPI, Widget, Operator), description \(B_d\), subtype \(B_s\) (e.g. simulation MSaaS, RESTful WebAPI), author \(B_a\), path \(B_p\) (the URL of a related WS), and method \(B_m\) (the method name of a related WS). A box can have multiple input and output ports. We use these ports to connect boxes. Each port includes a port name, type (the message type in the port) and description (a text that describes the port). Box Signatures can be described in XML files, which can be either provided by users or constructed automatically from existing sources.

For instance, given a WebAPI WS to forecast the weather, its name is *WeatherForecast*, type is *WebAPI*, description is *return weather forecast information*, subtype is *SOAP*, author is *JohnDoe*, path is *http://www.webservicex.net/weatherforcat*, and method is *GetWeather*. This WebAPI Box has one input port (with *CityName* as its name, *xsd:string* as its type and *city name* as its description), and one output port (with *weather* as its name, *xsd:string* as its type and *weather forecast in 5 days* as its description).

- **Box Function**: Each box has a function. Boxes are event-driven. When an input event comes, they trigger a function in the Box. After that, they send data through output ports. Different types of boxes have different functions. An MSaaS Box combines multiple MSaaS services. For example, for the box of a simulation experiment, its box function will: 1) construct a new experiment.xml; 2) create a new experiment; 3) start the simulation; 4) check the result; 5) get result and send it to output. For a WebAPI Box, the function executes as defined in the Web API (e.g. SOAP WS, RESTful WS). For a
Widget Box, the function tells how to visualize the input data. For an Operator Box, the function is the action to be executed (e.g. splitting, combing, data conversion).

- **Box View**: Each box can also have a view in web browsers. The box view is what end users will see. Boxes can have HTML/CSS files for visualization purposes. For boxes of MSaaS and WebAPIs, their views could be either their signatures or execution status. For Widget Boxes, they can reuse existing HTML/CSS files. Users can also customize these files to change how the data will show. A Box view is optional. Boxes like operators are only for transforming data between boxes; it is not mandatory to view operators in a web browser.

### 5.1.2 Box Wiring – linking boxes

A key feature of the boxes is that they can be connected to each other, which is called *Box Wiring*. It is for composing different boxes through inputs and outputs.

\[
\text{Box Wiring (W)} = <\{B\}, \{B_x.I, B_y.O\}> \quad \text{(Definition 2)}
\]

\(B = \{B_x, B_y...\}\) is a set of boxes,

\(\{B_x.I, B_y.O\}\) is a set of connections between boxes

A Box Wiring is a combination of boxes. Each Wiring \((W)\) contains a set of boxes \(\{B_x, B_y...\}\) and connections \(\{B_x.I, B_y.O\}\). For instance, \(B_x.I, B_y.O\) means the output port \(O\) of Box \(B_y\) can be linked to the input port \(I\) of Box \(B_x\). By wiring boxes, users can reuse them in multiple M&S scenarios without understanding the internal details.

Box Wiring in M&S Mashup is based on an Event-driven architecture (EDA). As shown in Figure 28, boxes can interact with each other via events. An event can be anything that is “change in state”. For example, after a simulation finishes, the simulation changes from “running” to “done”. A mashup with this simulation may treat this state change as an event. M&S mashup transmits events among loosely coupled boxes. Boxes can be event emitters or event consumers. Wiring connects Boxes. A wire allows the transmission of events from emitter boxes to consumer boxes. An Emitter Box has the responsibility to generate events, it does not know the consumers of the event, it does not even know if a consumer exists, and in case it exists, it does not know how the event is
used or further processed. A Consumer Box has to react as soon as an event is presented. The reaction can be in many ways. For instance, an MSaaS box or WebAPI box will call services provided by CloudRISE or Web APIs; a widget box will provide a view for the receiving events; and an operator box will filter, transform or simply forward the event to other boxes.

Box wiring implies that any two boxes can be connected. It is a general approach to compose resources. There is no constraint about the logic flow among the boxes. The wiring among boxes supports a loosely coupled sequential composition. Operators can help to improve this composition with more complicated controls (e.g. filter, aggregator, splitter, adapter, business process management rules). In addition, boxes can contain heterogeneous data and be linked together. We can add more controls in boxes. For example, we can add a Human-in-the-Loop widget box to allow users to control the output and decide which box should be triggered next.

We use Ajax provided by JQuery to support this event-driven pattern for box wiring. Boxes are developed in HTML/JS/CSS. We implemented a mashup platform to store all boxes and define all wirings. The events that are transmitted among boxes are happened in the user’s Web Browser. JQuery is the most popular JavaScript library in use today. Ajax in JQuery allows M&S Mashup to create asynchronous event messaging. With Ajax, M&S mashup can send data among boxes asynchronously. Events are transmitted between boxes as XMLHttpRequest objects. Each box may have input event functions (for handling received events) and output event functions (for sending events to
other boxes). More details will be discussed in the implementation section (see Section 5.2.2).

5.1.3 M&S Mashup – building mashup applications

The boxes and wiring mechanism can be used to build M&S mashup applications.

\[
\text{Mashup (M)} = \langle \{B\}, \{W\}, U \rangle \quad \text{(Definition 3)}
\]

- \( B = \{B_x, B_y, \ldots \} \) is a set of boxes,
- \( W \) is a set of Box Wirings,
- \( U \) is a user workspace for this mashup.

An M&S Mashup consists of a set of Boxes (B), Box Wirings (W), and a User Workspace (U). Users can have different workspaces. In any workspace, users select boxes and wire these boxes through their input/output ports. After that, the mashup application is ready. Users can run and visualize it.

![Figure 29. Box/Wiring/Mashup example](image)

Figure 29 shows an example of an M&S Mashup. It is made up of six boxes (B1 to B6). For example, we have MSaaS Boxes B1 and B2, Web API Boxes B3 and B4, Operator Box B5, and Widget Box B6. Users can build a mashup application by wiring these boxes as shown in the right part of Figure 29. At run-time, the data generated in B1
will pass to B3 and trigger B3’s function, and then B3 will output its data to B2. Similar actions happen in other boxes. Users can see the mashup as defined in each Box View on web browsers.

5.2 M&S Mashup Implementation

We have developed different tools to support the M&S Mashup methodology. Here we discuss the Box Development Tool (used to extract and develop boxes) and the Mashup Platform (used to wire boxes and run M&S mashups).

5.2.1 Box Development Tool

A Box is the fundamental element in MAMSaaS. We developed a Box Development Tool. This tool can extract Box Signatures from existing files. It also can load and save Box Signatures in XML. In addition, it can generate the configuration file that is used in box package. This tool can also suggest to users similar existing boxes.

The Box Development Tool was developed in Java using SWT (Standard Widget Toolkit). SWT is a graphical widget toolkit for Java. It is open source and provides an efficient UI. The Box Development Tool follows the MVC design pattern.

![Figure 30. Class Diagram of Box Development Tool](image-url)
Figure 31 shows its class diagram, which consists of three groups:

- The *Data* classes manage the data in the *BoxSignatures*. *BoxSignatures* keeps all box signatures in a list. *Signature* contains one or more *Operations*. Each *operation* has one *Parameter* or *ComplexParameter*. *Parameter* has basic port information (e.g. name/type/description) and *ComplexParameter* uses complex data types (e.g. user-defined XSD).

- The *Logic* classes are used to define the logic of boxes. They can extract information from other files and construct *Signatures*. 
  *SimulationFrameworkXMLLoader* and *FunctionFrameworkXMLLoader* load EF from CloudRISE. They can extract and save EF as *Signature*. Similarly, *WADLReader* and *WSDLReader* load and parse the description file of Web APIs. 
  *SignatureXMLHandler* loads and saves signature as XML. 
  *Signature2ConfigurationXMLConverter* converts a *Signature* into a box configuration (a format used in Simulation Mashup Platform).

- The *View* classes are used to build UI. The default UI is *MainShell*. It has two menus: *BoxSignatureUI* for extracting and loading a *Signature* and *BoxDevelopmentUI* for saving a *Signature* and developing box packages.

As discussed before, each box is packaged in a Zip file. In our tool, Boxes are developed using web technologies (XML, JS, and HTML/CSS). This Zip file is extended from WireCloud [Zah14], which will be discussed in the next section. The Box Signature is a XML configuration file. The Box Function is a JS file that defines the actions for input events, while the Box View contains HTML/CSS files to show the data in web browsers. For each box, we generate its configuration as XML. For the JS and HTML/CSS files, this Box Development Tool can suggest to users similar boxes.

The JS function triggers inputs events by reusing the WireCloud’s API: 
*MashupPlatform.wiring.registerCallback(inputName, callback)*. When the function finishes, it outputs message by reusing the WireCloud’s API:
*MashupPlatform.wiring.pushEvent(outputName, data)*. The JS function can execute SOAP WS or RESTful WS. In general, executing these WSs need to build input messages, ports, and handle output messages.
1) For RESTful WS, the input message is the input event, the channel is the HTTP method (GET/PUT/POST/DELETE) on a particular URL, and the output message is the HTTP response either in XML or JSON format.

2) For SOAP WS, the input message is a SOAP message, the channel is the HTTP POST method, and the output message is sent in the HTTP response.

Figure 31 shows the GUI of Box Development Tool. It has two menus: “Box Signature” and “Box Development”. Each menu has sub-menus. “Box Signature” is for Loading and extracting the Box Signature (through XML) while “Box Development” is for generating Box template configuration (to be used in M&S mashup platform).

![Figure 31. Box Development Tool menus](image)

Figure 32 shows an UI example of the Box Development Tool. This example extracts the Box Signature from Facebook REST APIs. As shown in this figure, different APIs are extracted from Facebook’s WADL file. Each API contains the operation name, method, description, inputs and outputs. For example, an API to search public objects (e.g. user, photo, page, comment) is shown in Figure 32. This API contains method (GET-search), description (search over all public objects in the social graph), inputs (two parameters of query string and object id), and outputs (search results). Note that for the WebAPIs not supported by WADL/WSDL files, they can be easily obtained from online HTML pages. In addition, user can build a signature XML by a sub-menu “Loading – Box Signature in XML”. Some popular service repositories in HTML are https://www.mashape.com/explore and http://www.programmableweb.com/
Figure 32. Extraction Box Signature from Facebook REST WebAPIs

Figure 33 shows an example of generating a box configuration. After loading and extracting box signatures, a user can select an API from the tree-like GUI and save it to
the format to be used in M&S mashup platform. For example, in Figure 33, the user can load saved signatures, select the “geoIpToAddress” REST API, and specify a file-path. Then the configuration file of its box package will be saved. At the same time, the Box Development Tool can suggest to user similar boxes, so users can reuse them as examples.

5.2.2 M&S Mashup Platform

The M&S Mashup Platform has a wiring editor. This editor allows users to build M&S mashups rapidly. Users drag and drop appropriate boxes into a workspace, and then connect the boxes. The mashup is then ready and users can visualize this mashup at runtime.

The M&S Mashup Platform is an extended version of WireCloud [Zah14]. WireCloud is an open source mashup platform. It supports widget uploading and wiring, user workspace management, and mashup execution. The main difference between WireCloud and our mashup platform is that our platform supports boxes for M&S mashup. WireCloud is a general-purpose mashup platform, but not for M&S mashups (which lacks the support of different boxes with different handling processes). The M&S mashup platform extended WireCloud. It supports boxes for M&S mashup (i.e. MSaaS, WebAPIs, widgets, and operators).

![Class diagram of M&S Mashup Platform (powered by WireCloud)](image)

Figure 34. Class diagram of M&S Mashup Platform (powered by WireCloud)
The M&S Mashup Platform is implemented in Python using Django. Django is an open-source web application framework. The reason we chose Django is that WireCloud is developed using Django and it has many benefits. Django supports the MVC architectural pattern. Django's primary goal is to ease the creation of complex database-driven websites. Django emphasizes reusability of components and rapid development. Django also provides an optional administrative interface. This interface simplifies the back-end data maintenance. Many well-known sites use Django, including Pinterest, Instagram, and Mozilla.

The M&S Mashup Platform follows MVC design patterns (see Figure 34), with following classes.

- The Data classes extend the Category in WireCloud to BoxCategory, in which we added two new types, which are MSaaS and WebAPI. We reuse most features of widget and operator provided by WireCloud. Each box has three elements: signature, view, and function.

- The Logic classes extend the uploading and searching logic of WireCloud to support all types of Boxes. In BoxUpload, we changed the package format as zip file, and modified the uploading and parsing logic. In BoxSearch, we changed the databases and searching logic. We extended the BoxWiring mechanism and MashupExecution mechanism in WireCloud to support all types of boxes.

- The UI classes change the UI of WireCloud. The overall UI has been modified in M&S Mashup UI. For Box Management UI, we changed the box uploading and searching pages. For Workspace UI, we modified following functions: 1) Add Box button in user workspace, 2) Wire Boxes button that drags and drops boxes in the wiring editor, 3) My Boxes to select from available boxes. For Mashup UI, we reuse the mashup execution engine of WireCloud for box execution.

Here we show some UI interfaces in this mashup platform.

At first, this platform can manage boxes by uploading and searching them. Figure 35 shows the box uploading page. Users can choose the box zip file developed from the above Box Development Tool, and upload it into the M&S Mashup Platform.
Figure 35. Box uploading

Figure 36. Box selection

Figure 36 shows the box selection page. The platform will keep the boxes that user uploaded. Users can either search keywords of box by their name, or select particular type of box, and then the mashup platform will return users with matched boxes. Users can click on each box to view its details.

Figure 37. Adding box resources into user’s workspace
The M&S Mashup Platform has a wiring editor to build M&S mashups. Figure 37 shows how to add box resources into a user workspace. Users can add boxes by clicking the “+” button and customize its layout.

After the boxes are loaded, users can select boxes, then drag and drop them into the workspace.

Figure 38 shows how to wire boxes. In this example, one box is a widget that allows users to provide a GIS TIFF URL (containing GIS dataset in a particular area); another box collects information from a GIS TIFF file and generates initial values for simulation model. After wiring the output port “keyword” of GIS TIFF box and the input port “GISTIFF” of GIS data collection, the GIS TIFF data will transmit between them.

![Figure 38. Box wiring](image)

Next, the mashup is ready and users can visualize it at run-time. Figure 39 shows how the mashup looks like for the above example. User can see the workspace page, and provide input value of “GIS-TIFF URL”. After pressing the “send” button, the GIS data collection box will receive the input. In that box, data is collected, and corresponding results are shown (e.g., data has been successfully collected).

![Figure 39. Mashup execution](image)
Note that in order to let users other than the mashup owner (e.g. project manager, course professor) to easily view and run the mashup, the mashup owner can also make the mashup public. This is done by reusing a feature of WireCloud which can expose a mashup as a public URL. Thus, other users can view and run it by accessing the URL. However, only the mashup owner has the permission to modify the mashup (e.g. adding boxes, rewiring boxes, deleting workspaces).

The M&S mashup platform is deployed in the Cloud. The current implementation is in AWS. We also saved all the configuration of this platform in an AWS image, so we can recover it easily when the platform is down. Like CloudRISE, we can choose different instance type (e.g. free, high memory, high CPU) for the mashup platform.

Currently, students in our ARSlab start using this platform for their M&S application projects. This mashup platform has also been used in the M&S graduate course of Dr. Gabriel Wainer. For example, one requirement of student’s term project is to provide box signatures for the project, which can be used to keep a box/mashup library; therefore, students can easily find boxes and collaborate to develop mashups. In Chapter 7, we will discuss different mashup case studies. We will see that the mashup platform usually takes students less than one month to develop a mashup, which takes much less time than using traditional ways.

5.3 Simple Case Study: Fire Spreading M&S Mashup

In this section, we show a simple M&S mashup example. This example uses the mashup tools to build a mashup from scratch. The case study focuses on building a Fire Spreading M&S mashup.

The spread of fire is a complex phenomenon, which depends on many variables, including the type of fuel, the geography of the area, and the weather. M&S technologies have been used to study the fire spreading for many years. M&S is able to represent how fire spreads, and now M&S is generally the preferred solution for predicting wildfires [Wai09]. For example, in [Wai09], the author has present different Cell-DEVS models to study fire spreading. One of these models is based on the well-known Rothermel’s Rules, which considers environmental and vegetation conditions, computing the ratio of spread
and intensity of fire. However, in order to develop an application for the study of fire spreading, only having models is not enough, it also involves various M&S resources. The M&S mashup method is used to simplify the development process of such an application. Our mashup application includes:

- **Fire spreading modeling**: a Cell-DEVS fire spreading model.
- **Cloud-based Simulation**: to execute simulation experiments for the model.
- **Results analysis**: a function to analyze simulation results.
- **Visualize results**: a widget to visualize simulation results.
- **WebAPIs**: existing web APIs that are useful for this study, e.g., a web API to search latest fire spreading news.

The Box Development Tool can help users develop boxes. The developed boxes are as follows:

- **MSaaS Box** uses CloudRISE. It contains a *Cell-Devs model simulation* Box (an experiment for the Cell-DEVS fire spreading model). Its signature is extracted from the corresponding configuration; its function combines multiple MSaaS in a same experiment framework; and its view shows its execution status.

- **WebAPI Box** calls the existing Web API. It contains a *Web_search* Box (a REST WS to search from web pages like Google, Yahoo and Bing). Its signature is extracted from the WS description file or HTML files; the function calls the Web API; and the view shows its output message.

- **Widget Box** shows input data in web browsers. Here we integrate the *simulation log viewer* developed by Omar Hesham, a PhD student of Professor Wainer. Originally, this log viewer was a web-based tool to show Cell-DEVS simulation results. We reuse and reconstruct this log viewer as a box. We changed this widget so it can receive inputs from URIs of model, cell color and log.

- **Operator Box** handles inconsistencies between boxes. It contains *Get log from zip* (extracting Log files from a zip file). Its signature is generated by the Box development tool; the function receives a zip file and outputs its log file as an URL. There is no view for this operator.
All boxes have similar structure: box signature, box function and box view. In Chapter 4, we have shown a similar example using EF to simplify simulation experiment. Here we will show an example for another type of Box (i.e. web search WebAPI Box). This box uses an open API provided by Faroo (http://www.faroo.com/hp/api/api.html), which can search news and articles from more than 2 billion pages. This API is integrated from Google, Yahoo and Bing. It has been used by more than 3700 third-party applications.

At first, the Box Development Tool is used to extract configuration file as shown in Figure 40. This configuration file specifies its vendor (ARSlab), name (WebSearch) and version (1.0). In addition, it stores the author, description and view page (index.html), as well as ports information (input port “keyword”); note this box does not have output port.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<webapi
xmlns="http://wirecloud.conwet.fi.upm.es/ns/macdescription/1"
vendor="ARSlab" name="WebSearch" version="1.0">
<details>
<title>Web Search</title>
<authors>Sixuan Wang</authors>
<description>News search from more than 2 billion pages, integrated from Google, Yahoo and Bing web search APIs, used by more than 3700 apps.</description> ...
</details>
<contents src="index.html" useplatformstyle="true"/>
<rendering width="5" height="20"/>
<wiring>
<inputendpoint
    name="keyword"
    type="text"
    description="keyword"/>
</wiring>
</webapi>

Figure 40. Web Search Box configuration example

Next, Figure 41 shows the view page (index.html), which is much simple. It has several JS definitions (jQuery for the AJAX API call, xml2json for converting XML to
JSON, \textit{function.js} for the customized box function). It also has a test field “textarea” to show the box content.

\begin{verbatim}
<html>
<head>
    <script src='https://code.jquery.com/jquery-git.js'></script>
    <script src='https://x2js.googlecode.com/hg/xml2json.js'></script>
    <script src='js/function.js'></script>
</head>
<body>
    <div id="textarea"></div>
</body>
</html>
\end{verbatim}

\textbf{Figure 41. Web Search Box HTML view example}

Finally, another important file is the function.js (see Figure 42), which specifies the function when inputs comes. In general, \texttt{MashupPlatform.wiring.registerCallback ('keyword', inputfunction)} defines the input handling function “inputfunction”. This function calls the web search API with following parameters: the URL \url{https://faroo-faroo-web-search.p.mashape.com/api} and the input model name. Then it parses result and show its web content (e.g. titles and links) in “textarea” of the page view.

\begin{verbatim}
(function () {
    var inputfunction = function \texttt{inputfunction}(event_data) {
        var div = document.getElementById("textarea"); ...
        \texttt{generalRestGet(modelname, function (results) {
            var j_results = JSON.parse(results);
            for(i=0; i< j_results['results'].length; i++) {
                finalResult += JSON.stringify(j_results['results'][i]['title'])+
                JSON.stringify(j_results['results'][i]['URL']) ;
            }
            div.innerText = 'web search results from Google, Yahoo
and Bing for (' + modelname + ' spreading):
+finalResult; 
},... });
\end{verbatim}
```javascript
var generalRestGet = function generalRestGet(word, onSuccess, onError) {
    var URL = 'https://faroo-faroo-web-search.p.mashape.com/api?q=' + word;
    MashupPlatform.http.makeRequest(URL, {
        method: 'GET',
        "requestHeaders": { "Accept": "application/json" },
        onSuccess: function (response) {
            onSuccess(response.responseText);
        },
        onError: function () {
        }
    });
}
MashupPlatform.wiring.registerCallback('keyword', inputfunction);
})();
```

Figure 42. Web Search Box JS function example

After the boxes are developed, user can package it as a zip file and upload it to the M&S Mashup Platform. Figure 43 shows the uploaded boxes in the mashup platform.

Figure 43. Uploaded boxes for Fire Spreading M&S mashup

Now it is time to wire these boxes into a mashup. Figure 44 shows the wiring editor page in the M&S Mashup Platform. Users can drag and drop boxes from different box types and wire the boxes with their ports. There are three user inputs: input box (the fire spreading model in CloudRISE); initial file (the initial value for the fire-spreading model) and color file (the cell colors for fire-spreading model). We can wire the outputs of model
input box and initial file as the inputs of cell-Devs model simulation, so it can create a fire spreading experiment and run the simulation. After that, we can wire the cell-Devs model simulation with the Operator Box get log file from zip, so it can extract the log file and then wire the log to Log viewer for visualizing purposes. In another path, the input box wires to WebAPI Box Web Search to get the latest news about “fire spreading”.

Figure 44. Boxes wiring for Fire Spreading M&S mashup

Figure 45. Executing the Fire Spreading M&S mashup application
After the boxes have been wired, a new M&S mashup application is ready. After setting three inputs (model input URL, initial file URL and color file URL), the M&S mashup will run. The messages follow the wiring flow. Each box runs its function when receiving inputs and shows the corresponding view in the workspace. Figure 45 shows its execution view in Google Chrome. We can see that the MSaaS Box of model simulation has been executed successfully, and a fire simulation result was generated and shown with a timeline control (in which user can step forward and backward). In addition, this mashup also shows the queries of web search results. From this case study, we can see that anyone with basic web development knowledge (like HTML/CSS/JS) can easily develop boxes. Then users can simply select boxes and wire them as mashup, they do not need to have web development knowledge. In addition, since mashups can be shared as public URIs, so other users can visualize and run the mashup without any specific knowledge.

5.4 Summary

We presented a new method (i.e. M&S mashup) in the MAMSaaS architecture for building M&S mashups. M&S mashup allows fast application development by integrating heterogeneous data and services. This method is a layered architecture to deploy and identify M&S mashup components as well as link and execute mashups. M&S Mashup components are called boxes, which consists of MSaaSs, Web APIs, widgets and operators. Each box has its own structure, function and view. M&S Mashup is created through box wiring mechanism. We developed tools for developing boxes, wiring boxes and run mashups. We presented a prototype with a Fire Spreading M&S mashup. This prototype has shown that the proposed method can create simulation mashup in an easy and rapid way.
Chapter 6: Semantic Selection

The objective of MAMSaaS is to simplify the development and integration of M&S applications. In previous chapters, we have presented different approaches: 1) to simplify the development of MSaaS services in the Cloud, 2) and to integrate these services and other M&S resources as M&S mashups. However, another issue to be simplified is to select right resources and services. To do so, we proposed a new semantic selection method in the Tag Ontology Layer of MAMSaaS.

There are two motivations for this semantic selection method. First, it is hard to identify the meanings of M&S resources. When developing M&S applications, there could be hundreds of or thousands of resources. These resources can be either M&S resources (like models), or M&S-related services (like simulation services and Web APIs). It is hard to understand what they are and how they are related to each other; as a result, it is hard to select from them. Second, ontologies have been used in M&S for years. Most ontologies are developed manually by skilled ontology experts. These pre-defined ontologies are hard to build and understand. They are limited to specific cases and rules. Little has been done about learning domain-specific ontologies for M&S.

In order to handle the above issues, we propose a new semantic selection method. Our work is the first one trying to learn a tag hierarchy as a domain-specific ontology in M&S. It can identify and select M&S resources based on their semantic. The general idea is to automatically mine tag signatures from M&S resources, learn the tag-tree ontology from these signatures, and then compose resources based on the learnt ontology.

6.1 Semantic Selection Methodology

The semantic selection method is in the Tag Ontology layer of MAMSaaS. This layer works closely with other layers (like the Box Layer and the Wiring Layer). It provides new tag mining and ontology-learning algorithms. The tag-mining algorithm can mine tag signatures from boxes. These signatures can best represent the meanings of boxes. The ontology-learning algorithm can construct a domain-specific ontology from tag signatures, without the need of pre-defined ontologies.
Figure 46. Procedure of the Tag-tree based semantic selection method

We can break down this semantic selection method into following sub processes, shown in Figure 46.

**Step 1: M&S Resource Signature**: it is responsible to identify each M&S resource in a unified signature. The signature tells what the resource is and what inputs/outputs are.

**Step 2: Tag Mining**: it is responsible to get tags from resource signatures. Different natural language processing functions are used in this step.

**Step 3: Tag Tree Learning**: it is responsible to learn a domain-specific ontology based on resource tags. Different syntax and semantic methods are considered.

**Step 4: Semantic Selection**: it is responsible to suggest resources based on their tags and the learnt tag-tree. It helps users to select from a large amount of resources.

This Tag Ontology layer of MAMSaaS has many advantages. First, it can automatically add semantic to the resources by using tag-mining techniques. It explores the descriptions of M&S resources (e.g. Box Signatures or Web resource service descriptions). These descriptions can be easily obtained from users. As this mining process can be automated, each resource will have several tags attached. These tags imply the semantic of the resources.
Furthermore, it can learn the tag-tree ontology based on the user-interested resources. Rather than depending on domain ontologies (like DeMO [Mil04]) or semantic web ontologies (like OWL-S [Owl15b]), this method uses tag-based learning techniques to construct tag-tree all by itself. This method can consider most syntactic, semantic and structural issues.

Likewise, it can be used to integrate resources in an automated way. Most selection methods are done by linking different interfaces manually. For example, boxes are wired by wiring engineers via their ports. This method, instead, uses tags and the learnt tag-tree. This can be done because: 1) the resources have semantics already attached by their tags, and 2) the tag-tree has been built from these tags.

Finally, the method has better user participation. The method provides different ways to search (by name/input/output), allowing users to select right resources for building applications. This layer works together with other layers in MAMSaaS, and is always available on the cloud.

Note that this semantic selection method is a general approach to find the relations among a large amount of resources. It works for but not limited to the boxes. Any resource that supports the basic signature structure (i.e. inputs and outputs) can apply this method. For instance, in Section 6.3, we will show an example using this method in the semantic model selection.

6.1.1 Identifying Resources – Signature

As discussed above, in this semantic selection method, the resources are more general than Boxes. In this sense, we defined a less restricted signature, shown below.

\[
\text{Resource Signature} = < R, I, O > \quad (\text{Definition 4})
\]

\( R = < R_n, R_t, R_d > \) is the general information of the resource, including method name, type and its text description;

\( I = \{ p \} \) is a set of input parameters;

\( O = \{ p \} \) is a set of output parameters;

\( p = < p_n, p_t, p_d > \) is a set of parameters, including parameter name, type and its description.
We define each resource as a collection of the operation method \((R)\), its input \((I)\) and output \((O)\) parameters. \(R\) is the general information of an operation, including its name, type and description. For instance, “GetWeather” is an operation name, “REST” is the type and “return weather forecast information” is the description. \(I/O\) is a set of parameters; each parameter can have its parameter name, type and description. For instance, an input parameter has “ServerTitle” as its name, “xsd: string” as its type and “server name” as its description.

M&S resource can be of any type. For example, it can be any type of Box (like MSaaS, WebAPI, Widget, Operation), model (like DEVS atomic/coupled models), or other type of M&S-related resource. In most cases, each resource has its own description file (e.g. WADL for REST, WSDL for SOAP, EF for MSaaS, and definition file for model). This signature can be extracted automatically through its description file (e.g. reuse the Box Development Tool discussed in Section 5). The only assumption here is that the descriptions are “meaningful”, i.e., the users provide enough useful information for building an ontology. Any element of the signature can be optional.

```
<resource path="util/ping">
  <method id="GetServerInfo" name="GET">
    <doc>Get server information</doc>
    <request>
      <param name="user_name" type="xsd:string"/>
      <doc>user name...</doc>
    </param>
    ...
  </request>
  <response>
    <representation mediaType="text/xml"/>
    <param name="server_title" type="xsd:string"/>
    <doc>server name</doc>
    ...
  </response>
</method>
</resource>
```

**Figure 47. WADL Example from RISE**

Let us consider a REST simulation service as an example to show how to build the defined Resource signature (Figure 47). We can get each element of this signature directly from the WADL, and the resulting mapping is shown in Table 5. It is a straightforward process, we extract \(<Method>\) and \(<doc>\) directly for the \(R\) in signature. We get the \(<param>\) of \(<request>/<response>\) in order for the \(I/O\) in signature, and we
put the attribute “name” as the structure name. If this involves complex data structures, we iteratively get all the basic parameters.

**Table 5. Mapping WADL and Resource signature**

<table>
<thead>
<tr>
<th>WADL element</th>
<th>WADL attribute</th>
<th>Resource signature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Method&gt;</code></td>
<td><code>id, name, text</code></td>
<td>R</td>
<td>GetServerInfo, GET (REST), server info description</td>
</tr>
<tr>
<td><code>&lt;doc&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;request&gt;</code></td>
<td><code>&lt;param&gt;</code></td>
<td>I</td>
<td>user_name, xsd: string, ...</td>
</tr>
<tr>
<td><code>&lt;response&gt;</code></td>
<td><code>&lt;param&gt;</code></td>
<td>O</td>
<td>server_title, xsd: string, ...</td>
</tr>
</tbody>
</table>

### 6.1.2 Getting Meanings - Tag Mining

After finding the M&S resource signatures, a second issue is to get their “meanings”. In M&S, usually we need to add another semantic layer. To do so, we are using the tagging system and mining techniques to get tags from the resource signatures automatically. The resource is attached with semantics by tags.

We define resource tag signature in Definition 5. Briefly, each resource signature has a corresponding Resource Tag Signature, and each element of resource signature corresponds to a set of tags.

**Resource Tag Signature = < T_m, T_i, T_o, T_d > (Definition 5)**

- T_m is a set of tags for the resource name, type and its text description.
- T_i = {tp} is a set of tag for the input parameters;
- T_o = {tp} is a set of tag for the output parameters;
- tp = < tp_n, tp_t, tp_d > is a set of tags for a parameter, including name, type and description tags.

As is conventional in tagging systems (also termed as Folksonomy) [Wal14], a user can tag a resource with any terms. Its data can be viewed as triples (user, tag, resource) [Hey06]. The difference of the usage of tags here is that tags automatically mined from their signature instead of being directly specified by users. Users can provide any kind of description files for resource signatures. We can use the box development tool to extract the signatures from these files.
To mine the tags from resource signatures, the tag mining function \( \Gamma \) is used (Definition 6).

**Tag Mining Function** \( \Gamma: S_{\text{term}} \rightarrow S_{\text{tag}} \)  (Definition 6)

\( S_{\text{term}} \) is a set of terms of elements from the resource signature. It can consist of any form of terms \( \{e1, e2, ...en\} \).

\( S_{\text{tag}} \) is the set of tags after tag-mining \( \{t1, t2, ... tn\} \).

The tag-mining function is a function that maps a set of terms into a set of semantically meaningful tags. The idea is to reduce the terms in signature and generate the tags that best represent the resource. The mining function has two steps: Syntactic pre-processing (to handle basic syntactic issues) and Semantic pre-processing (to handle basic semantic issues).

1) Syntactic pre-processing

**Tokenization:** a resource structure can contain multiple terms. A term either can contain delimiters, like \{‘_’, ‘-’, ‘/’, ‘(’, ‘)’, ‘:’, ‘=’, ‘;’, ‘,’\}. The text element is first isolated by punctuation, and then each term is found using the delimiters. For example, *the user_name for Server* will be tokenized into \{the, user, name, for, Server\}.

**CamelCase:** Some terms are CamelCase, which are compounded words (e.g. *ServerName*). We separate this kind of word into different terms \{Server, Name\}.

**StopWordFilter:** Some words do not make sense for the further tag ontology learning. We pre-defined a filter list. Any word in this list is not considered a tag, like \{‘the’, ‘a’, ‘and’, ‘it’, ‘its’\}.

2) Semantic pre-processing

**Part of Speech (POS):** Meaningful tags should only be noun or verb terms. We use the online API (e.g. [http://www.merriam-webster.com/dictionary/](http://www.merriam-webster.com/dictionary/)) to get POS of terms and only keep noun and verb terms. Another benefit is that it will filter other types of term, like numbers, non-English words, etc.

**Abbreviation:** Some terms are abbreviation, e.g. *ID vs. identification, Dr. vs. Doctor*. We use the online API (e.g. [http://www.abbreviations.com/](http://www.abbreviations.com/)) to get the most popular full name of each abbreviation and replace it with the full name.
**Plural noun and verb conjugation:** We keep only the single form of the noun and the infinitive form of the verb. For example, *apples* will be *apple*, and *retrieves* will be *retrieve*.

Synonyms like *{car, automobile}* and spelling differences like *{color, colour}* are not considered. We leave them to the tag-tree learning part. The reason is to reduce the tags. The idea is to let the tag-mining function handle as many issues as possible by reducing the number of tags, and then leave the remaining tags for the next tag-tree ontology-learning phase.

### 6.1.3 Learning Ontology – Tag Tree

At this point, we have all tags for M&S resources. In this section, we will discuss the Ontology Learning algorithm. It can build a tag-tree ontology, considering all kinds of relationship between these tags, including their structural, syntactic and semantic similarities.

The ontology-learning process is based on an assumption that tags tend to express the same concept if they frequently occur close together. In other words, frequent tags keep the semantics of their resources. Here we define the Tag Co-occurrence Graph to capture the frequency relationships among tags (see Definition 7).

**Tag Co-occurrence Graph**  
\[ G = (T, E) \]  
*(Definition 7)*

The Tag Co-occurrence Graph is a weighted graph. It stores the tags and their relations. *T* is the vertex set, and *E* is the edge set. Each vertex corresponds to a tag while each edge corresponds to the frequency relationship between two connected tags.

![Figure 48. Co-occurrence graph (a) & Tag tree ontology (b)](image-url)
Figure 48-a shows an example of a co-occurrence graph, it has six tags (i.e. a-f) and the frequency relationship between each two tags. For example, the value 2 on the link between tag a and tag e, indicates that these two tags have occurred twice.

This graph is calculated with the following process. For a tag signature, it gets each tag set. For this tag set, it checks each tag pair (any two tags of the tag set) and performs the following steps. First, if any tag in it does not appear in the vertex set $G(T)$ of $G$, we add this tag into $G(T)$. Second, it updates the edge value of this tag pair on G (increasing it by 1). Finally, we can get the co-occurrence graph.

The co-occurrence graph can show how often two tags appear together. However, we still need to build a tag hierarchy. The reasons of doing so are as follows:

1) The graph is not structured, it only tells us how often two tags appear together, not their hierarchical relations. The only relation between tags is the frequency number, which contributes little for the purpose of selecting semantics. For instance, it is hard to solve the structure issues like generalization and association. The hierarchy of the given tags will help to handle the structure issues.

2) Even though there are some clustering algorithms for grouping the tags, the issue is that they are coarse-grained thus not suitable for heterogeneous M&S resources. The clusters cannot tell us the relations between two different clusters, which limits its use in semantic selection.

Therefore, we need to build the tag hierarchy from frequent tags. The general idea is that the tags in higher levels are more general and abstract, while the tags in lower levels are more detail and concrete.

The tag-tree ontology is defined in Definition 8.

**Tag hierarchy Tree**  
$TR = (T, E)$  
(Definition 8)

$TR$ is a Directed Acyclic Graph. It consists of $T$ and $E$. $T$ is a set of vertices that represent tags $\{t1, t2, ..., tn\}$, and $E$ is a set of edges that represent “sub-tag” relationship between two tags (formally $t1 < t2$).

Figure 48-b shows a tag-tree example. Currently, the only relationship is sub-tag, which is learned from the tag co-occurrence graph. The sub-tag relationship shows the
equivalence of tags from the semantic point of view. It implies that if a resource can be
described as a child tag, it can be described as the parent tag as well. The positions of tags
reflect their semantics. The tag-tree ontology acts as a simplified domain-specific
semantics. It represents knowledge using tags. Unlike complex structures in normal
ontologies, a tag-tree ontology only includes tags and their hierarchy, which makes it
easy to understand and modify. A tag-tree can also be a subset of an existing ontology
defined by semantic experts. Users are also free to define their own tag-trees.

Now, let us see how to learn the ontology. We defined a **Tag Tree Learning**
Algorithm based on Heymann’s algorithm [Hey06], which can the learn tag-tree from a
tag-occurrence graph. However, Heymann’s algorithm cannot be directly used for M&S
semantic selection because it only works for connected tag graphs. Heymann’s algorithm
assumes that there should always be a path between two tags, and the weight of two
connected tags should always be one. Therefore, we cannot use it (i.e. the tags mined
from resources can be anything; the tag graph could be weighted and unconnected). In
addition, Heymann’s algorithm uses the cosine similarity, and it does not consider all
kinds of relationships between tags (i.e. frequency, syntactic or semantic similarities).

Instead, our algorithm deals with these issues. It is an iterative process to learn tag-
tree from a tag co-occurrence graph. Each iteration has two major steps: 1) select a best
tag to be added into the tree; and 2) find a best position in the tree to place the selected
tag. In general, for the step to select the right tag, we use centrality theory and modify its
closeness centrality equation to support the weighted and disconnected graph. For the
second step to place the tag in the right position into the tree, we use a similarity function
that will consider all the structural, syntactic and semantic issues.

The algorithm has four inputs. 1) The tag co-occurrence graph that we get from tag
resource signatures. 2) The centrality function that can get closeness centrality from a tag
co-occurrence graph. 3) The similarity function that calculates how similar two tags are.
4) The threshold to determine where to put the tag (as a child of a target tag or under the
root).

At beginning, the algorithm initializes the tree $TR$ with a *root* note representing the
top of the tree. Then, it calls the **centrality function** $G$ to get the closeness centrality list
of given tags in descending order. This centrality list indicates how central the tag is in
the co-occurrence graph. Next, it selects each tag in the closeness centrality list and attempts to add the tag to the tree. The similarity function tells the algorithm where to add the tag in the tree. It checks the similarity with each tag in the existing tree and finds the highest one to attach. If the highest similarity is large enough (greater than the predefined threshold), it adds the tag as a child of the tag with the highest similarity; otherwise, it adds the tag as a child of the root note.

Algorithm 1: Tag Tree Ontology Learning

Input: $G = (V, E)$, is a tag co-occurrence graph

Input: Centrality ($G$), is the centrality function, it gets the descending order of the closeness centrality from a tag co-occurrence graph

Input: Similarity ($ti, tj$), is the similarity function of two tags, it considers the structure, syntactic and semantic issues

Input: $threshold$, is the value to determine whether to put the tag as a child of a related tag or under the root

Output: $TR = (T, E)$, is a tag-tree learnt from the given $G$

Def $TR = null$

Add $< null, root >$ into $TR (T, E)$

List $l_{cen} < v > = Centrality (G)$

For each $v$ in $l_{cen}$ do

Def $maxSim = 0$, $tagToSave = null$

For each $t$ in $T$ of $TR(T,E)$ do

If $Sim(v, t) > maxSim$ then

$maxSim = Sim(v, t)$,

$tagToSave = t$

end if

if $maxSim > threshold$ then

$add<t,v>$ into $TR (T, E)$

else

$add<root,v>$ into $TR (T, E)$

end if
Next, the centrality function and the similarity function are discussed further below.

6.1.3.1 Tag selection – centrality function

To select the right tag from graph, we use a greedy algorithm to order tags. In graph theory and network analysis, centrality measures relative importance within a graph [New10]. The closeness of nodes is defined as the sum of its distances to all other nodes [Bav50]. The formula of closeness of tag $i$ is:

$$
\text{closeness}(i) = \frac{1}{\sum_j D_{ij}} \quad (\text{Formula 1})
$$

where $D_{ij}$ is the shortest distances between tag $i$ and tag $j$.

Thus, the more central a node is, the lower its total distance is to all other nodes. Closeness shows how long it will take to spread information from a node to all other nodes [Sab66]. However, as discussed above, Heymann’s algorithm has the shortcoming that it cannot be applied to a weighted and disconnected graph. In Heymann’s algorithm, each edge is one, cannot be a weighted value. Furthermore, when the distance between tags in disconnected, the closeness of all the tag will be a same value (zero); this will affect the accuracy of the algorithm.

In the tag co-occurrence graph, each edge links two tags (e.g. $t_i$ and $t_j$), and the edge is the occurrence frequency number (e.g. $F_{ij}$). In order to overcome the lack of weighted graph, we redefine the distance $D_{ij}$ of two tags; in order to overcome the disconnected parts in the graph, we calculate the closeness of tag $i$ using Formula 2.

First, if the tag $i$ and tag $j$ are connected directly (i.e. they have a frequency number on their edge), we calculate the $D_{ij}$ as the inverse of their frequency number. Generally, the more frequently a tag occurs in the graph, the less distance this tag is to others. By doing so, the edge is weighted.
closeness(i) = \sum_{j} \frac{1}{D_{ij}} \quad \text{(Formula 2)}

where \( D_{ij} = \begin{cases} 
\frac{1}{F_{ij}} & \text{if tag } i \text{ and tag } j \text{ are connected directly} \\
\text{shortest distance between them,} & \text{else}
\end{cases} \)

Second, based on \( D_{ij} \), we can calculate the closeness of each tag in the modified equation in Formula 2. The closeness is calculated as the sum of the reverse value of its distances to all other nodes (instead of the inverse of the sum of distances). The higher the closeness is, the more frequent this tag will be. If two tags are not connected (e.g. tag \( b \) and tag \( h \) in Figure 49), the distance of them will be infinite (e.g. \( F_{bh} = 0 \), so \( D_{bh} = \infty \)). Then, the closeness of these two tags will be 0 (e.g. \( 1/D_{bh} = 1/\infty \approx 0 \)). Therefore, the unconnected tags will contribute nothing to the closeness calculation.

![Figure 49. Modified closeness centrality](image)

According to the above discussion, the algorithm of centrality function is:

**Algorithm 2**: Centrality function

*Input*: co-occurrence graph \( G=(V, E) \)

*Output*: closeness centrality list \( l_{cen} <v, c> \)

*For each edge \( e \) in the \( G \) do*

\[ D_{ij} = 1/F_{ij} \]

*End for*

*For each tag \( i \) in \( V \)*

99
Def closeness for tag i \( C_i = 0 \)

For each \( t_j \) in \( V \)

\[ D_{ij} = \text{Dijkstra} \left( t_i, t_j \right) \]

\[ C_i = \frac{C_i + 1}{D_{ij}} \]

Save \( <t_i, C_i> \) into \( l_{cen} \)

Sort \( l_{cen} <v, c> \) in descending order of \( c \)

The centrality function is able to order the tags. It assigns the weight of each edge with the inverse value of its frequency. For each tag, it calculates the sum of inverse distances to all other nodes, and then it saves the results in the centrality list. Finally, it orders the centrality list in descending order. The results of sorted centrality list can tell us how to select them when we construct the tree.

6.1.3.2 Finding position to place – similarity function

The above algorithm is used to select tags from graph, while the similarity function finds the best position in the tag-tree. In general, the similarity function compares two tags and returns their similarity. The learning algorithm uses this similarity function (see Definition 9) to check each tag in the existing tree and find a tag with a highest similarity.

**Similarity function** \( \text{Sim} \left( t_i, t_j \right) = \left( \frac{1}{D_{ij}} \right) * \text{SYN}_{ij} * \text{SEM}_{ij} \quad \text{(Definition 9)} \)

\( D_{ij} \) is the shortest distance of the two tags in the co-occurrence graph.

\( \text{SYN}_{ij} = (1 - \text{LevRatio}) + \text{LevRatio} * \frac{\text{Lev} \left( t_1, t_2 \right)}{\max(\text{len}(t_1, t_2))} \), \( \text{is the syntactic similarity value of the two tags.} \) \( \text{Lev} \) is the Levenshtein distance, \( \text{LevRatio} \) is the ratio that \( \text{Lev} \) takes effect, and it is between 0 and 1.

\( \text{SEM}_{ij} = \text{SemRatio} \) (if \( t_i \) and \( t_j \) are synonym); 1-\( \text{SemRatio} \) (otherwise). \( \text{SemRatio} \) is the value if the two tags are synonym; it is between 0 and 1.

First, \( D_{ij} \) calculates the structure of the tags; it follows the Formula 2 defined above. The reason to include \( D_{ij} \) in the similarity function is that it can indicate how close tags are to each other.
Second, we consider syntactic variations by using Levenshtein distance. In information theory, the Levenshtein distance, also called edit distance, is for measuring the difference between two sequences [Sol11]. The Levenshtein distance between two words is their minimum number of edits (i.e. insertion, deletion, substitution) required to change one word into the other. For example, the Levenshtein distance between "kitten" and "sitting" is 3 (i.e. substitution of "s" for "k", substitution of "i" for "e", insertion of "g"). Here, we adopt the Levenshtein distance to check their editing distance between two tags, and we divide this distance by the maximum length of the two tags, then we multiply the result with a ratio $LevRatio$ (see Definition 9).

Third, we consider semantic synonyms (e.g., car vs. automobile). $SemRatio$ is the value to be used in this function if the two tags are synonym. Here we check whether the two tags are synonym or not. It adopts using WordNet [Fel98] for finding synonyms. Generally, for given tags $t1$ and $t2$, we will get the synonym list $SL1$ for $t1$ and $SL2$ for $t2$. If either $t1$ is in $SL2$ or $t2$ is in $SL1$, we will calculate the similarity with a predefined ratio $SemRatio$; otherwise, we will use the ratio $1 - SemRatio$.

6.1.4 Recommending resources – Semantic Selection

So far, M&S resources are identified by resource signatures; these signatures are further attached with tags; and then we can derive the tag-tree ontology from these tags. In this section, we will discuss how to use these artifacts to select resources semantically.

Here we define Composable Resources (Definition 10).

**Composable Resources** $R1 \rightarrow R2$ (Definition 10)

$R1=<T_{m1},\{T_{i1}\},\{T_{o1}\}>$ and $R2=<T_{m2},\{T_{i2}\},\{T_{o2}\}>$ are two Resource Tag Signatures for a given tag-tree ontology $TR = (T, E)$. $M1 \rightarrow M2$ are said composable if they satisfy that $\forall t2 \in T_{i2}; \exists t1 \in T_{o1}; \exists t1 = t2$ or $t1 < t2$ in $T$.

Let us assume there are two Tag Signatures $R1$ and $R2$. If for any tag $t2$ in tag set $T_i$ of a $R2$’s input port, there is a tag $t1$ in tag set $T_o$ of an output port of $R1$, and $t1$ is equivalent to or it is a “sub-tag” of $t2$ (formally $t1 < t2$), then $R1$ can be composed with $R2$ (formally $R1 \rightarrow R2$). Tags are semantically equivalent if they have a relation in the
tag-tree. If the tags produced by a resource are semantically equivalent to the tags accepted by another resource, these two resources could be composed.

The composable resources can suggest “dataflow”, as in Definition 11.

\[ \text{Dataflow} \ R1 \rightarrow R2 \rightarrow R3 \quad (\text{Definition 11}) \]

\( R1, R2, R3 \) are three tag resource signatures, if \( R1 \rightarrow R2 \), and \( R2 \rightarrow R3 \), a dataflow among \( R1, R2, R3 \) is \( R1 \rightarrow R2 \rightarrow R3 \).

That is, for given sets of resources, if there are composable resources and these resources can be linked together, a dataflow can be obtained from them. The linking of a resource has transitivity, which can help to wire boxes and build mashups.

![Diagram](image)

**Figure 50.** An example of resource composition based on a tag-tree ontology

Let us show how to select resources using this method (see Figure 50). Here we use a GIS related M&S case. In this case, there are different resources (i.e. Api1, Api2, and Api3). *Api1* is a SOAP resource (which can get geographical information about a location). *Api2* is a REST resource (which can get a wildfire Cell-DEVS model). *Api3* is a REST resource (a simulation service for a fire model). Other resources include *IPlocationDetector, SimulationRunning, SimulationResults, GoogleMapVisualization*, etc. For these three resources, the left part of Figure 50 shows their Tag Signatures and the upper-right part of Figure 50 shows a tag-tree. Since there is a “sub-tag” relation between *geography* (a tag of *Api1*’s outputs) and *layout* (a tag of *Api2*’s inputs), it satisfies the
conditions of composable resources, so we say \( \text{Api1} \) is composable to \( \text{Api2} \) (formally \( \text{Api1} \rightarrow \text{Api2} \)); similarly, we can say \( \text{Api2} \rightarrow \text{Api3} \) since \( \text{wildfire} \prec \text{fire} \), \( \text{model} = \text{model} \), \( \text{celldevs} \prec \text{devs} \). Then, a data flow \( \text{Api1} \rightarrow \text{Api2} \rightarrow \text{Api3} \) can be built.

In addition, these composable resources can recommend potential resources for users, as defined in Definition 12.

**Resource Recommendation** \( \text{R1 recommends } \text{R2} \) \textbf{(Definition 12)}

\( \text{R1} = <T_{m1}, T_{i1}, T_{o1}> \) is a given resource and \( \text{R2} = <T_{m2}, T_{i2}, T_{o2}> \) is a resource that the users may be interested, and a given a tag-tree ontology \( \text{TR} = (T, E) \). \( \text{R1 recommends } \text{R2} \) if they satisfy that if \( \text{R2} \rightarrow \text{R1} \) or \( \text{R1} \rightarrow \text{R2} \).

That is, for a given resource \( \text{R1} \), it can suggest semantically equivalent resources (i.e. tags that are identical or have a sub-tag relation). The suggested resource can be either a predecessor or a successor. It cannot only suggest boxes, but also help the wiring engineer to wire boxes.

It can also help users discover resources based on queries, as defined in Definition 13:

**Resource Discovery** \( \text{R1 meets Q} \) \textbf{(Definition 13)}

\( \text{R1} = <T_{m1}, T_{i1}, T_{o1}> \) is a tag resource signature, \( \text{Q}(Qm/Qi/Qo) \) are tag sets given by users for resource name/inputs/outputs, and \( \text{TR} = (T, E) \) is a given tag-tree. We define \( \text{R1 meets the query Q} \) if: 1) \( \forall qm \in Qm, \exists tm \in Tm U Td, \exists qm = tm \) or \( tm < qm \); 2) \( \forall qi \in Qi, \exists ti \in Ti, \exists qi = ti \) or \( ti < qi \); 3)\( \forall qo \in Qo, \exists to \in To, \exists qo = to \) or \( qo < to \).

That is, if for any particular \( \text{tag } t1 \) in the query of name/input/output, we can always find a semantic equivalent \( \text{tag } t2 \) in tag set of a resource \( \text{R1} \), and then we can say \( \text{R1} \) meets the query.

Resource recommendation and resource discovery are different. The resource recommendation is a relaxed query to find related resources. Resource discovery is a strict query to find the exact resources that meet queries of name/input/output. Both of
them can increase user participation. We will discuss more to see its usage in mashups in Chapter 7.

Note that one limitation of the above semantic selection methods (i.e. composable resources, data flow, resource recommendation and resource discovery) is that they only support sequential composition among resources. The reason is that all the semantic selection methods depend on the tag-tree ontology, which only supports “sub-tree” relationship between tags. This can be improved by extending the tag-tree ontology with more relations and constrains. For instance, we can improve the learning algorithm by analyzing other semantic relations of tags (e.g. junction/disjunction). Also, ontology experts can improve the tag-trees with specific constraints (e.g. business modeling process rules).

6.2 Implementation

We developed a Python project (called PyCom) to implement the semantic selection method.

![Figure 51. PyCom Class Diagram for semantic selection](image)

PyCom uses two kinds of classes (see Figure 51): data type and logic. The data type classes include ModelDescription, ModelDescriptionHandler, TagSignatureList, TagSignatureHandler, TreeList, and TreeHandler. These classes store data in XML.
Their handler classes load and save these XML files into a list. For the logic classes, TagMining implements the functions in Section 6.1.2 to mine tags. TagTree-Learning has functions to learn a tag-tree ontology based on the proposed learning algorithm (as in Section 6.1.3). The resource selection can check whether two resources are composable. The Discovery can find resources that have semantically equivalent tags as input/output queries. The Recommendation can suggest resources to users. Note that PyCom can be extended for other specific functions.

To test its performance, several experiments have been carried out [Wan14d, Wan15b]. The experiments have been carried out on a personal computer (Intel(R) Core(TM) i7 CPU @ 1.73GHz, 4GB RAM, Windows7 64-bit OS). The test set has a total 659 resources. They are from different kinds of M&S resources, including MSaaS and WebAPI boxes. There are REST-based, SOAP-based, HTML-based, or EF-based.

1) REST-based WSs contain 34 popular resources (e.g. Twitter, Facebook, Delicious, GitHub) and 44 resources from the current CloudRISE MSaaS. We get their resource signatures were found from WADL and EF files using the Box Development tool.

2) SOAP-based WSs contain 22 resources of programmableweb.com (e.g. WeatherPrediction, CurrentTime, CurrencyConverter, and IPtoGeolocation). We get their resource signatures from WSDL using the Box Development tool.

3) HTML-based pages contain 248 model resources from ARSLab model repository (http://www.sce.carleton.ca/faculty/wainer/wbgraf/doku.php), 299 publication resources from ARSLab publication repository (http://vs3.sce.carleton.ca/publications/) and 11 open resources (e.g. Google, YouTube, Flickr). These resources are REST-based. They use GET method to access resources. We get their resource signatures from HTML description pages.

Then, we run tag mining to get their tags. Next, we learn tag-tree from these tags. Some of the basic variables for the learning algorithm are as follows: the threshold of the tag is 10 (to remove tags with lower frequency). The threshold tag pair is 3 (to remove tag pairs with lower frequency). The similarity threshold is 0.2 (to determine to place the tag as a child of candidate tag or under the root); the LevRatio is 0.2 (for the ratio of
editing length in terms of their syntax); the SemRatio is 0.2 (for the ratio of the synonyms).

The final tag-tree results have a total of 205 tags and 3519 tag pairs. Figure 52 shows a part of it. The tag “model” is in the second level and it is the parent of all other tags, it is reasonable since almost all the tag signatures have the word “model”. Some hierarchies are interesting, for example, “model -> devs -> cell” or “simulation -> event -> external”.

In order to evaluate the performance of our algorithm and Heymann’s algorithm, we implemented the Heymann’s algorithm and did some tests. As discussed above, Heymann’s algorithm does not support disconnected and weighted graph, so our test here is based on the cases that both algorithms can solve (i.e. connected and un-weighted graph).

![Figure 52. Part of learnt tag-tree ontology from simulation services](image)

![Figure 53. Tag-tree learning algorithm vs. Heymann’s algorithm](image)
We measured the overall performance on all tags. In particular, for each “child” tag \( C \) (i.e. the leaf nodes), we get all of its “parent” tags (i.e. all the tags on the branch between tag \( C \) and \( \text{Root} \)). Next, we construct tag pairs under different criteria. Tag pairs contain each “child” tag on one side, and each “parent” tag of the “child” tag on the other side. The criteria are taken from [Mar09] [Per12]:

\[
\text{Structure}_\text{Union} = \frac{R(t1 \cap t2)}{R(t1) + R(t2)}
\]

are the resources that are tagged by both tags over the resources that are tagged by either tag. It represents the Jaccard similarity coefficient.

\[
\text{Structure}_\text{Min} = \frac{R(t1 \cap t2)}{\min(R(t1), R(t2))}
\]

are the resources that are tagged by both tags over the minimum tagged resources between them. It is one variation of the Jaccard similarity coefficient.

\[
\text{Structure}_\text{Max} = \frac{R(t1 \cap t2)}{\max(R(t1), R(t2))}
\]

are the resources that are tagged by both tags over the maximum tagged resources between them. It is one variation of the Jaccard similarity coefficient.

The performance test is shown in Figure 53. We get two trees from 659 M&S-related resources: one uses our algorithm and the other uses Heymann’s algorithm. Figure 53 shows the test results (i.e. average values of each criterion mentioned above). Our proposed learning algorithm has significant better performance: Structure_Min (improved by 119%), Structure_Union (improved by 38%), and Structure_Max (improved by 26%). The results have shown that our algorithm has a better structure to represent the tag tree.

6.3 Case Study: Semantic Model Selection

As discussed above, this semantic selection method is not limited to boxes. It is a general approach for any resource that has input/output. In this sense, we have successfully applied this method in DEVS model selection.

Model is one of most important M&S resources. In recent years, the M&S community has investigated new methods to reuse models in order to save time and reduce costs. One way to increase such reuse is through model composition, defined as
the process of selecting and assembling models in various combinations to meet user requirements [Pet03].

![Diagram showing model composition, repository and simulation]

**Figure 54. Model composition, repository and simulation**

Figure 54 shows the idea of model composition and its relations with repositories and simulation. Using model composition, we can select and assemble models from a repository and then simulate the assembled model. Model composition is more than just putting models together: it is also used to recombine components for different requirements [Pet14]. Model composition can be addressed at the syntactic and semantic levels. Syntactic composition focuses on the actual implementation of composition while semantic composition focuses on the underlying abstraction and meaning of the models [Dav04].

Even though much work has been done in syntactic composition, semantic composition research is still in its early stages. In particular, little research has been done in semantic selection for model composition, which requires that the users understand the meaning of models at the semantic level. The major issues are as follows: 1) users cannot fully understand the formal definitions of models [Roh07]. 2) It is hard to get the “meaning” of the models [Tol05b]. 3) There is no way to build domain specific ontologies just based on the models [Wan14b].

### 6.3.1 Semantic Model Selection Process

We want to improve the selection for model composition based on semantics. To do so, we applied the semantic selection method discussed in this chapter. Here we focus on the semantic selection for model composition. Note we will not focus on assembling
heterogeneous models, but rather on selection of individual models for composition. We are particularly interested on the semantic selection of DEVS models [Zei00].

As discussed above, in order to understand existing models better, we apply the resource signature to models. We can get these signatures in different ways: 1) the modeller can provide it; and 2) it can be constructed from a well-defined modeling language automatically. For example, CD++ [Wai09] is a simulation environment to execute DEVS models. We can extract most parts of signature from the CD++ model definition file. We keep these model signatures in XML. Theses files follow the format shown in Figure 13 and are stored in cloud.

As mentioned before, a second issue with semantic composition is how to get the meaning. In our case, we apply the tag-mining function to mine tags from the model signature and build tags for it. The signature is saved in XML and stored together with its signature.

After we have all tags of models, we learn a domain-specific tag-tree ontology for these models using the learning algorithms introduced before.

Then, we can use the Composable Models that we discussed for semantic selection. Based on the composable models, we can recommend users with the models. We can suggest semantically equivalent models (i.e. tags are identical or have a sub-tag relation). For DEVS models, the composable models suggest potential coupling connections (i.e. linking input and output ports of two models).

6.3.2 Success Rate Testing

We tested our approach using an existing repository of CD++ models. We chose 16 model samples at random (with a total 103 sub-models). They are DEVS models and cover a wide range of domains, such as network protocols, mobile phones, alarm systems, secure area access, computer networks, and transportation. Each model has different sub-models that could be reused for new purposes.

In order to test the effect of the model description, we used two different versions. In the first version, we obtained the model and port names from its model definition file, and built the description automatically. Note that in this version the ports only have name but no description. For the second version, we improved them with meaningful text. We studied each model carefully and got detailed descriptions.
Then, we used the tag-mining functions to get tags and built a tag-tree from these tags following the tag-tree learning algorithm. We got two trees from the two versions of description. The tag hierarchies of the two trees were reasonable. Most sub-tag relations were correct. For example, $\text{feedback} < \text{notify} < \text{model}$; $\text{wireless} < \text{network} < \text{data} < \text{system} < \text{model} < \text{treeRoot}$. Note that the trees are domain-specific and can be modified by the users. We customized the second tree (e.g. removing a few tags that were nonsense, like $\text{access} < \text{area}$).

The tests were based on the idea that for existing DEVS models, we know how the models can be composed, and we use them as an expected set. Using our approach, we get a learnt set that suggests models. The idea of this test is to compare how similar these two sets are. We measured the overall success of the two sets using popular metrics in graph similarity analysis: sensitivity ($S$), precision ($P$), and $F$-measure ($F$), which are defined in Definition 14.

\begin{align*}
\text{Composition Metrics:} & \quad (\text{Definition 14}) \\
S &= \frac{C}{C+M}; \quad P = \frac{C}{C+E}; \quad F = \frac{2}{1/S + 1/P}
\end{align*}

Where, $C$ (Common) is the common number of couplings in both sets, $E$ (Extra) is the number of couplings that only exists in the learnt set; and $M$ (Missing) is the number of couplings that only in the expected set.

In particular, the expected set is the sum of common models ($C$) and missing models ($M$) while the learnt set is the sum of command models ($C$) and extra models ($E$). Sensitivity ($S$) represents the ratio of common models in expected set, i.e. $S = \frac{C}{C+M}$ while Precision ($P$) represents the ratio of common models in learnt set, i.e. $P = \frac{C}{C+E}$. Usually, sensitivity and precision are combined into F-measure ($F$), which is defined as their harmonic mean $F = \frac{2}{1/S + 1/P}$.

Figure 55 shows the test results. We show the results of three different tests. The first one used an empty tree (i.e., if two models can link, they must have the same tag). The other two tests used the two trees discussed above. As we can see, using tag-trees has a higher success rate. Tree 1 is the one learnt from the port names, and Tree 2 is customized with more descriptions. Tree 2 has a better success with an average of 75%
for each criterion, which shows how the description improves semantic selection. The empty tree is always worse: it can only find around 40% of all the three metrics. A possible reason for this is that this tree is only based on a *keyword* search, in which two models that can be linked must have the same tag, without any semantic considerations. Using our tag-tree approach, since more semantic factors are considered, we can find more couplings (i.e., our approach will use tags that are semantically equivalent).

![Figure 55. Model composition success rates](image)

### 6.3.3 Fast Model Development Example

Let us assume that we want to build an occupancy model for a public service office. During the day, people come into the office at random, and there are different officers serving them throughout the day. We want to evaluate the work efficiency of officers and to improve their service. To do that, we want to know: 1) the *time* of the day, and 2) the total *number* of clients who have been served. If we search existing models for the keywords *time* and *number*, we obtain nothing. Instead, using PyCom with semantic model selection, we obtain results like Figure 56, which shows a part of Tree 2. In this tree, we can see that there are several sub-tag relations, such as *count* ≺ *number*, *clock* ≺ *time*, and *passenger* ≺ *customer* ≺ *user*.

Initially, we know that the occupancy model should have two output ports: one with tag *number* (number of clients who have been served) and one with tag *time* (the time of the day). By searching *time* as an output tag, PyCom suggests models with those
ports, and we can choose the one that meets our demands best. For instance, it can suggest model *Clock* since in the tag-tree there is *clock* ≺ *time*, which means that the output *clock* generated from the *Clock* model can be the output *time* for the target model (see Figure 57). *Clock* is an atomic model included in the *Alarm* model sample. *Clock* can take inputs of *hour* and *minute* and keep updating the time as a wall clock in the output *clock* port.

![Figure 56. A part of the learnt tag-tree ontology](image)

Similarly, when searching *number* as an output tag, since in the *count* ≺ *number*, PyCom will suggest model *Counter*, which is included in the *Garage Door* model. The model takes an input in *done*, and it *counts* how many inputs were received. PyCom can be used to suggest models that link to *Counter* as inputs. Since we have *finish* ≺ *done*, which means that the input *done* of the *Counter* model can be linked with the output *finish* of the *Processing* model. The *Processing* model is from the *Airport Boarding*
model and it can process an input user. After serving the user, it will generate an output finish. Similarly, since customer < user, PyCom can suggest Queue from the Panama Canal model for Processing, which takes a customer in a buffer. Then PyCom can suggest Generator from the Train model since there is passenger < customer, which can generate new passengers to be served.

Now, the model selection is done, we can link them as a new coupled model. This model will take the input of hour and minute, and generate the output of time and the number of users that have been served. Inside this model, it will generate clients, queue them in the reception, serve them and count the number that have been served.

As we can see, we have reused models from five different samples, and saved development time by selecting and reusing existing models. Please note that these suggested models can be reused fully or partially. This depends on the users’ responsibility to customize these models. In fact, they can do whatever they want to improve the logic inside each reused model. For example, the queuing logic in Queue may be FIFO; but in this case, we need a priority-based order. If so, we can change it accordingly. In addition, the serving time in the Processing model maybe dependent on the time of day, so we can add a new port time in the Processing model and link it as an output with the clock in the Clock model (the dashed line added in Figure 57). So far, the model is complete and we can assemble the five models together. We can also execute the assembled model in either standalone simulator (i.e. CD++) or on our Cloud-based Simulation (i.e. CloudRISE).

**6.4 Summary**

In this section, we discussed a new semantic selection method to supports self-learnt semantic for M&S resources. It can identify and select right resources based on their meanings. It allows MAMSaaS to recommend resources. This approach can help mashup engineers to select potential boxes during the wiring process. In addition, this is a general approach for all kinds of M&S resources. We also discuss an example applying this approach in semantic model selection. This example shows the effectiveness of our approach for selecting models using the tags and the leant tag-tree ontology.
Chapter 7: Case Studies: M&S Mashup Applications

In this chapter, we will show four M&S mashup cases. These cases will demonstrate how the MAMSaaS architecture and developed tools can simplify the development and integration of M&S applications. We will first show a general mashup for Cell-DEVS models; following by two more complicated cases (GIS landuse Mashup and Crowd M&S Mashup). Finally, we will show another mashup for DEVS model.

7.1 General Mashup for Cell-DEVS Models

In this section, we show a general mashup application for Cell-DEVS models. This mashup contains different M&S resources (e.g. models, simulation, visualization, information retrieving, and videos).

Cell-DEVS is a powerful approach to model complex scenarios, taking advantage of both Cellular Automata and DEVS formalism. Numerous Cell-DEVS models have been developed in different domains, such as real-time systems, networking and communications, biology and medicine, building architecture, defense and security, traffic, and environmental problems. Developing M&S application for Cell-DEVS involves different activities, e.g. developing the model, setting up the simulation environment, collecting data from other resources, and analyzing simulation results. As a result, it is hard to develop an M&S application.

In Section 5, we have discussed an example for building a Fire Spreading M&S mashup based on a Cell-DEVS model. That case study includes M&S resources such as fire spreading model, cloud-based simulation, result analysis and visualization, and an API for searching news. Here, we extend this mashup and make it more general for all Cell-DEVS models. This general mashup has additional boxes (e.g. Operator Box - get all files from model and Widget Box - YouTube). It works for general Cell-DEVS models and it simplifies the development process of Cell-DEVS mashups.

In particular, this general Cell-DEVS mashup includes:
• **Cell-DEVS model**: use existing Cell-DEVS models from different domains (e.g. fire spreading, snow flow, flower growing, and worm spreading).

• **Cloud-based simulation**: execute simulation experiments for Cell-DEVS models using CloudRISE.

• **Results analysis**: a function to analyze simulation results, e.g. getting simulation log from results.

• **Visualize results**: a widget to visualize simulation result.

• **WebAPIs**: an existing web APIs that is useful for the model (e.g., a web API to search the latest news of the model).

• **YouTube videos**: a widget to show some videos for the Cell-DEVS model.

• **Operators**: for simplifying the developing process (e.g. searching existing supported files or getting log file from simulation results).

We use CloudRISE to develop M&S resources as MSaaS (i.e. Cell-DEVS models and Cloud-based Simulation). First, CloudRISE provides Model as a Service for Cell-DEVS models. For example, a model developer can share the Fire Spreading model by a POST to `https://cloud-rise-instance/msaas/models/cell-devs/firespreading`; then the Cell-DEVS Model Box can GET it from this link. Second, CloudRISE provides Simulation as a Service for managing simulations. CloudRISE provides URL like `https://cloud-rise-instance/msaas/simulations/firespreading` and an EF description for each Cell-DEVS model. Similar to the one shown in Figure 23, this EF description contains the model name, context, input and output parameters, termination time and environments.

Here, the box **Cell-DEVS model simulation** follows the EF lifecycle to manage the experiment (recall Figure 18). In general, once it receives input events (i.e. a model URL and an initial state URL), this box will trigger its function. In this function, it will build a new experiment with the Experiment Description defined in Figure 58, using PUT to `.../msaas/simulations/firespreading/exp376` (376 is a random number). Note that the input parameter value will be replaced with the input value that the box receives. Finally, it will start the experiment, and the simulation results are sent to the output.

```xml
<Experiment>
  <Name>Evacuation experiment of fire spreading</Name>
```
an evacuation experiment of fire spreading for a Cell-DEVS model in a forest

.../simulation/firespreading

<EF>Environment group="1"/></EF>
<NumOfRuns>1</NumOfRuns>

<Inputs>
  <Parameter name="value" value="..."/>
</Inputs>

<Outputs>
  <Parameter name="zip results" value="..."/>
</Outputs>

Figure 58. Experiment Description for fire spreading model

We developed boxes for this mashup using the Box Development Tool. Most boxes are reused from the ones discussed in Chapter 5. In particular, the boxes for this case are as follows:

- **MSaaS Box** uses CloudRISE. It contains two boxes: *Cell-DEVS model* Box (a model as a service), and *Cell-DEVS model simulation* Box (a simulation experiment for the Cell-DEVS model). The models and their frameworks are available in CloudRISE (e.g. fire spreading, snow flower and worm spreading). Their box signatures are extracted from their corresponding configuration files in CloudRISE. Their box functions call corresponding MSaaS services in CloudRISE. Their views show their status.

- **WebAPI Box** calls existing open Web APIs. It contains *Web_search* (a REST WS to search the input from Google, Yahoo and Bing). This API is provided by Mashape (i.e. https://market.mashape.com/faroo/faroo-web-search). Using the Box development tool, we can extract its box signature from the above link; also, its function calls the Web API through HTTP request/response and the view shows its search result in the web browser.

- **Widget Box** shows input data in web browsers. Here we have two widgets: *Log Viewer* (a Cell-DEVS result web viewer) and *YouTube Videos* (a YouTube plug-in to show related videos). First, as discussed in Chapter 5, the log viewer is initially a
web-based tool to show two-dimensional Cell-DEVS simulation results. We reused and reconstructed this log viewer as a Box structure. It can receive inputs from URIs (e.g. model file, cell color file and simulation results). Then it can visualize the simulation results. Second, for the YouTube viewer, it reuses the YouTube web plug-in, and then it shows the videos related to the model in our lab channel https://www.YouTube.com/user/ARSLab.

- **Operator Box** handles inconsistencies between boxes. It contains *get all files from model* (getting existing supported files for given model) and *get log file from zip* (extracting simulation result log from a zip file). First, since the supported files (e.g. initial value, state color) of a Cell-DEVS are always in a same location within the model, the operator *get all files from model* gets the supported files based on the given input model. Second, simulation results are in a zip file. The operator *get log file from zip* extracts the zip file, and it outputs the log file as an URL.

![Figure 59. Uploaded boxes for general Cell-DEVS mashup](image)

After the boxes are developed, user can package them as zip files and upload them into the M&S Mashup Platform. Figure 59 shows the uploaded boxes in the platform. User can search/redeploy/un-deploy any boxes.

Now it is time to wire these boxes into a mashup. Figure 60 shows the box-wiring page in the M&S Mashup Platform. In the wiring editor, users can drag and drop boxes...
from different types, wire the boxes with their input and output ports. There is only one user input: *Input Box* (the MSaaS model link in CloudRISE). We can wire the output of *Input Box* and the input of Operator *get all files from model*, which will output different supported files such as *model* (the input model URL), *initialization* (the initial value *val* file for this model), and *pallet* (the state color pal file for the model). In addition, we can wire the outputs of *model* and *initialization* as inputs of *Cell-DEVS model simulation* in order to create a new Cell-DEVS experiment and run the simulation. After that, we can wire the output port of *Cell-DEVS model simulation* with the Operator Box *get log file from zip* to extract the log file and then wire its output *log* to the input *result* of *Log viewer* for visualizing purposes. In another path, the *Input Box* is connected to the WebAPI Box *Web Search* (to get the latest posts related to given model name) and Widget Box *YouTube* (to get related demos in our lab channel).

![Figure 60. Boxes wiring for general Cell-DEVS mashup](image)

This wiring process can also be simplified by using the semantic selection approach in MAMSaaS. Recall that there could be numerous boxes available, which make it complex for users to determine which boxes are “wirable”. To do so, we can use the semantic selection approach and its implemented tool PyCom described in Section 6. Each box is identified by its box signature so we can get its tags of its name/input/port. Then, since the tag-tree has been already learnt for the domain-specific M&S resources (which contains different MSaaS services, Web APIs and other M&S resources, as
discussed in Section 6), we can reuse this tag-tree here. Figure 61 shows parts of the tag-tree that are used in this Cell-DEVS mashup. In this tree, we can see that there are several sub-tag relations, such as \( \text{simulation} < \text{model} \), \( \text{initialization} < \text{value} \), and \( \text{pallet} < \text{color} \).

![Figure 61. Part of Tag-tree used in Cell-DEVS mashup](image)

Based on this tag-tree, given a box, PyCom can suggest “wirable” boxes (both predecessors and successors). For example, initially, a wiring engineer is not sure which boxes he/she will use in this Cell-DEVS mashup, he/she is only sure the Box \( \text{Cell-DEVS model simulation} \) (to run the Cloud-based Simulation experiment) is definitely needed. Box \( \text{Cell-DEVS model simulation} \) has two input ports: \( \text{model} \) (Cell-DEVS model URL) and \( \text{value} \) (initial state values), and one output port \( \text{zip results} \) (simulation results). By searching \( \text{model} \) and \( \text{value} \) as output tags, PyCom suggests boxes with those ports, and the wiring engineer can select the one that meets his/her demands best. For instance, it can suggest Box \( \text{get all files from model} \) since there is \( \text{initialization} < \text{value} \) and \( \text{model} = \text{model} \), which means that the output \( \text{get all files from model} \) are consumable by Box \( \text{Cell-DEVS model simulation} \). Similarly, when searching \( \text{zip result} \) as output port (since Box \( \text{get log file from zip} \) also has input port with \( \text{zip} \)), a \( \text{Cell-DEVS model simulation} \) can wire to this box. In addition, since \( \text{log} < \text{result} \), Box \( \text{get log file from zip} \) can link Box \( \text{Log viewer} \). Since \( \text{pallet} < \text{color} \), Box \( \text{get all files from model} \) can link to Box \( \text{Log viewer} \). In addition, since \( \text{model} < \text{word} \), Box \( \text{Input Box} \) can wire with both Box \( \text{Web Search} \) and Box \( \text{YouTube Videos} \).
Figure 62. Fire Spreading mashup using the general Cell-DEVS mashup

After the boxes have been wired, a new M&S mashup application is ready. Users can go back to the workspace and execute it. Figure 62 shows an example of a Fire Spreading Cell-DEVS mashup. After setting the model input (i.e. Fire Spreading Model URL), the M&S mashup is executed. Each box runs its function when receiving input events and it shows the corresponding view in the workspace. We can see that the MSaaS Box of the model simulation has been executed successfully, and a fire-spreading viewer with simulation results was shown with a timeline control (in which user can step forward and backward to predict the fire spreading). In addition, this mashup also shows the information of web search results. It queries the latest posts from Google, Yahoo and Bing (with the keyword “fire spreading”), and it lists title and links for these posts. Furthermore, it also shows related videos about “fire spreading” in the ARSlab YouTube Channel; a user can choose and watch any of them.
Figure 63. Snow growing mashup using the general Cell-DEVS mashup

Figure 64. Cancer propagation mashup using the general Cell-DEVS mashup
This general Cell-DEVS mashup can also be used in other Cell-DEVS models. Since the mashup is ready, a user only needs to provide the model URL. The mashup will follow its wirings to get its supporting files, run its simulation, and show the simulation results. For example, Figure 63 shows a mashup example for snow growing. We can see the snow-growing simulation has been successfully done because its simulation results are shown in the Log viewer. It also shows web search and videos for snow growing. Figure 64 shows another mashup example for cancer propagation. We can see the cancer model’s simulation has been successfully done because simulation results are shown in Log viewer. It also shows web search and videos for cancer.

7.2 A Mashup for Crowd Modeling and Building Information Management

In this section, we will show a complex mashup example for studying crowd behavior. It is for developing an application that can help the study of crowd behavior.

In recent years, the population in public areas and transportation facilities has become much denser. Numerous incidents and accidents with crowds have been recorded recently [Maj13], thus, predicting and trying to control the behavior of a crowd is an important issue. Modeling and Simulation (M&S) can support the analysis of such behavior, and it can help designers to evaluate the performance of their designs. Designers can use Crowd M&S to find flaws in particular areas of their buildings or urban area designs before the construction has begun [Hoo04].

Crowd modeling is complex. This complexity is related to the area where the crowds are studied (e.g. traffic intersections, shopping malls). To better study the behavior of a crowd in the design of a building or city section, we need to use a large amount of data (for instance, the layout information of a traffic intersection could be used as input for a crowd model). Nowadays, new technologies and tools are available to manage this kind of data. For instance, building designers use CAD (Computer-Aided Design) tools to improve the quality of building and urban designs, and BIM (Building Information Modeling) tools to enhance CAD’s databases for managing the buildings’ data [Ham08]. These tools’ data can be used as crowd model inputs (e.g., the layout of
the building or the coordinates of objects in a facility under study), and the tools can be used for 3D visualization, taking advantage of advanced visualization of the crowd simulation results [Alh06].

At present, crowd M&S lacks integration with advanced CAD/BIM tools. This case study has allowed us to develop a crowd M&S mashup integrating crowd M&S with CAD/BIM tools and visualization, as well as other related resources.

This mashup is originally developed by Stuti Rastogi, who was an undergraduate student in the ARSlab. It only took her around one month for developing this mashup. It took much less than using traditional development processes, which usually need months or years. Similarly, more students in the M&S graduate course of Dr. Gabriel Wainer are participating in developing mashups. The long term plan is to maintain a box/mashup library, so students can share their works and collaborate to develop mashups.

In particular, this mashup includes:

- **Crowd modeling**: a Cell-DEVS model to simulate crowd evacuation scenarios.
- **Crowd data collection**: a function for generating initial data files from CAD/BIM file to be used as inputs to the Cell-DEVS model.
- **Crowd model simulation**: for executing simulation experiments in the Cloud.
- **Results visualization**: a widget to visualize the simulation result.
- **Flickr pictures**: used as model’s state during the visualization.
- **YouTube videos**: videos related to crowd M&S.
- **Publications**: existing publications related to crowd M&S.

The Box Development Tool can help users develop boxes for the above M&S resources. The developed boxes for this case study are as follows:

- **MSaaS** uses CloudRISE. They contain *Crowd model* (a model as a service for the crowd behavior model), *Crowd model simulation* (a simulation experiment for this model), and *Crowd data collection* (a function experiment for collecting GIS data as model input). For each MSaaS Box, its signature is extracted from the corresponding configuration files; its function combines multiple MSaaS in a same experiment framework; and its view shows its execution status.
• **WebAPI** calls existing open Web APIs. They contain *Flickr keyword search* (a Flickr REST API for searching images of people) and *Publication search* (a REST API to query related publications of given model). For each WebAPI Box, its signature is extracted from the existing WS description file or HTML files; the function calls the Web API through HTTP request/response; and the view shows its output message.

• **Widget** shows input data in web browsers. They contain *Input IFC Box* (a CAD/BIM file for a multi-floor building), *Input DSM Box* (a domain-specific model for building layout extraction), *Log Viewer for Pedestrian*, and *YouTube videos* (a widget to show related videos of the given model). For each Widget Box, its signature is provided by users; the function analyzes input data; and the view visualizes input data.

• **Operator Box** handles inconsistencies between boxes. It contains *Log from Zip Box* (extracting the log from simulation results) and *fetch model name from URL* (extracting model name from given URL).

The **Crowd model** in MSaaS is a Cell-DEVS model to determine the evacuation time and the occupancy level of a building under emergency conditions. The building has multiple floors connected by stairwells. There are exit doors on the first floor. In the case of emergencies, people try to evacuate along the pathway on each floor, moving down towards the exits. Each cell contains the information of direction to the exit. Each person occupies a cell. The behavior of this person depends on the cell state (presented in Table 6). Besides walls (state 1) and exits (state 2), cells can be either occupied (e.g. state 4, 6, 12) or unoccupied (e.g. state 3, 5, 13).

<table>
<thead>
<tr>
<th>State</th>
<th>Color</th>
<th>Name</th>
<th>State</th>
<th>Color</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wall</td>
<td>8</td>
<td>8</td>
<td>N</td>
<td>Occupied</td>
</tr>
<tr>
<td>2</td>
<td>Exit</td>
<td>9</td>
<td>9</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>S</td>
<td>10</td>
<td>10</td>
<td>W</td>
<td>Occupied</td>
</tr>
<tr>
<td>4</td>
<td>S Occupied</td>
<td>11</td>
<td>11</td>
<td>Top of Stairs</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>12</td>
<td>12</td>
<td>Top Occupied</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>E Occupied</td>
<td>13</td>
<td>13</td>
<td>Bottom of Stairs</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>N</td>
<td>14</td>
<td>14</td>
<td>Bottom Occupied</td>
<td></td>
</tr>
</tbody>
</table>
The simulation rules will be listed in descending order according to their priorities. To avoid collisions, people have different priorities to move (e.g. S>E>N>W), and only the person with the highest priority can move (note that the order of these priorities is just an example and they can be customized). The rules of this model can be categorized as follows [Wan12a]: a) Someone enters a cell; b) Someone moves out a cell; c) Someone moves from a cell to a stairwell; d) Someone enters a stairwell; e) Someone exits a stairwell. After the model is ready, it is deployed as a Model as a Service in the CloudRISE.

Figure 65. Wiring boxes of Crowd M&S mashup

After the boxes are ready, we need to wire them in Mashup Platform. After uploading these boxes and importing them in user’s workspace, we can start wiring them via their inputs and outputs (see Figure 65). First, the links of Input DSM Box and Input IFC Box wire to the corresponding inputs of Box Crowd data collection, then its outputs initialization and parameters wire the inputs value and variable of Box Crowd model simulation, respectively. We also wire the outputs of Crowd model simulation to the inputs of Operator Log from Zip with pictures, which later wire to Widget Log viewer. In order to get the image links of the cell states, we wire the output port crowdmodel to Operator Fetch model name from URL, then it links Box Flickr Keyword search, after
which its output wires the input port pictures of Log from Zip with pictures. In another
direction, the Crowd model wires both Box Publication Search and YouTube videos.

This wiring process can also be simplified by the semantic selection approach in
MAMSaaS. We can get “wirable” boxes by using this approach and the PyCom described
in Chapter 6. We can reuse the learnt tag-tree. Figure 66 shows parts of the tree. We can
see that there are several sub-tag relations, such as crowd ≺ devs ≺ model, initialization
≺ value, and flickr ≺ picture.

![Figure 66. Part of Tag-tree used in Crowd M&S mashup](image)

Based on this tag-tree, given a Box, PyCom can suggest “wirable” boxes. For
example, initially, a wiring engineer is not sure which boxes he/she will use in this
Crowd M&S mashup, he/she is only sure about Box Crowd model simulation (create and
run the Cloud-based Simulation experiment). This box has three input ports: model
(crowd model URL), value (initial state values) and variable (initial variables), as well as
two output ports: model (this crowd model), and zip results (simulation results). By
searching model, value and variable as output tags, PyCom suggests Boxes with those
ports. For instance, it can suggest Box Crowd Model Box since there is crowd ≺ model,
which means that the output of Crowd Model Box can be consumed by Box Crowd model
simulation; also, it can suggest Box Crowd Data Collection since there is initialization ≺
value, and parameter ≺ variable. Similarly, since zip = zip and model = model, Box
Crowd model simulation can wire to both Box Log from Zip with picture and Box Fetch
model name from URL. Since flickr ≺ picture, Box Fetch keyword search can wire to Box Log from Zip with picture. Since log ≺ result, Box Log from Zip with picture can wire to Box Log viewer for picture. Since crowd ≺ word, Box Crowd Model Box can wire to both Box Publication Search and Box YouTube videos.

When the wiring is done, the mashup for this crowd M&S mashup is ready; a user can input the URIs of crowd model, IFC building file, and DSM model to the input boxes. After that, the mashup will do everything automatically (see Figure 67). It extracts initial data for the simulation, creates new experiment and executes it for the simulation and visualizes it using images obtained based on the model name. Thus by entering only the initial files, the user gets a proper visualization with functionalities to play, pause, step and randomly view the simulation in 3D. In addition, we can see the searched Flickr images for cell states, the YouTube videos and the publication lists that relate to crowd M&S. Please note that most of the boxes are reusable in other mashups. For example, the log viewer has been developed in a way to handle both 2D and 3D Cell-DEVS models. Similarly, the Flickr box can be re-used by searching new images for the model.

Figure 67. Crowd M&S mashup
Now, a complete application is available for studying crowds under emergency scenarios. The designers can run this mashup to evaluate design alternatives. For instance, a designer might want to test the evacuation time with different occupancy levels. To do so, they can export the IFC files from their designs and run the mashups with different settings. Figure 68 shows an example. Designers obtained different results for design alternatives (original design, relocating the doors, or adding more stairs). The idea was to reproduce the mashup to extract floor data, run the simulation in the cloud, and visualize the results obtained. As shown in Figure 68, when the occupancy level increases, the corresponding evacuation needs more time with a roughly linear manner. However, the relocating design shows a least increasing rate. Therefore, this mashup can tell the designers that the relocating design is the best alternative in terms of reducing the evacuation time.

**Figure 68. Building Occupancy vs. Evacuation Time, adapted from [Wan12a]**

### 7.3 A Land Use with GIS M&S Application

In this section, we will show another complex GIS M&S mashup. The case study focuses on building a Land Use M&S mashup using GIS and other related M&S resources. GIS allows managing, analyzing, and displaying geographically referenced information, and it has been studied for several years in M&S. However, developing M&S applications using GIS is still a complex process [Wan14a] because GIS M&S requires many M&S related resources.
Changes for Land Use have drawn much attention in urban planning, engineering, urban economics, and related fields. Land use can also affect the development of transportation, population, and land distribution. Our mashup application includes:

- **Environmental model**: a Cell-DEVS environmental model to simulate the land use scenarios.
- **Data collection**: a function for generating initial data files from GIS. These files can be used as inputs to the Cell-DEVS model.
- **Cloud-based Simulation**: simulation experiments in the Cloud.
- **Results analysis**: a function to analyze simulation results, e.g. parsing, converting, statistical analysis.
- **Visualize results**: a widget in web browsers to visualize the simulation result.
- **WebAPIs**: existing web APIs for the GIS M&S. For instances, to better predict the landuse trend, a web API to forecast weather for the studied GIS area is needed; to know the zip code information, a web API to search zip code based on geographic information is helpful.

We use CloudRISE to develop M&S resources as MSaaS. Model developers can share the GIS Landuse model by POST to https://cloud-rise-instance/msaas/models/cell-devs/landuse. Second, CloudRISE provides the URL https://cloud-rise-instance/msaas/simulations/landuse for this model. The Landuse Model Simulation Box uses this MSaaS to start simulation experiment for the Cell-DEVS model (recall Figure 18). Similarly, the Data collection and results analysis functions use a similar method to start their functions and get results.

The Box Development Tool can help users develop boxes. The boxes for this case study are as follows:

- **MSaaS Boxes** use CloudRISE. They contain *landuse model simulation* (a simulation experiment for the landuse model), *GIS_data_collection* (a function experiment for collecting data), and *GIS_KML_analysis* (a function experiment to parse simulation results to a KML file). For each MSaaS Box, its signature is extracted from the corresponding configuration files; its function combines multiple MSaaS in a same experiment framework; and its view shows its execution status.
• **WebAPI Boxes** call existing open Web APIs. They contain `City_weather_forecast` (a SOAP WS to forecast weather) and `GeoIP_to_address` (a RESTful WS to get an address from a GIS). For each WebAPI Box, its signature is extracted from the existing WS description file or HTML files; the function calls the Web API through HTTP request/response; and the view shows its output message.

• **Widget Boxes** show input data in web browsers. They contain `Input_box` (allowing users to input messages), `Input_show` (showing input messages), `KML_viewer` (viewing KML files in Google Map), and `Wikipedia` (getting Wikipedia information). For each Widget Box, its signature is provided by users; the function analyzes input data; and the view visualizes input data.

• **Operator Boxes** handle inconsistencies between boxes. They contain `get log file from zip` (extracting Log files from an archive), and `Split_geo info` (splitting Geo information into coordinates). For each Operator Box, its signature is provided by users; and there is no view for operators.

Each box is packaged in a zip file. Figure 69 shows the list of developed boxes, which contains its signature (XML), function (JS) and view (HTML/CSS). Now we can import them into the Mashup Platform and add them in the GIS M&S mashup workspace.

![Figure 69. Boxes developed for GIS M&S mashup](image)

130
After the boxes are ready, it is time to wire these boxes into a mashup. Figure 70 shows the box-wiring page in the M&S Mashup Platform. There are three user inputs: `model` (the land use model in CloudRISE); `GIS Tiff` (the user-selected area in a GIS dataset) and `Geo Info` (the global geographical references of GIS dataset). We wire the output of `GIS Tiff` as input of `GIS Data collection`, to send `GIS Tiff` and extract an initialization file for the `Land Use Simulation`. Similarly, we wire `model` and `GIS data collection` to `Land Use Simulation` to receive inputs of the model and initial files. After that, we wire the `Land Use Simulation` with the Operator Box `get log file from zip` to extract the log file and then wire this to `Results Parsing`, so a KML file is generated, which wires to the Widget Box `Google Map` for visualizing purposes. In another path, the `Geo Info` wires to Operator Box `Split Geo Info` to get the coordinates of the area under study, then the coordinates link to the WebAPI Box `get IP to Address` to get `zipcode` and `city address`. After that, the `zipcode` wires to WebAPI Box `City weather` for getting the weather forecast (shown in the Widget Box `ShowInput`); and `city address` wires to Widget Box `Wikipedia` to get wiki information.

![Figure 70. Boxes wiring for GIS M&S mashup](image)

This wiring process can also be simplified by the semantic selection approach in MAMSaaS. We can get “wirable” boxes by using this approach and its implemented tool.
PyCom in Chapter 6. Since the tag-tree has been already learnt for the domain-specific M&S resources, we can reuse this existing tag-tree. Figure 71 shows parts of the tag-tree that is used in this Cell-DEVS mashup. In this tree, we can see that there are several sub-tag relations, such as `landuse < devs < model`, `zipcode < city`, and `lat < latitude`.

Based on this tag-tree and available boxes, given a Box, PyCom can suggest “wirable” boxes (both predecessors and successors). For example, initially, a wiring engineer is not sure which boxes he/she will use in this GIS M&S mashup, he/she is only sure about Box `Cell-DEVS Model simulation` (create and run the Cloud-based landuse simulation). This box has two input ports: `model` (GIS model URL), and `value` (initial state values), as well as one output port `zip results` (simulation results). By searching `model`, and `value` as output tags, PyCom suggests Box `Input Landuse Model` since there is `landuse < model`, which means that the output of `Input Landuse Model` can be consumable by Box `Cell-DEVS Model simulation`; also, it can suggest Box `GIS Data Collection` since there is `initialization < value`. Similarly, since `log < result`, Box `Get log file from zip` can wire to Box `GIS KML parser`. Since `kml < visualization`, Box `GIS KML parser` can wire to Box `Google Map Visualization`. In another direction, since `lat < latitude` and `long < longitude`, Box `Split Geo Info` can wire to Box `Geo IP to Address`. Since `zipcode < word`, Box `Geo IP to Address` can wire to Box `City Weather forecast`. Since `city < word`, Box `Geo IP to Address` can wire to Box `Wikipedia`.

![Figure 71. Part of Tag-tree used in GIS M&S mashup](image_url)
Note that PyCom can suggest most of boxes that the wiring engineer needs, but may not cover them. It is the wiring engineer’s responsibility to refine the wiring process. For instance, there is no weather <text> in the learnt tree, so the wiring of Box City weather forecast and Box show input cannot be suggested by this approach.

After the boxes have been wired, a new M&S mashup is ready. After setting three inputs (i.e. Model URL; GIS Tiff URL; and Geo infor URL), the M&S mashup runs. Figure 72 shows its execution view. We can see that the MSaaS Boxes of data collection, model simulation, and results parsing have been executed successfully, and a KML with simulation results was generated and shown in the Google Map with a timeline control (in which user can choose time to predict the land use population). In addition, this mashup also shows the information of city address, weather forecast in following five days and Wikipedia with North Carolina.

![Figure 72. Executing the GIS M&S mashup application](image)

### 7.4 General Mashup for DEVS Models

The previous sections show the use of MAMSaaS in developing Cell-DEVS M&S mashups. Nevertheless, MAMSaaS is a general approach; CloudRISE can also support traditional DEVS models. In this section, we are going to discuss its use for developing DEVS M&S mashup.
We use a simple DEVS model for simulating customer reception in a barbershop. This model covers basic logic of queering and processing. There are limited barbers. Customers are served with a First-Come-First-Server order. Customer need to wait for his/her turn if all barbers are busy. However, to develop a rich M&S application for this barber model, many resources are required. This Barber DEVS mashup includes:

- **Barber model**: a DEVS model to simulate the barbershop scenarios.
- **Cloud-based Simulation**: simulation experiments in the Cloud.
- **Results analysis**: an operator to get each file from the simulation results.
- **Results showing**: a widget in web browsers to visualize the simulation result. DEVS models have simulation log file (with all details) and simulation out file (with all output messages).
- **Wikipedia**: to get more information of the barber service.
- **Web Search**: existing articles that are related to the barber service.
- **Publications**: existing publications that are used for the barber model.

The Box Development Tool can help users develop boxes for the above M&S resources. The developed boxes are as follows:

- **MSaaS Box** uses CloudRISE. It contains two boxes: *Barber model* Box (a model as a service provided by CloudRISE), and *DEVS simulation* Box (a simulation experiment for the DEVS model). Their box signatures are extracted from corresponding configuration files in CloudRISE; their box functions call corresponding MSaaS services (i.e. model as a service, simulation as a service) in CloudRISE; and their views show their status (i.e. execution status of experiment).

- **WebAPI Box** calls existing open Web API. It contains *Web_search* (a REST WS to search the input from web pages like Google, Yahoo and Bing). The Box development tool can extract its box signature from the above HTML page; also, its function calls the Web API through HTTP request/response; and the view shows its output search result in the web browser.

- **Widget Box** shows input data in web browsers. Here we have two widgets: *Wikipedia* (a Wikipedia search in web browser) and *YouTube* viewer (a YouTube plug-in to show related videos). First, the Wikipedia is for showing more
information of given input keyword, which has been developed in previous GIS M&S mashup. Second, for the YouTube viewer, it reuses the YouTube web plug-in, then searches and shows the videos that are related to the given model in our lab channel https://www.YouTube.com/user/ARSLab.

- **Operator Box** handles inconsistencies between boxes. It contains `getFilesFromZip` (getting each file out of given zip file) and `getModelName` (extracting model name from a given model URL).

![Diagram](image)

**Figure 73. Boxes wiring for Barber DEVS M&S mashup**

After the boxes are developed, users can upload them into the M&S Mashup Platform. Now it is time to wire these boxes into a mashup. Figure 73 shows the box-wiring page in the M&S Mashup Platform. There is one user input: `model` (the barber DEVS model link in CloudRISE). We can wire the output of `DEVS model simulation`, which will start a new experiment of the Barber model. Then the simulation result zip URL wires Operator `getFilesFromZip`, which will output each file of `log` and `out`. In addition, the model also wires the `Web Search`, `Publication Search` and Operator `getModelName`. `getModelName` will extract the model name “Barber” out of the model URL and wires it to Wikipedia.
After the boxes have been wired, a new M&S mashup application is ready. Users can execute it. After entering the Barber model URL, the mashup will run. Each box runs its function when receiving input events and shows the corresponding view in the workspace. Figure 74 shows its execution view in Google Chrome. We can see that the results of simulation out and log are shown. In addition, the publications related to the Barber model, wiki information and more web information are also displayed.

![Image of Barber M&S mashup execution](image)

**Figure 74. Executing the Barber M&S mashup application**

### 7.5 Summary

In this chapter, we presented different M&S mashup examples, i.e. General Cell-DEVS M&S mashup, GIS landuse M&S mashup, Crowd M&S mashup, and DEVS M&S mashup. These mashups show how to use the MAMSaaS architecture and tools for developing and integrating M&S applications. From these mashups, we can see that the proposed method can create and run simulation mashup in an easy and rapid way. Anyone with basic web development knowledge (like HTML/CSS/JS) can easily develop boxes. For building M&S mashups, users can drag and drop boxes and they do not need to have any specific web knowledge. In addition, since mashups can be shared as public URIs, so users can visualize and run the mashup without any specific knowledge. In
addition, we also discussed how to use the proposed semantic selection method in these case studies. We can see that it can recommend “wirable” boxes during the box wiring process.
Chapter 8: Conclusion

This chapter summarizes this thesis and suggests future research directions. Section 8.1 summarizes the major objectives and contributions of this thesis. Section 8.2 suggests different research directions.

8.1 Thesis Summary

Nowadays, M&S related technologies and applications have widely used in almost every aspect of life. Developing M&S application becomes more and more complex due to the varied M&S resources (such as systems, models, simulators, experiments, database and related functions), environment configuration (such as maintaining simulation environment for different model experiments) and increasingly people (such as the people who are involved in M&S activates but do not have expertise). With the fast development of information and technology, people start using new technologies like web services and cloud computing in M&S. New approaches have been proposed, including Web-based Simulation (WBS, which exposes M&S functions as web services) and Cloud-based Simulation (CBS, which integrates WBS and Cloud Computing). However, it is still a complicated process to develop M&S applications. These new technologies in M&S systems bring new issues, and it is still difficult to develop, deploy and integrate M&S applications in practice. This kind of complexity mainly reflects on the following issues:

1. It is hard to develop web services for varied M&S resources, due to the Web framework constraints and the variety of M&S resources.
2. It is hard to deploy M&S resources in the Cloud, since current solutions are not practical for reproducing and configuring M&S experiments.
3. It is hard to integrate with varied M&S services, since no efforts have been made to easily integrate, execute and visualize M&S services and existing WebAPIs.
4. It is hard to select resources and services based on their meaning; although an ontology layer can be used, it is always predefined and limited to specific rules.
The primary goal of this research was to simplify the development and integration of M&S applications using web technologies. To do so, we proposed the Mashup Architecture with Modeling and Simulation as a Service (termed MAMSaaS). MAMSaaS is a layered and lightweight M&S application development approach. It has five layers, which are Cloud Layer, Box Layer, Wiring Layer, Mashup Layer, and Tag Ontology Layer. It has a simplified life cycle to develop, deploy, identify, select, integrate and execute varied M&S resources as services in the Cloud.

The reasons that the MAMSaaS is able to simplify M&S application development and integration, as well as solving the above issues are as follows:

1) In order to develop varied M&S resources as service, MAMSaaS uses RESTful WS as the WS framework in its Cloud Layer. REST is directly built upon the Web and it takes full advantages of the Web. MAMSaaS uses RESTful WS to expose every M&S resource as a unique URL that can be operated on using HTTP methods. MAMSaaS uses the concept of Modeling and Simulation as a Service (MSaaS) in the Cloud. MAMSaaS implemented a MSaaS middleware, named CloudRISE, which allows users to develop M&S resources as services in the Cloud.

2) In order to deploy M&S services on the Cloud, MAMSaaS uses a new method using an Experimental Framework concept to simplify the deployment of environments. MAMSaaS uses Cloud computing technology to store all resources and experiments. MAMSaaS developed a lightweight Experimental Framework Template to control the life cycle of experiments. Likewise, MAMSaaS introduced an MS-Image concept, which includes cloud middleware, simulation and function environment. MS-images allow users to configure environments easily.

3) In order to integrate available M&S services and existing WebAPIs, MAMSaaS introduced a new method (termed M&S Mashup). Mashup has never been used to simplify the process of building M&S applications. This novel method is a lightweight M&S application development technique. M&S resources are grouped into MSaaSs, WebAPIs, Widgets and Operators, and they are packaged as a Box a signature, function and view. MAMSaaS implemented a Mashup platform to
develop and identify M&S mashup components as well as link and execute mashups for quick M&S application development.

4) In order to simplify the selection process of M&S resources, MAMSaaS introduced a new semantic selection approach with data mining and machine learning algorithms. The general idea was to automatically mine tags from user-interested resources, learn the tag-tree ontology from the tags, then select appropriate resources based on their tags and the tag-tree ontology. This semantic selection method does not need a predefined ontology, and it increases the automation of M&S application development.

The general idea of MAMSaaS is as follows: it can develop varied M&S resources as REST services, deploy these services in the Cloud, integrate varied services as mashups, run and show mashups results to users. In addition, it is also able to suggest resources during the integration process based on their meanings. This thesis focuses on our thoughts on MAMSaaS with methodologies and implementations, as well as M&S mashups applications adopting this architecture.

The main contributions of the MAMSaaS architecture are as follows:

- MAMSaaS is a layered mashup architecture to simplify the development and integration of M&S applications. The MAMSaaS architecture has a lightweight development process of M&S applications to develop, deploy, identify, select, integrate and execute varied M&S resources as services in the Cloud.

- MAMSaaS supports an experimental-oriented framework for configuring environments and reproducing experiments (for either simulations or supported functions). The EF helps users document the conditions of models and experiments (e.g. context, inputs/outputs, and environment). CloudRISE uses the EF Template to manage this EF information and control the lifecycle of experiments.

- MAMSaaS develops IaaS APIs to provide a scalable infrastructure for M&S resources. CloudRISE uses these IaaS APIs to save M&S resources in Storage Units and reproduce experiments in Compute Units. In particular, the MS-Image in the Cloud Layer can scale up/down environments for experiments.
• MAMSaaS uses MSaaS to make everything as a service. CloudRISE allows users to deploy any resource related to M&S as a service. These resources are decomposed into hierarchical URIs. They exchange information through XML via HTTP methods, e.g. its models/data/simulations/functions branches support Model as a Service, Data as a Service, EF as a Service, and Function as a Service respectively.

• MAMSaaS introduces a new M&S Mashup development process. M&S resources are grouped into MSaaSs, WebAPIs, Widgets and Operators, and they are packaged as Box with their signature, function and view.

• MAMSaaS introduces algorithms of adding semantic to M&S resources and learning domain specific ontology from them. MAMSaaS proposes a uniform way of identifying all kinds of resources. The learning algorithm can learn ontology just from these resources, instead of requiring experts to pre-define the ontology (which is time-consuming and may not cover all information used in these MSaaS).

• MAMSaaS introduces semantic selection and recommendation of M&S resources. During the mashup development, it promotes a “dataflow” fashion to suggest “wirable” boxes for the users. Given a box, MAMSaaS can recommend users with boxes that can be its predecessors or successors. It can further simplify the development of M&S application.

• MAMSaaS promotes collaborative development among people from different disciplines (e.g., Cloud Administrator, Box Providers, Ontology Manager, Wiring Engineer, and Client). Each person can focus on a specific activity in the M&S application development process. In addition, they can hide technical details in boxes and reuse them in mashups in the Mashup Platform.

• MAMSaaS improves the reuse of legacy systems in M&S. All the legacy systems can be deployed as MSaaS. They can be stored and executed in the Cloud. In addition, users can develop boxes/mashups for the legacy systems, and share them to others.

• MAMSaaS improves interoperability in M&S. The Mashup Platform supports syntactic interoperability by box development and wiring. The semantic selection method supports semantic interoperability by suggesting box “wirable” boxes for the users.
• MAMSaaS implements different tools, including CloudRISE middleware on AWS, box development tool, mashup platform, and semantic selection tool. These tools are available and open for reuse.

• MAMSaaS provides different mashup case studies to illustrate its usage in the development of M&S mashup, including Crowd M&S Mashups, GIS Landuse M&S Mashups, and general DEV5/Cell-DEV5 M&S Mashups. All the boxes and mashups are reusable.

### 8.2 Future Work

The following list includes a number of topics for future research:

• **To add more non-DEVS MSaaS services**

  CloudRISE implements the MSaaS concept. It allows users to develop their own M&S resources as services in the Cloud. The current implementation of CloudRISE mainly focuses on DEVS-related resources, such as DEVS/Cell-DEV5 models, different versions of CD++ simulators, and other related resources. One future direction is to extend this work with non-DEVS resources, such as other model formalisms and their simulators (e.g. Modelica), big data from other datasets (e.g. http://aws.amazon.com/datasets/), and different data analysis tools (e.g. Apache MapReduce). These non-DEVS resources can be added in any branch of CloudRISE, such as /data, /model, /simulations, /functions.

• **To switch MS-Image to Docker container**

  The current MS-Image is a VM-like approach. It simplifies the deployment of environments for simulation. Recently, another popular approach is to deploy this kind of image using Docker container. VMs are based on hypervisor; each VM has a full OS with its own memory management. Docker containers are executed with the Docker engine rather than the hypervisor. Containers share the host’s kernel. They are smaller than VMs and they are faster to start up. Containers have better performance, less isolation and greater compatibility. In current implementation, MS-Image contains MSaaS
Middleware, simulator and function environments; the VM approach is enough for our need since it does not have huge data right now. In the future, if more resources are added, it is better switch to Docker container for better performance.

• To develop more M&S mashups

Another future direction is to develop more M&S mashup applications using the proposed architecture and tools. We can ask help from graduate students in the M&S course of Prof. Wainer. For example, one requirement of student’s term project is to provide box signature for the project, so a box repository can be developed and shared among these students. They can reuse existing boxes or develop their own ones. They can also rewire, run and visualize boxes in their workspaces. Mashup hides the technical details of different M&S resources. In this way, they can focus on more about the real work that they are studying. In addition, the mashup platform can be extended to support more types of datasets and integrated with other systems (e.g. Internet of Things, Human-In-The-Loop). Therefore, this platform can further simplify M&S operations and improve user experience.

• To extend tag-tree ontology learning algorithm

In current tag-tree ontology learning algorithm, there are only “sub-tree” relations between tags. However, the tag-tree could be complicated with more relations and constrains. For instance, one future work is to improve the learning algorithm by analyzing other semantic relations of tags (e.g. junction/disjunction, business modeling process relations), and ontology experts can improve the tag-trees with specific constraints. Furthermore, if the resources are very close, our approach may suggest users with more recommendations. In that case, users are responsible for deciding what to use, since this thesis focuses on recommending resources to users instead of making decision for them. This issue could be handled if users provide more accurate resource signatures or descriptions; so close models can have different tags and they can be suggested more accurately.
• **To integrate semantic selection in mashup platform**

PyCom implements the semantic selection method, which can learn a tag-tree ontology and recommend resources based on their tags. The mashup platform is a tool to develop M&S mashup by managing and wiring boxes. In the current implementation, PyCom and the mashup platform are separate tools; the PyCom is not fully integrated into the mashup platform. One future work is to integrate the feature of semantic selection provided by PyCom into the mashup platform. Ideally, when a user adds a box, all potential “wirable” boxes (either its predecessors or its successors) should pop up and their potential “wirable” ports should be highlighted, so the user can select the one that meets his/her demands very easily.
Appendix A: Alternative Integration Approach using Workflow

The objective of this thesis is to simplify the development and integration of M&S application. In previous chapters, we have introduced the MAMSaaS architecture to develop M&S resources as boxes and integrate them as mashups. Besides these efforts, we also did a workflow approach to link resources directly.

In general, this workflow-based approach is an alternative to the M&S mashup approach in MAMSaaS. It provides a different way to develop and integrate the M&S application. The main differences of this workflow approach compared to the M&S mashup approach in MAMSaaS are as follows:

1. MAMSaaS integrates resources at web service level. If the web service is ready, we put it in box; otherwise, we use CloudRISE to develop MSaaS for it, and then put it in box. In contrast, the workflow only uses SimaaS. Functions are deployed in the Cloud, and workflow calls them directly by system-level commands.
2. MAMSaaS provides a view for each box, which supports rich visualization. In contrast, the workflow only focuses on the integration of components; it does not provide vivid visualization of each component during run-time.
3. MAMSaaS integrates all kinds of M&S resources, providing easy ways to develop and integrate run-time mashups. In contrast, the components in workflows are a subset of resources in mashups. It includes SimaaS and functions components, but not Widgets, WebAPIs and Operators.

Now, we will discuss how to build such workflow, by using the crowd M&S example shown in Section 7.2. This workflow integrates different activities (e.g. data collection, cloud-based simulation, and visualizing results).

Assume a construction project needs to study a crowd scenario. In particular, the building designers of a public area (e.g., a railway station) want to evaluate the crowd behavior during peak hours. In the area under study, the flow of the crowd is bi-directional: people tend to move forward, but they can change direction if blocked, and they can sidestep to avoid collisions [Hoo04]. Let us also assume that the building
designers have used BIM tools for their design, and they want to integrate the crowd model to improve decision-making. They want to use the BIM floor plan as model inputs, and use the BIM tools to visualize results. Nevertheless, there is no well-established method to integrate, formalize and automate all these activities. In order to deal with these issues, we defined the following workflow architecture.

![Three-layer Workflow Architecture for Crowd M&S](image)

**Figure 75. Three-layer Workflow Architecture for Crowd M&S**

The architecture presented in Figure 75 focuses on integrating different aspects of the crowd M&S with CAD/BIM tools. It is organized in three layers:

1) **The Component Layer** is responsible for deploying components in the Cloud. We need components for each activity required in the crowd M&S process, specifically:

   a) **Cell-DEVS modeling:** it is used for building crowd models using Cell-DEVS theory. Currently various Cell-DEVS crowd models are available (e.g. bi-directional, unidirectional, multi-floor) and they can be expanded and modified easily.
b) **Data Collection:** it is used to extract data from the designs. It can retrieve information from CAD/BIM tools to a Domain Specific Model (DSM) instance. Then, this instance can generate inputs for the crowd model.

c) **SimaaS:** it is used to run crowd simulation experiments using web services. We use CloudRISE SimaaS for building and executing crowd simulations in the Cloud.

d) **3D Visualization:** it is responsible for visualizing results. We defined a mechanism to parse the simulation results and visualize them directly in CAD/BIM tools. This visualization component can be easily customized.

These components are independent from each other, and they use well-defined interfaces to inter-connect. Cloud server saves components. It supports necessary hardware dependencies, allowing them to be ready-to-use. In Section 7.3, we have discussed in general how to develop these components. We can reuse these components in this workflow approach, only one difference is that we do not need to deploy all of them as services and develop them as boxes (instead, we call them directly using system-level commands).

2) **The Composition Layer** is responsible for defining workflows to formalize the integration of components and automate its execution. Users can easily replace components and modify the workflow for their particular purposes. In addition, these workflows can be stored in a workflow repository. Therefore, users can share and reuse them.

3) **The Application Layer** is responsible for collaboration in a simulation-based design process. It allows designers to build crowd M&S easily. The designers do not need to care about the workflow details: they choose a workflow and provide a CAD/BIM design as input. Then, they wait for the end of the workflow and visualize the results. In addition, they can evaluate their design and restart the process.

The components here have two categories: *crowd simulation services* (e.g. SimaaS in CloudRISE) and *crowd M&S tools* (e.g. BIM, CD++ and 3ds Max). The crowd simulation services are accessible via the Web. A workflow can invoke these simulation
services directly. For example, SimaaS in CloudRISE use RESTful WSs. We use the Taverna RESTful WS module to call these services. On the other hand, the crowd M&S tools provide particular functions. Component providers save them in the Cloud.

The composition layer defines workflows of the integration process. Figure 76 illustrates its overall structure.

![Figure 76. The Composition Layer in the Cloud](image)

- **Component Deployment**: it is responsible for deploying components in the Cloud. A component provides its functionality as an interface (a simulation service, a tool, a script, etc.). Its provider uploads and deploys it to the Cloud.

- **Workflow Engine**: it is responsible for defining workflows. A workflow engineer is in charge of building workflows by linking components. Here, we have used the Taverna, which supports RESTful WSs. We integrated Taverna and RISE.

- **Workflow Repository**: it is responsible for sharing the workflows in the Cloud. Building designers can reuse a workflow, specify inputs and execute it using a workflow engine. In our case, we used myExperiment [Gob07] as the workflow repository.

Based on these ideas, we can build a workflow to execute the Crowd M&S. Figure 77 shows the overall view of the workflow. In general, it takes the inputs of *crowdModel* (the cell-DEVS crowd model), *DSMmodel* (the DSM model including the particular information for the crowd simulation), the *IFCfile* (the building information of the studied area), and a Framework (a string name to describe the new crowd simulation EF
in CloudRISE). The output of this workflow is the \textit{parsedLogFile} (the parsed simulation results that can be visualized in 3ds Max). The designer can simply provide these inputs, and run this workflow using the Taverna Engine, which will run the workflow and generate the simulation results. After these are ready, the designer can visualize simulation results in 3ds Max. We can share the workflow on myExperiment, e.g. the workflow can be found at \url{http://www.myexperiment.org/workflows/3960.html}.

![Overall workflow in Taverna](image)

\textbf{Figure 77. Overall workflow in Taverna}

In this workflow, the input parameter \textit{crowdModel} records the model’s location (e.g. a Model as a Service in CloudRISE). In order to integrate CAD/BIM with the crowd model, \textit{IFC2CDpp} is the crowd M&S tool to extract information as the model’s inputs: it takes the input parameters of \textit{DSMModel} and \textit{IFCfile}, and then collects input data. Then,
it generates the outputs of the DSM and the initial files (e.g., the layout file and the initial parameters) as model inputs. After that, it is combined with the crowdModel as inputs for another two tools (XML Configuration and Files2Zip). XML Configuration is a tool to generate the XML configuration file for the simulation, and Files2Zip is a tool for compressing the files as input to the simulation framework.

At this point, we have the crowd model and its inputs ready. We now need to define a new simulation experiment to start the simulation. The designer can reuse the SimaaS services of CloudRISE. However, using SimaaS services still needs several steps (similar as in M&S mashup, they are grouped in an MSaaS Box). For example, the designer needs to choose the right URL and HTTP methods to start an experiment, upload the initialized files, execute the simulation and get the simulation results. To automate this process, we provide a sub-task workflow defined in Taverna (Figure 78).

![Sub-task workflow for Cloud-based Simulation using SimaaS](image.png)

**Figure 78. Sub-task workflow for Cloud-based Simulation using SimaaS**
In Figure 78, CreateFramework helps us create a new framework for the crowd model. It uses Taverna RESTful WS, which PUTs the XML configuration file to http://www.amazon-ec2ip.com:8080/cdpp/sim/workspaces/Tom/cdppv1/{framework}. The \{framework\} in this URL takes the input parameter of framework. Next, SendZipModelFile POSTs the zip file generated from Files2Zip. Now, the simulation is ready to start. StartSimulation PUTs to the ../simulation URL and it starts the simulation. Then, WaitSimulation checks the simulation execution status using GET on URL ../simulation. When the simulation finishes, the results are available in a zip file, and we use GetResults (GET to the URL ../results). Then, the extractLOGFile extracts the simulation results and passes it to 3DVizParser, which parsers the simulation results to be visualized in the 3D visualization. Using this workflow, the designer only needs three inputs: the emergency model file, the initial files generated, and an XML to configure the SimaaS. This workflow is http://www.myexperiment.org/workflows/2873.html.

\begin{itemize}
  \item \textbf{a)} realistic models
  \item \textbf{b)} arrow models
  \item \textbf{c)} crowd in stairs
  \item \textbf{d)} crowd in different floors
\end{itemize}

\textbf{Figure 79. 3D visualization results in different perspectives}

After this, the designer can use the advanced 3D Visualization tool [Fre13] to see the results in Autodesk 3ds Max. The designer can reuse the simulation results parser
discussed in [Fre13] to get the information (e.g. time, position, direction and occupation of each cell). Then, the designer can load the parsed results in 3ds Max. Figure 79 shows different perspectives: e.g., a) a realistic visualization in the main exit; b) an arrow visualization on the top of building; c) crowd movement in a stairwell; and d) crowd movement in different floors. The 3D Visualization tool also allows the designer to filter floors and individuals for better tractability. The demo of this case study can be found at http://www.YouTube.com/watch?v=u5idq-PDLck.

When the workflow is ready, workflow engineers can run it to automate the whole process. A designer can now download this crowd simulation workflow, specify the workflow inputs, and execute the workflow in a Taverna workflow engine in the Cloud. For instance, they can reuse the workflow described in Figure 78. A designer can pass inputs (i.e., the crowd model, the DSM model, the IFC build file, and the experiment name .../evacuation). Then, the designer can run this workflow using the Taverna Engine in the Cloud. Next, the workflow can collect the information, run the simulation and parse the results. After that, the designer can visualize the simulation results in Autodesk 3ds Max (as shown in Figure 79).

To sum up, in order to integrate different M&S resources and automate their execution, besides the M&S mashup approach proposed in MAMSaaS, we also did another effort using workflow. In this section, we introduced this workflow-based approach, which has three layers: a component layer (to provide different components of the crowd M&S process), a composition layer (to define workflows to integrate those components), and an application layer (to allow users to build crowd M&S applications easily). In addition, we took the example of crowd M&S to show its usage. From the example we can see, this approach can work as an alternative to the M&S mashup approach.
References


