chapter seventeen

Web-based simulation using Cell-DEVS modeling and GIS visualization

Sixuan Wang and Gabriel Wainer

Contents

Introduction

System engineering is an interdisciplinary engineering that consists of an integrated, life-cycle-balanced set of system solutions that satisfy customer requirements (ANSI/EIA-632 1999). System engineering is the art and science of creating optimal solutions to complex issues and problems, focusing on designing and managing complex engineering projects over their life cycles, and handling their work processes, optimization methods, and risk management tools (Hitchins 2008). On the one hand, system engineering allows the collaborative development that unifies all the contributors into one team, following a structured development process that transforms needs into a set of system product descriptions, generates information for planners, and provides input for the next level of development (Leonard 1999). On the other hand, system engineering facilitates the design phase of projects with a vast amount of data and variables, aiming at the integration of all aspects of the system into a whole.

System engineering encourages the use of modeling and simulation (M&S) to validate assumptions on systems, handle the interactions within them, and manage their complexity (Sage and Olson 2001). M&S has been playing increasingly important roles for analyzing and designing complex systems in system engineering. A model is a physical, mathematical, or logical abstract representation of a system entity, while a simulation is the implementation of a model over time that brings the corresponding model to life (Leonard 2001). It is useful for testing and analyzing the system design before the real project has begun. The other benefits of using

M&S are as follows: it is cheaper and safer than the real case; it is more realistic than traditional experiments; and it is faster than doing it in the real time (Tolk 2010).

Especially in the design phase of the life cycle of a system engineering project, M&S can be used to evaluate the performances of a new product concept, verify design specifications, or suggest improvements for a product. This kind of simulation-based design enables designers in different fields to test whether design specifications are met. It can provide the designer with immediate feedback of different design alternatives and facilitate decision making for optimal performance (Sinha et al. 2001).

For simulation-based design, modeling languages and simulators must take into account the special characteristics of the design process. The modeling language should allow models to be updated and reused easily, and simulators should be well integrated with the design tools for collaborative developments of designers. Among different M&S approaches, discrete event systems specifications (DEVSs) (Zeigler et al. 2000) and Cell-DEVS (Wainer 2009) are two of the most well-defined formal M&S methodologies. DEVS responses to external events based on continuous timing, composed of behavioral (atomic) and structural (coupled) components. Cell-DEVS extends DEVS to the field of Cellular Automata (CA), allowing modeling complex spatial problems.

In recent years, web-based simulation has received increasing attention in the simulation community, which has resulted in the birth of the area of web-based simulation (WBS). WBS is the integration of the Web with the field of simulation, by invocating computer simulation services through the World Wide Web. It has drawn much attention in the simulation community and has been growing for recent years (Huang et al. 2005). It can be defined as the use of resources and technologies provided by the Web with interaction of client and server M&S tools, supported by a browser for graphical interface interaction (Bencomo 2004). We can gain numerous benefits in this way: (1) Users can reproduce the execution of simulation online easily instead of installing complicated configuration of required simulation software. (2) Users can reuse and share simulation resources on site without worrying about the capability of their local machine CPU or memory. (3) With the help of advanced distributed simulation technologies, the simulation can be executed on distributed computers via communication networks, which can further improve interoperability and speed up the execution time. (4) Simulation results can be retrieved or visualized by the emergency crews on site, which can crease the emergency response success.

This chapter presents an effort toward integrating four specific components: geographic information systems (GISs), modeling, simulation, and visualization, to support the simulation-based design process in system engineering. The focus of this chapter is to use WBS and GIS visualization techniques to deal with this kind of integration. The objective is to develop an integration method that enables designers, decision makers, and planners in multidisciplinary fields to easily run simulation models for testing the performance of design and visualize simulated results faster for making fast decision. The basic idea is to provide a general method to extract information from GIS, model with Cell-DEVS theory, run WBS, and visualize results back in Google Earth. To do so, we will discuss the challenges and address the advantages of this integration method, as well as different ways to realize it, and real application case studies.

A GIS is an environmental system that combines hardware, software, and data for managing, analyzing, and displaying all forms of geographically referenced information. Currently, GIS is widely used to manage large spatial databases, to perform statistical analyses, and to produce effective visual data representations. GIS applications have been used to quickly and reliably process spatially referenced data as a decision support tool (Badard and Richard 2001).

The generation of M&S of environmental systems can be traced back to the early days of computer simulation (Botkin et al. 1972). These simulations have been combined with GIS (Band 1986; Desmet and Govers 1995; Van Derknijff et al. 2010; Wang 2005) since the 1980s. M&S in GIS is for characterizing and understanding environmental patterns and processes, and estimating the effects of environmental changes. Many GISs already contain embedded simulation capabilities; however, it usually focuses on specific simulation that is limited by the power of simulator and poor for scaling up. On the other hand, simulating advanced environmental simulation models separated from GIS tools is complex (Zapatero et al. 2011). Therefore, there are growing interests in the potential for integrating GIS technology and environmental simulation models. Several works have been done in the efforts to integrate GIS and DEVS M&S for environmental systems (Gimblett et al. 1995; Hu et al. 2011; Wainer 2006; Zapatero et al. 2011), trying to transform the GIS information into a DEVS/Cell-DEVS model and to visualize simulation results in Google Earth. Although designers use GIS, M&S, and visualization in various projects, the integrative uses between them are still at an elementary stage. Furthermore, most M&S methods and applications run on single-user workstations, which normally cost too much time for installation and configuration of all the software and dependencies needed by the simulation. It is better to have remote access to the simulation resources with Web service (WS) interfaces, improving data accessibility, interoperability, and user experience.

Visualizing large amounts of information interactively is one of the most useful capabilities of GIS. Ware (2000) points out that visualization helps to present mass data, identify patterns or the problems with data, and facilitate understanding of data. Visualizing data using the current computing technology and interactive GIS can create multiple perspectives, enhancing a designer's abilities to better understand the studying phenomenon. On the other hand, visualization of simulation results are usually accompanied by high-fidelity graphics or vivid animation, which can provide a number of benefits: to compare a simulated result to the expected effect, to provide interactive environment to verify models, and to easily refine their solutions with different scenarios. In system engineering, the visualization features can present data views of the present and future. Usually, the visualization provides a graphic user interface to support interactions between the system and the users.

The contributions of this chapter are mainly as follows: (1) It proposes a new integrated framework that combines GIS data collection, Cell-DEVS modeling, WBS, and GIS visualization, which allows users to choose the best available technologies to analyze GIS system behaviors and predict future scenarios. The parts in this framework are loosely coupled and easy to scale up. (2) It introduces details of modeling using Cell-DEVS formalism for analyzing a GIS system in the simulation-based design phase of system engineering. (3) It provides a prototype with different case studies that is implemented based on the RESTful simulation services middleware for WBS and GIS systems with Google Earth visualization. The simulation engines are stored on a server and can be run remotely using our RESTful Interoperability Simulation Environment (RISE) middleware. DEVS is a universal abstract formalism, separating M&S, and RISE middleware separates simulator implementation and underling hardware; therefore, they make it perfect for deployment online remotely.

The rest of the chapter is organized as follows. The Section "Related work" reviews the issues of current approaches for integrating Cell-DEVS modeling, Web-based simulation, and GIS visualization. Then, the Section "Architecture" presents a novel architecture to solve these issues, followed by details in the Sections "Cell-DEVS modeling," "GIS data collection," "Web-based simulation," and "Visualization in Google Earth." Finally, the Section "Applications" shows real cases to demonstrate the advantages of the proposed integration architecture.

Related work

In an integrated system of GIS, M&S, and visualization, each component contributes to the system with distinctive features. GIS provides the functions to manage spatial information between entities. M&S allows representing the dynamic relationships among studying entities and predicting the behaviors in the following period. WBS eases the implementation of simulation in a much easier way using web technologies. Visualization is to represent simulation results in an intuitive and vivid way for fast decision making. In this section, we are going to review the different integration aspects of these three components. First, we will review the M&S method architecture using DEVS in system engineering. We will not only discuss the advantages of using Cell-DEVS for modeling environmental system problems but also review the different categories of WBS for the model execution.

Integration of M&S using DEVS/Cell-DEVS

Many M&S methods can be used to simulate environmental GIS models. DEVS (Zeigler et al. 2000) is a mathematical formalism for specifying discrete events systems using a modular description. DEVS is a perfect option for M&S to system engineering due to many reasons. DEVS is very easy to reuse and integrate with other components. DEVS has been successfully used in this area due to its ease of modeling, the varied ways to combine existing models, and the efficiency of the simulation engines. Other benefits are coupling of components, hierarchical structure, and modularity construction.

Figure 17.1 shows the layered DEVS M&S architecture for system engineering. From bottom to top, it goes from infrastructure hardware, software implementation to conceptual abstraction. DEVS models are closed under coupling, which means a coupled model can be viewed as an atomic one, allowing reusing and integrating to other DEVS models without changes. Each model can be associated with an experimental framework, allowing the individual simulation executing and testing easier. Similarly, WBS, including simulators and supporting middleware, is also independent of the modeling framework, which allows different simulators of particular or customized purposes and a layered view of M&S.

DEVS models are composed of behavioral (atomic) and structural (coupled) components. Cell-DEVS (Wainer 2009) extends DEVS by supporting cellular models in a spatial lattice. Cell-DEVS defines a cell as a DEVS atomic model and a cell space as a coupled model. Each cell holds a state variable and a computing function that updates the cell state based on its present state and its neighborhoods. CD++ (Wainer 2002) is an open-source environment capable of executing DEVS and Cell-DEVS

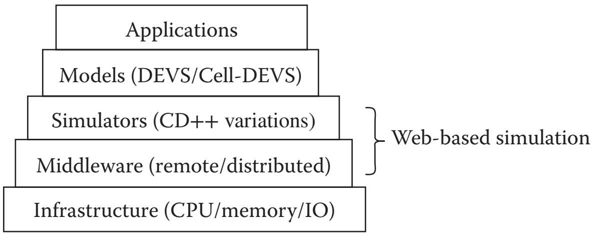

Applications
Models (DEVS/Cell-DEVS)
Simulators (CD++ variations)
Middleware (remote/distributed)
Infrastructure (CPU/memory/IO)
Web-based simulation

Figure 17.1 A layered DEVS M&S architecture for system engineering.

models, supporting different variations for stand-alone, parallel, or other improved simulators. RISE (Al-Zoubi and Wainer 2011) is a simulation middleware to support RESTful WSs for web-based CD++ simulation. Because DEVS strictly separates models from simulators, and WBS based on RISE Middleware strictly separates simulation from the supporting hardware, each part is loosely coupled and easy to scale up.

The Cell-DEVS formalism combines both CA and DEVS (Zeigler et al. 2000). Cell-DEVS has been widely used in many complex system engineering projects, not to mention GIS system. The advantages of using Cell-DEVS for GIS system are as follows: (1) the inheritance of DEVS, (2) the spatial rule-based features, (3) the event-driven asynchronous execution, and (4) the input or output (I/O) ports for easy integration with each other.

Categories of Web-based simulation

From previous discussion, we have known that WBS gains a lot of benefits for implementing the models: executing repeatedly without complicated installation, resources reuse on site regardless the local hardware constrains, and support for distributed simulation to speed up the execution time. The WBS is one of the focuses of this chapter; now let us see its main categories.

Early WBS efforts began in 1995, as old as the Web itself. However, the area of WBS is still in its infancy with many issues to study (Byrne et al. 2010); besides, the number of real applications and tools for WBS are still very small (Wiedemann 2001). Many authors classified WBS, like Byrne et al. (2010), Myers (2004), and Page (1999). The classification could be developed architecturally and summarized as the following four categories.

Local simulation

Local simulation is where the simulation engine is downloaded directly by the client to the user's local computer. The simulation engine executes the model completely in the client, usually with the capacity to visualize simulation results in a 2D or 3D way. Local simulation makes the server as a central distributor, but it does not do any real work (Bencomo 2004). The common approaches of using this local simulation are through Java applets or executable files. Usually, the user opens a browser and navigates a web page via a uniform resource identifier (URI), which contains an applet or executable file for downloading. After the downloading phase, the user can run the simulation engine within the applet or executable file. Figure 17.2 gives the basic local simulation architecture after this downloading phase. Originally, the simulation engine usually runs on a single local processor (LP). However, as the demand is for larger and more complex models, a single processor becomes very time-consuming. This results in the emergence of parallel simulation, distributing simulation over a set of LPs that are geographically close to each other (e.g., clusters),

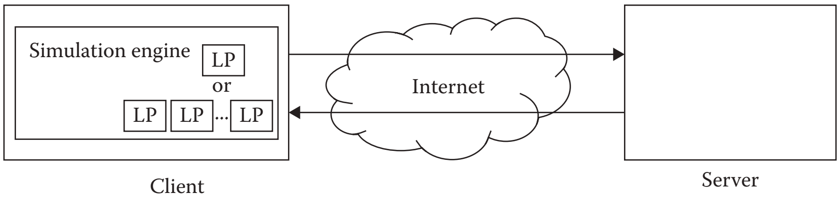


Figure 17.2 Basic local simulation architecture.

to reduce the simulation time (Page 1999). Parallel simulation also handles simulation protocols and synchronization management.

Remote simulation

In remote simulation, the simulation engine is located and executed remotely on the server side (Bencomo 2004). User accesses the simulation engines through a browser on the client side. Users can submit their requests (with specified message/parameters) to the simulation engine through the Web server, then simulation will run remotely, and results are returned to the user once the simulation has finished. The communication and manipulation between the client browser and the simulation engine in the server could be performed by many ways. Following the history of the development of the Web, the means include Common Gateway Interfaces, Java remote method invocation, Common Object Request Broker Architecture (CORBA), remote procedure call (RPC), and WSs (Bencomo 2004). The user opens a browser and navigates to a Web page via a specific URI, follows interface of the Web server operations, communicates to server with message protocol, and then waits for the response from the server on this URI.

In recent years, WSs are increasingly used for remote simulation, improving data accessibility, interoperability, and user experience. WSs are mainly categorized into two classes: RESTful WSs (Richardson and Ruby 2008) (to manipulate Extensible Markup Language [XML] representations of Web resources using a uniform set of *stateless* operations), and arbitrary WSs (Ribault and Wainer 2012) (such as Simple Object Access Protocol [SOAP]-based WS, in which the service exposes an arbitrary set of operations). RESTful WSs imitate the web interoperability style. The major RESTful WSs interoperability principles are universally accepted standards, resource-oriented, uniform channels, message-oriented, and implementation hiding. Figure 17.3 gives the basic RESTful remote simulation architecture. RESTful WSs expose all services as resources with uniform channels where messages are transferred between those resources through those uniform channels. We can access RESTful WS through web resources (URIs) and XML messages using hypertext transfer protocol (HTTP) methods (GET, PUT, POST, and DELETE). RESTful WS is simple, efficient, and scalable. Its strengths of

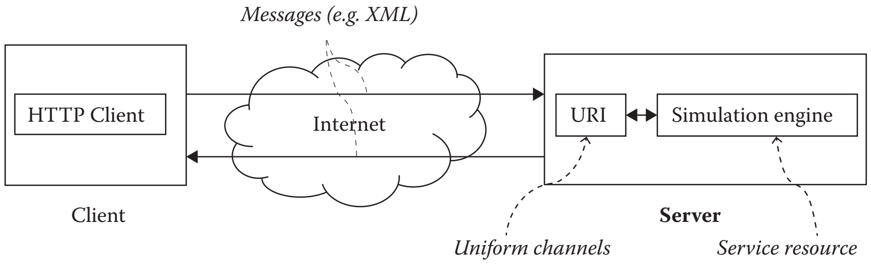


Figure 17.3 Basic remote simulation architecture with RESTful Web service.

simplicity, efficiency, and scalability make RESTful WS an excellent candidate to perform remote simulation. On the basis of these ideas, in Al-Zoubi and Wainer (2011), the authors presented the first existing RISE middleware. The main objective of RISE is to support interoperability and mash-up of distributed simulations regardless of the model formalisms, model languages, or simulation engines. The details of SOAP-based WS and the comparison between SOAP-based WS and RESTful-WS will be discussed later in Section "Distributed simulation."

Remote simulation enables larger simulations to be run on powerful computers in terms of processing, memory, and networks so that users can get the results from any low-end computer with a browser, or any other personal-aided devices (e.g., smartphone, tablet). By this way, thin client is achieved. Another advantage is that the user can reuse the simulation engine that is already available on the server, without worrying about simulation environment setup and other software dependencies issue. Besides, using a familiar interface that separates model and simulator, the user can focus on their model, making maintenance easier and promoting productivity. Some disadvantages of remote simulation are network latency, standardization, and dynamic interaction with simulation (Myers 2004).

Regarding whether the location of visualization/animation engine is on the server side, together with the simulation engine, remote simulation can further be classified into pure remote simulation and hybrid simulation (Byrne 2010).

Note that remote simulation does not specify the number of simulation engines or underling machines. Remote simulation focuses on the user's point of view, providing a way for user to access simulation services easily and efficiently. Conceptually, there is one simulation engine located on a single machine. However, if the simulation runs on various machines located remotely, the simulation is said to be distributed. This topic is discussed in the Section "Distributed simulation."

Distributed simulation

Distributed simulation is created to execute simulations on distributed computer systems (i.e., on multiple processors connected via communication networks) (Fujimoto 2000). Figure 17.4 shows the basic distributed simulation architecture. Its main objective is to interface different simulation resources, allowing synchronization for the same simulation run through different simulation engines across a distributed network, to interoperate heterogeneous simulators or geographically distributed models. The other benefits of using distributed simulation include model reuse, reducing execution time, interoperating different simulation toolkits, and providing fault tolerance and information hiding (Boer et al. 2009; Fujimoto 2000).

By its nature, all of WBS can be described as distributed simulation (Alfonseca et al. 2001; Page et al. 1998). Indeed, Page et al. (1998) classifies distributed simulation as a category of WBS. Web technologies and distributed simulation technologies have grown up largely independently, and influenced each other. The difference of distributed simulation with remote simulation is that distributed simulation aims to speed up simulation time by partitioning of models, focusing on the message transmission between simulation engines that are heterogeneously located and geographically distributed, whereas remote simulation is aiming to provide easy and efficient way for the user to access simulation services. Therefore, we separate them here into two categories for clearance.

The defense sector is one of the main users of distributed simulation technology, providing virtual distributed training environment, relying on the high level architecture (HLA) for simulation interoperability (Khul et al. 1999). Besides, Strassburger et al. (2008) predict the bright future of distributed simulation in the nonmilitary area in the gaming industry, the high-tech industry (e.g., auto, manufacturing, and working training), and emergency and security management. To make distributed simulation more attractive to the industrial community, Boer et al. (2009) suggested that we need a lightweight commercial-off-the-shelf product to interoperate different parts efficiently, effortlessly, and quickly. This demand leads the nonmilitary distributed simulation community reaching out to other

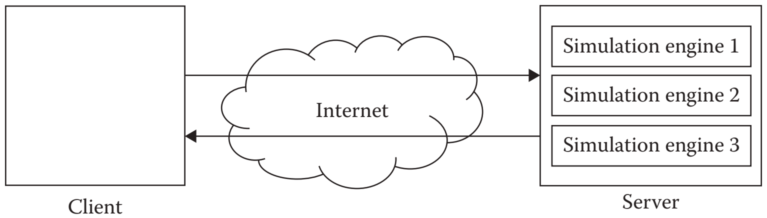


Figure 17.4 Basic distributed simulation architecture.

communication technologies in the Web to overcome HLA shortcomings (e.g., standards complexity, high dependency, and poor scalability). CORBA was used during the 1990s to interoperate heterogeneous simulations, using RPCs style. Those procedures glue different objects operations together, giving the impression that an operation invoked remotely as a local procedure call. The distributed simulation community has also turned to WSs since its birth in year 2000 to provide interoperability between geographically located simulation engines.

As discussed previously, the most widely adopted arbitrary WSs technology is SOAP-based. SOAP-based WSs (Papazoglou 2007) provides a similar way to CORBA RPC-style (see Figure 17.5). It exposes services that encapsulate various procedures on the server side. These services are addressed using URIs and described in XML Web Services Description Language (WSDL) documents. The client side can compile WSDL into procedures stubs. Assume two simulation engines want to interact with each other; the one that starts the request is in the role of the client, while the other that receives the message is the server, and vice versa. Note that these simulation engines will change the roles of client/server during their interactions. At runtime, the client converts the RPC into an SOAP message (XML-based), wraps the SOAP messages in an HTTP message, and POST (HTTP method) to the server. On receiving the HTTP message, the server will extract it reversely into the appropriate procedure call and respond to the client in the same way.

Another popular WS technique for distributed simulation is RESTful WS (Richardson and Ruby 2008), which exposes all services as resources with uniform channels, and messages are transferred between those resources through those uniform channels. Heterogeneous simulation engines can be located on different distributed machines. As shown in Figure 17.6, we can access RESTful WS through the main Web resource (URI) via XML messages

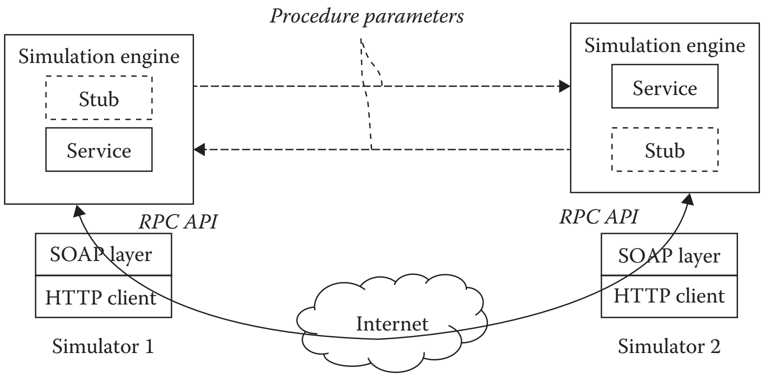


Figure 17.5 Distributed simulation SOAP-based Web service architecture.

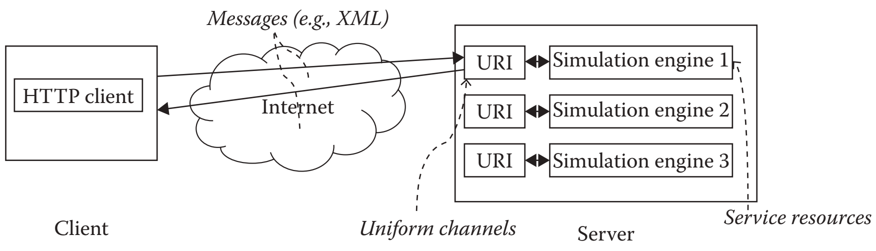


Figure 17.6 Distributed simulation RESTful Web service architecture.

using HTTP methods (GET, PUT, POST, and DELETE), and this main Web resource (URI) acts as manager to coordinate with other simulation engines though uniform channels by their URIs. After simulation finishes, the main Web resource collects all simulation results and forwards them to the client.

In Al-Zoubi and Wainer (2011), the authors identify the shortcomings of using SOAP-based WS comparing with RESTful WS: (1) SOAP-based WS communicates simulation information in the form of procedure parameters that are actually the application programming interface (API) of the simulation component, whereas REST defines them directly as XML message that hides internal implementation. (2) SOAP-based WS transmits all SOAP messages only using HTTP POST channel, while REST uses all HTTP channels for universal interfaces and clear manipulation semantics. (3) SOAP-based WS clients need to have a stub that needs to be written, integrated with existing software and compilers, whereas REST does not require this process. (4) SOAP-based WS groups all services as procedures and exposes them via a port, whereas REST exposes them as resources that are easier to manage and maintain. Therefore, we are using RESTful WS in RISE to interoperate heterogeneous simulators for distributed simulation.

Online model/documentation repository

Another type of Web-based simulation uses online repositories for models or related documentation. This kind of repository aims to use a server-based centralized repository that can be used to store retrieval simulation models (Miller et al. 2000) or online documentation to existing simulations (Narayanan 2000). User can share their work with others, just as a Web document that is hyperlinked to server, improving collaboration and cooperation. Besides, Ribault and Wainer (2012) provide a method using myExperiment (Goble et al. 2010), an online social networking environment, to find, share, and reuse workflows that formulate and automate simulation-based WSs.

Integration of GIS and M&S

Researchers have categorized the integration of GIS and simulation models as *loose* and *deep* coupling (Bell et al. 2000). The loose coupling category consists of most integration efforts through exchanging data files. This approach often needs human interaction, which hinders the automatic operation. The deep coupling approach links GIS and M&S with a friendly user interface, which has drawn much attention in recent years. Some GISs allow getting information from outside using public APIs to support this kind of deep coupling approach. Geographic Resources Analysis Support System (GRASS) (GRASS GIS 2013) is a popular GIS project of the Open Source Geospatial Foundation. GeoTIFF (OSGeo Foundation 2013) is an open standard to establish a TIFF-based interchange format for georeferenced raster images.

Many of the environmental simulation models use GISs (Gimblett et al. 1995; Hu et al. 2011; Wainer 2006b; Zapatero et al. 2011), which allows manipulating georeferenced information and performing different operations with maps (Longley et al. 2005). However, there is limited data sharing between these GIS system and M&S. Normally, the models for simulation do not use GIS data directly or save model output into a GIS database (Wang 2005). Besides, Ribault and Wainer (2012) state that few of the current solutions can distribute the varied simulation engines based on WBS, and formalize the workflow-like and scalable way to extract data and visualize the results on Google Earth.

GIS is usually organized in multiple data layers, centralizing all the environmental data available and making it accessible in several forms (maps, digital maps, or raw data files). Whatever format is chosen, it is necessary to transform GIS data into a format compatible with the simulation software. In particular, in Wainer (2006), we showed how to simulate environmental systems efficiently using DEVS and Cell-DEVS. Those studies used the CD++ M&S environment (Wainer 2009), and this software stack was recently expanded to allow GIS data to be transformed into CD++ simulation engine (Zapatero et al. 2011). This method relies on the GeoTIFF standard file format, which is supported by most GIS.

Integration of modeling and simulation and geographic information system visualization

We have discussed the benefits of visualization of GIS and simulation results in the Section "Introduction." Here, let us review some literatures of the integration efforts between simulation results and GIS visualization.

Combining the M&S and GIS visualization can significantly increase the representation and understanding of simulation results in an environmental project. This kind of integration goes through system engineering process from separated fields into a whole one. For example, Bishop and Gimblett (2000) present an example of prediction of visitor location and movement patterns in recreational areas. However, as noted by Wang (2005), the integration effort between M&S and GIS visualization is at a preliminary level. An operator often creates visualization and animations separately for simulation results of specific projects.

The limitations of GIS visualization are mainly the quality of presentation and the level of interaction/flexibility of the animation. Many studies have shown the effort to combine the GIS and visualization. Pullar and Tidey (2001) use a 3D GIS for visual impact assessment. Among these efforts, Google Earth (Google 2013) is one of the most popular and massively used for both scientific and generic purposes. Google Earth uses Keyhole Markup Language (KML) (OGC 2013), an XML-based language focused on geographic visualization that includes annotation of maps and images. Its file format can be used to display geographic data in Earth browsers (such as Google Earth), using a tag-based structure with nested elements and attributes. The geographic visualization needs to include not only the presentation of graphical data but also the control of the user's navigation. Once a KML file is created, it can be imported into Google Earth, allowing the visualization of the simulated results on a customized layer impressed over the standard layers (e.g., satellite views and street maps). Google Earth provides mechanisms to make layers evolve forward and backward in time, which is useful to analyze the progress of a simulation interactively (Google 2013). In Zapatero et al. (2011), we adapted Google Earth (Google 2013) as the geospatial visualization system. Here, we will expand our previous work with more advanced WBS techniques and advanced visualization methods of Google Earth.

Architecture

A simulation-based design system in GIS using system engineering enhances the collaboration among stakeholders and interoperability of multidisciplinary experts. This kind of system can facilitate agreement of different kinds of people on the proper alternatives. The integration of GIS, M&S, WBS, and visualization is expected to enhance the analyzing and evaluating process of such a GIS-based decision-making system. Figure 17.7 illustrates the general WBS architecture using Cell-DEVS and GIS. The basic idea is to get data from GIS, model with Cell-DEVS theory, run simulation remotely, and visualize simulation results in Google Earth. The overall approach includes the following four subsystems: Data Collection, Cell-DEVS modeling, WBS, and Visualization. The significance of this integration is to use the best available technologies to analyze GIS system behaviors and predict future scenarios.

1. Data Collection: It is done automatically, generating initial data files from GIS into the Cell-DEVS model. This includes a Dataset Reader for selecting georeferenced raster data from open standard

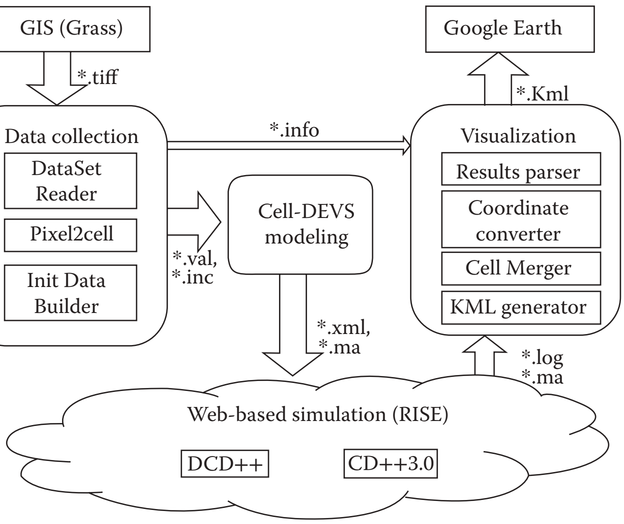


Figure 17.7 Web-based simulation architecture using Cell-DEVS and geographic information system.

GeoTIFF file, and a Pixel2Cell for approximating of collected data into the scale that can be used in Cell-DEVS. Then the Init Data Builder builds initial states of cells and necessary attributes for Cell-DEVS model. The Section "GIS data collection" discusses the details.

2. Cell-DEVS Modeling: It builds a Cell-DEVS environmental model according to the collected data. It defines the cell space size, neighborhood, and rules of the model behaviors using CD++ modeling tool. More details can be seen in the Section "Cell-DEVS modeling."
3. WBS: It submits Cell-DEVS models to RESTful simulation services URI, executes the simulation remotely, and then gets the simulation results. Different simulation engines with CD++ variations are stored on the server and can be run remotely using the RISE middleware. DEVS is a universal abstract formalism separating M&S, and RISE middleware separates the simulator implementation and underling hardware. The Section "Web-based simulation" gives more details.
4. Visualization: It visualizes the simulation results in Google Earth, providing an intuitive and interactive way for analysis. Once simulation results are retrieved from RISE, it parses the simulation log file, optimizes the cells with Cell Merger, and converts the coordinate system into the way used in Google Earth. Then it generates a KML file that can be imported into Google Earth, allowing the visualization

of the simulated results on a customized layer impressed over the standard layers. The Section "Applications" illustrates the details of this subsystem.

In the following sections, we will use a simple land use change example as a prototype to illustrate each step in the proposed WBS architecture. We will first show how to model it using Cell-DEVS, and then show the remaining steps in the process: GIS data collection, WBS, and Google Earth visualization for the model.

Cell-DEVS modeling

Simulation models in GIS are abstract representation for characterizing and understanding environmental patterns and processes. Recent attention has focused on the land surface process models for meteorological study. Changes for land use have drawn much attention in urban planning, engineering, geography, urban economics, and related fields. Some other networks, such as transportation, population, and soil distribution, can play a strong influence on land use pattern changes; on the other hand, new land use patterns can affect the developments of these networks. In this section, we will introduce the Cell-DEVS mechanism and CD++ tool, with a land use changes example to show how to model it.

The Cell-DEVS formalism (Wainer 2006) is defined as an extension to CA combined with DEVS (Zeigler et al. 2000), a formalism for specification of discrete-event models. In DEVS, atomic models describe model behaviors, specified as black boxes, and several DEVS models can integrate together forming coupled models (hierarchical structural models). Cell-DEVS defines a cell as a DEVS atomic model and a cell space as a coupled model. Each cell holds a state variable and a computing function that updates the cell state based on its present state and its neighborhoods. As DEVS models are closed under coupling, the independent and black box-like simulation mechanisms allow these models to communicate each other in single processor, parallel, or distributed simulators without too many changes, which makes it possible for WBS in terms of message transmission and synchronization.

CD++ (Wainer 2002) is a tool for the simulation of DEVS and Cell-DEVS models, and has been widely used to study a variety of models, including architecture, traffic, environmental, emergency, biological, and chemical. The behaviors of a Cell-DEVS atomic model are defined using a set of rules. Each rule indicates the future value of the cell's state if a precondition is satisfied. The precondition is usually checked around the neighborhood of the current cell; Figure 17.8 shows some of the most widely used neighborhoods. Moore's neighborhood contains the origin and its eight adjacent cells; Von Neumann's neighborhood includes the ones to the up, down, left, and right of center. The hexagonal's neighborhood is useful because of the

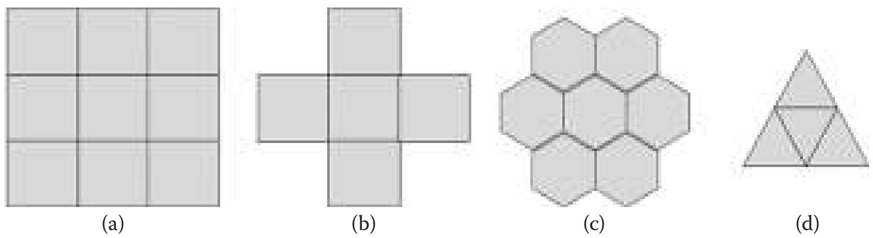


Figure 17.8 Widely used neighborhoods: (a) Moore, (b) Von Neumann, (c) hexagonal topology, and (d) triangular topology.

equivalent behavior in every direction, while the triangular neighborhood can cover more varied topology. The local computing function evaluates rules in order, until one of them is satisfied or no more rules. Each rule follows the form: VALUE DELAY {CONDITION}, which means when the CONDITION is satisfied, the state of the cell will change to the designated VALUE, and its output is DELAYed for the specified time.

To forecast land use changes, we use Cell-DEVS theory to model related behaviors. To simulate interaction between land use types and population, we put information into two layers: land use (retrieved from GIS data collection part) and population (predefined data from other system). Figure 17.9 shows the formal specification for this land use model in CD++ (Wang and Chen 2012).

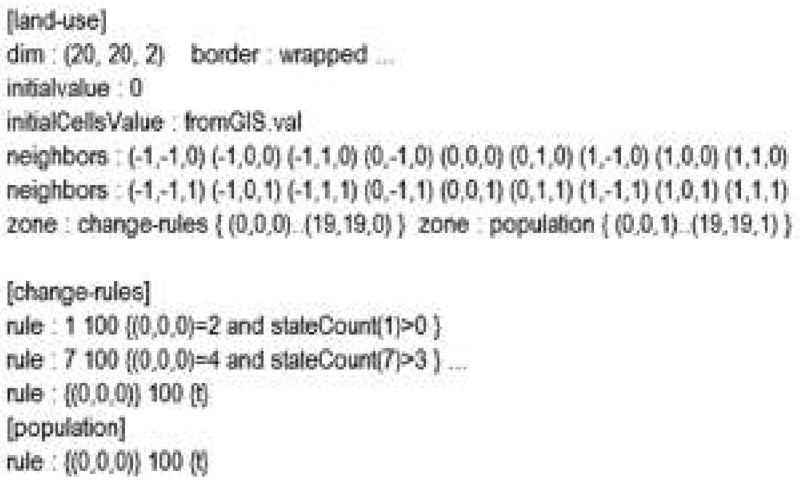
As we can see from Figure 17.9, it defines the size of the cell space (20 × 40), neighbors (extended Moore neighborhood with two layers), and the rules (simple local computing function). We can also notice that there are two zones each with specified rules. The basic idea of the rules is that as the time goes on, the land use pattern will increase/decrease of its intensity according to its neighbors. Take the first rule for example, it means whenever a cell state is 2 and more than one neighbor has the state value of 1, the cell state changes to 1. This state change spreads to the neighbors after 100 milliseconds.

GIS data collection

We have seen how to model using Cell-DEVS theory; now let us deal with how to get information from GIS system, and to link GIS with Cell-DEVS models. The subsystem of GIS Data Collection is responsible for extracting data from GIS and transforming it into inputs used in the Cell-DEVS model.

GIS and GeoTIFF

In Cell-DEVS, the model is always characterized in a specific spatial area that is composed of a set of cells. To integrate GIS and Cell-DEVS modeling, we need to build the initial files (e.g., the layout of studying area) for



```
[land-use]
dim : (20, 20, 2)    border : wrapped ...
initialvalue : 0
initialCellsValue : fromGIS.val
neighbors : (-1,-1,0) (-1,0,0) (-1,1,0) (0,-1,0) (0,0,0) (0,1,0) (1,-1,0) (1,0,0) (1,1,0)
neighbors : (-1,-1,1) (-1,0,1) (-1,1,1) (0,-1,1) (0,0,1) (0,1,1) (1,-1,1) (1,0,1) (1,1,1)
zone : change-rules { (0,0,0)..(19,19,0) }  zone : population { (0,0,1)..(19,19,1) }

[change-rules]
rule : 1 100 {(0,0,0)=2 and stateCount(1)>0 }
rule : 7 100 {(0,0,0)=4 and stateCount(7)>3 } ...
rule : {(0,0,0)} 100 {t}
[population]
rule : {(0,0,0)} 100 {t}
```

Figure 17.9 Land use change model in CD++. (Wang, Y. and Chen, P. 2012. *A Cell-DEVS Geographic Visualization Framework Using Google Earth, Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.)

Cell-DEVS based on the data extracted from GIS file. The general idea of this GIS Data Collection subsystem is that we are trying to map the geoinformation from real-world geo-ordinates to cell space ordinates, separates the whole region into cells, and stores them as the initial file (*.val file) for CD++. Besides, we also can get a property variable (*.inc) and an information file (*.info) where global geographical references are kept.

In GIS, a geographic dataset always includes a comprehensive collection of vector, or raster, or imagery data covering some parts on Earth. For vector dataset, it often includes hydrographic maps, geological maps, soils, administrative boundaries, and others; for raster dataset, it often includes elevation, slope, aspect, land use, and geology; for imagery dataset, it often includes 1 m resolution orthophoto, land scenes, and daily surface temperature time series (GRASS GIS 2013). GRASS is one of the most popular GIS and can handle with raster, topological vector, image processing, and graphic data. GRASS GIS use Geospatial Data Abstraction Library (GDAL) (GDAL 2013) for raster/vector import and export. Geographic information could be read from raster maps based on the data model of GDAL, such as coordinate system, affine geotransform, and raster band stored information.

Because of the popularity and wide use of GIS, to improve interoperability, a standard file format that is compatible and exchanges well with other different GIS formats is needed. We choose GeoTIFF as the standard file format, as it is supported by most GIS, including GRASS (OSGeo Foundation 2013).

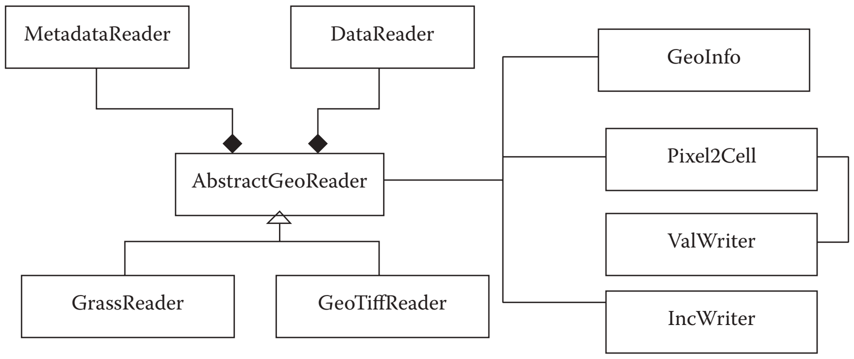


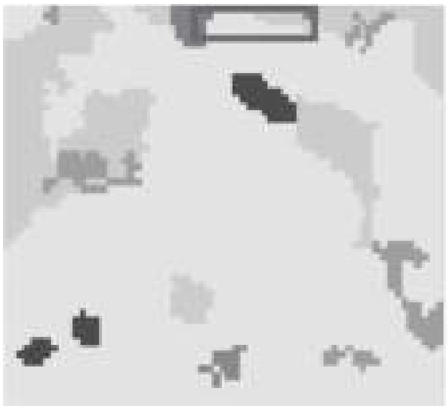
Figure 17.10 Class diagram of geographic information system data collection.

Class diagram

Figure 17.10 shows the implementation class diagram of this part, upgraded from Mariano (2011) and Wang and Chen (2012). AbstractGeoReader implements the general logic of the data input composing of two subclasses: MetadataReader (to obtain geographical references) and DataReader (to get data on each of the pixel). These classes provide an interface through abstract methods, and they can be extended to particular subclasses. Though other systems may have data in various formats, the main logic of this data collection for simulation model does not change. The GeoTiffReader class extends the AbstractGeoReader class, implementing these abstract methods to obtain information from georeferenced files in GeoTIFF format; the GrassReader class retrieves data through the GRASS API. GeoTiffReader implements these operations through the GDAL (GDAL 2013) library for raster geospatial data formats, with ability for efficient handling of large files. GeoInfo is a convenience class for storing geographic contextual data such as coordinates and the resolution of the area (*.info), which is used in visualization part for Google Earth. Pixel2Cell is to approximate the most common value covered in an area that contains multiple pixels, which contains ValWriter class for generating the cell's initial values file (*.val). IncWriter is for generating necessary attributes for Cell-DEVS models from data obtained.

Data collection process

As we have seen the general process and class diagram of this data collection process, now let us go deeper to see the three subprocesses depicted in Figure 17.7.



Color	Landuse type	State value
▨	*High intensity developed*	*1*
▨	*Low intensity developed*	*2*
▨	*Managed herbaceous cover*	*4*
▨	*Evergreen shrubland*	*7*
▨	*Southern yellow pine*	*15*
▨	*Mixed hardwoods/conifers*	*18*
▨	*Water bodies*	*20*

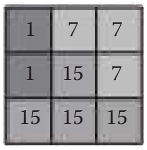
Figure 17.11 Studied area of *Landuse* map and corresponding states. (Wang, Y. and Chen, P. 2012. *A Cell-DEVS Geographic Visualization Framework Using Google Earth*, *Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.)

Step 1: Dataset Reading

Now we use a simple land use changes example to show how this Data Collection subsystem works. The first step is called Dataset Reading. We choose a sample raster dataset named North Carolina (NC, USA) in GeoTIFF format provided by GRASS. This dataset offers raster, vector, LiDAR, and satellite data. We choose the raster format in this GeoTIFF dataset that includes geographic layers of land use, elevation, slope, aspect, watershed basins, and geology. Raster dataset groups different layers into raster bands that contain common information. Each raster band contains the map size, GDAL data types, and a color table for mapping color and land use type values (see Figure 17.11). Each raster band consists of several blocks for efficient access chunk size in GDAL, and each block consists of several pixels. The following pseudo-code shows the algorithm to retrieve data from a raster dataset (Wang and Chen 2012). The general idea of this algorithm is to look through the studying dataset to get each pixel information.

```
For the given "landuse.tiff" dataset:
  For each raster band in this dataset:
    Get this raster band information: Xsize, Ysize, BlockSize.
      For each block in this band:
        Get block information: valid block size, stored block data;
        Output each pixel data in this block.
```

For the example shown in Figure 17.11, the land use of the studied area has one raster band that contains four blocks, separated in total by 179×165 pixels.



1	7	7
1	15	7
15	15	15

Figure 17.12 Approximate cell state value from multiple pixels.

Step 2: Pixel2Cell

The second step is Pixel2Cell, reading the block data of the raster band of *landuse.tiff*. We need to get the color value of each pixel and generate the initial value of the corresponding cell in Cell-DEVS. Ideally, we can match each pixel to an exact single cell in Cell-DEVS model. However, this is not always the case. In fact, the model behaviors in Cell-DEVS are relatively large (e.g., fire spread in a large forest); therefore, each cell size in Cell-DEVS usually covers multiple pixels that are scaled with minimum unit of a geographical map. Here we add an approximate method to solve this problem, implemented in Pixel2Cell. The idea is to maintain a queue for each cell, sort the pixels shown in this cell according to their colors, and choose the most common color in the queue as the representative value. Figure 17.12 shows an example for this case, in which the cells with state 15 appeared four times in total, more than the other cells (i.e., state 1 and state 7), so state 15 becomes the corresponding cell state value for the Cell-DEVS model.

Step 3: Init Data Building

The third step is Init Data Building. After approximating cell values from the pixels, for this land use change example, we get a Cell-DEVS model with the size of 20 × 20. We store them into an initial information file (*.val) for Cell-DEVS model, along with *.inc (necessary attributes for Cell-DEVS model) from data obtained and *.info (geographic contextual data such as coordinates and the resolution of the area).

Web-based simulation

So far, as we have modeled the rules and extracted the layout from GIS system, the Cell-DEVS land use change model is ready. Now we move to the next step of WBS to execute the model. We use RESTful WS to do so, implemented in RISE. RISE is accessed through web resources (URIs) and XML messages using HTTP methods. Implementations can be hidden in resources, which are represented only via URIs. Users can run multiple instances as needed, which are persistent and repeatable by specific URIs. The HTTP methods are typically four types: GET (to read a resource or get its status), PUT (to create or update a resource), POST (to append data to a resource), and DELETE (to remove a resource). An interface between RISE and different CD++ versions, DCD++ (Al-Zoubi and Wainer 2011) for distributed simulation or CD++ v3.0 (Lopez and Wainer 2004) for improved multiple state variables/ports version, allows running DEVS/Cell-DEVS distributed simulations. RISE API likes a classic website URL http://www.example.com/lopez/sim/workspaces, attached by the following services, and the full RISE design and API described in Al-Zoubi and Wainer (2011):

- ../{userworkspace} contains all simulation services for a given user.
- ../{userworkspace}/{servicetype} contains all frameworks for a given user and simulation engine type (e.g., DCD++ for distributed simulation or CD++ v3.0 for improved CD++ features).
- ../{userworkspace}/{servicetype}/{framework} allows interacting with a framework (including the simulation's initial files, configuration, and source code). The POST channel under this API is used to submit files; PUT is to create a framework or update simulation configuration settings; DELETE is to remove a framework; and GET is to retrieve a framework state.
- ../{userworkspace}/{servicetype}{framework}/simulation interacts with the simulator execution. By calling the Get channel, simulation dedicated to this framework will run. Since RISE supports different servicetypes (simulation engines), in DCD++, this URI is the modeler's single entry to a simulation experiment. It will initialize and communicate with other URIs (e.g., on different machines) to perform distributed simulation, handling synchronization messages.
- ../{userworkspace}/{servicetype}/{framework}/results contains the simulation outputs.
- ../{userworkspace}/{servicetype}/{framework}/debug contains the model-debugging files.

We have two types of servicetype of simulation engines (DCDpp and CD++ v3.0). Figure 17.13 shows an example of having multiple simulation engines at the same time. Requests to …/DCDpp/landuse are sent to *landuse* framework of DCDpp simulator, while requests to …/lopez/landuse are sent to *landuse* framework of CD++ v3.0 simulator.

For a specific simulation, we can realize the remote simulation by using these URIs with HTTP methods. For example, after getting initial model and configuration files for simulation, we use PUT to create a framework with the configuration file and POST to upload these initial model files to this framework. Then, this newly created simulation environment can be executed by using PUT to {framework}/simulation, then we wait for the simulation to finish and GET the simulation results files from {framework}/results.

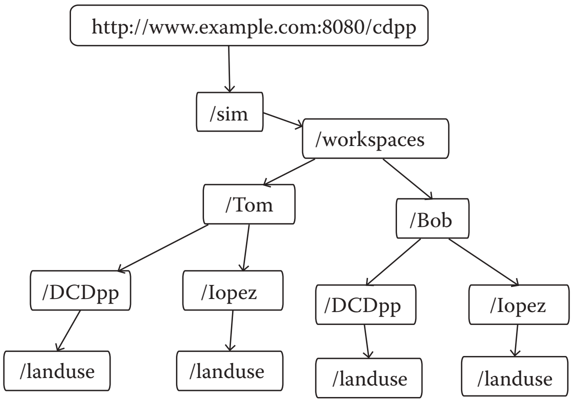


Figure 17.13 RESTful Interoperability Simulation Environment uniform resource identifier example with multiple simulation engines.

In each HTTP response (no matter from which URI), the Response Status is very informative. Normally, it means the request is successful if it returns 200 (OK) and 201 (Created); otherwise, some errors or unexpected exceptions would have happened, such as 400 (bad request), 401 (unauthorized), 403 (forbidden), 404 (not found), 406 (not acceptable), and 501 (not implemented).

In RISE, there are various available CD++ versions, including DCDpp for distributed simulation and CD++ v3.0 for improved CD++ version with different state variables/ports.

DCD++ simulator in RESTful Interoperability Simulation Environment

DCD++ simulator configuration

In RISE, different machines need to coordinate and exchange simulation HTTP messages to perform the distributed simulation. The simulation model is split and assigned into one of these machines. Each physical machine needs to have at least one instance of the RISE middleware (each contains one CD++ engine). DCD++ instances (CD++ Engine on different machines) act as peers to each other. Simulation manager takes responsibility to transmit XML messages and handle synchronization issues between these DCD++ instances, enabling each of them simulating its portion of the model (see Figure 17.14).

Concrete services (e.g., DCD++) are wrapped and accessed through URIs at the middleware level, allowing the middleware to be independent of any specific service. The middleware routes a received request to its appropriate destination resource and apply the required HTTP method on that resource. This middleware design allows additional services

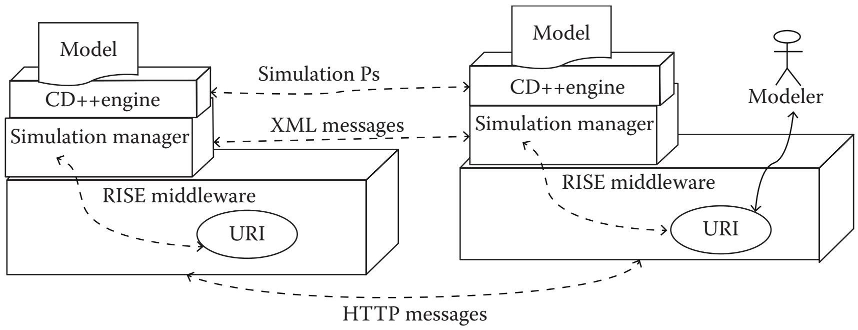


Figure 17.14 RESTful Interoperability Simulation Environment distributed simulation session. (From Al-Zoubi, K. and Wainer, G., *Intelligence-Based Systems Engineering*, 10, 129–157, 2011.)

(e.g., CD++ v3.0 for improved CD++ features) to be plugged into the middleware without affecting other existing services.

The DCD++ is constructed based on the partitioned model under simulation between different machines. The code below shows an example of a part of DCD++ XML configuration information for a Cell-DEVS model. This model-partitioning document describes each cells zone location. Each partition will run the simulation session in dedicated CD++ engine that is located in the belonging machine with the IP and port specified in this document. Note that DCD++ also supports the standard DEVS model, which allows the modeler to customize the partition to an atomic model or coupled model.

```
<ConfigFramework>
   <Doc> This model Simulates Life using Cell-Devs. </Doc>
   <Files>
      <File ftype = "ma">life.ma</File> …
   </Files> …
   <DCDpp>
      <Servers>
         <Server IP = "10.0.40.162" PORT = "8080">
             <Zone>fire (0,0)..(14,29)</Zone>
         </Server>
      </Servers>
      <Servers>
         <Server IP = "10.0.40.175" PORT = "8080">
             <Zone>fire (15,0)..(29,29)</Zone>
         </Server>
   </Servers>
   </DCDpp>
</ConfigFramework>
```

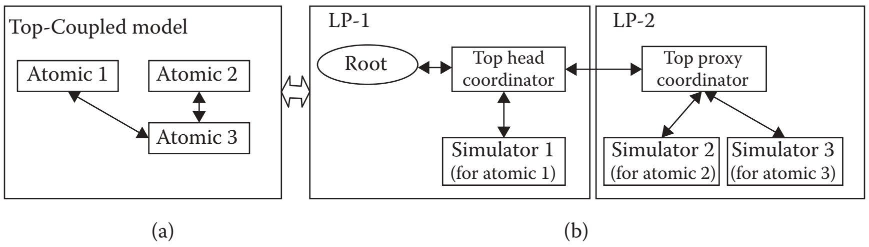


Figure 17.15 Head/Proxy Modeling Structure. (a) Coupled model defined. (b) Model hierarchy during simulation. (Al-Zoubi, K. and Wainer, G., Performing distributed simulation with RESTful Web-services, *Proceedings of the 2009 Winter Simulation Conference*, Austin, TX, © 2009 IEEE.)

DCD++ simulation synchronization algorithm

We have discussed that how to configure the model partitions into different DCD++ engines that are located on different machines; it is the time to introduce the synchronization algorithms of DCD++. Figure 17.15a gives a simplified DEVS coupled model that consists of three atomic models. Now suppose that this model is partitioned between two LPs. DCD++ would simulate this model hierarchy as shown in Figure 17.5b (Al-Zoubi and Wainer 2009). The simulator processor simulates an atomic model (e.g., Simulator 1 for Atomic 1 and Simulator 2 for Atomic 2), and the coordinator processor simulates a coupled model. DCD++ uses head/proxy structure to reduce the number of remote messages. In this case, messages exchanged between Simulator 2 and Simulator 3 are handled locally by the proxy coordinator that is responsible for grouping messages, communicating with the Head coordinator if necessary.

Note that this example is a simplified version, but it also shows sufficiently the way DEVS/Cell-DEVS model assigned to this head/proxy structure. Thanks to the coupling closure characteristic of DEVS models, each atomic model can be expended into a complicated coupled model. Also in Cell-DEVS model, each partition zone can be viewed as a coupled model and each cell can be viewed as an atomic model.

Simulation messages (shown in Figure 17.16) for synchronization among processors hierarchy can be grouped into two categories: (1) Content messages represent events generated by a model. There are External messages (X) and Output messages (Y). (2) Synchronization messages for forward simulation phase and time advancing. There are Initialize message (I) to start the initialization phase, Internal message (*) to start the transition phase, Collect message (@) to start collection phase, and Done message (D) to mark the end of a simulation phase.

DCD++ LPs follow the conservative algorithm approach, which guarantees the safe timestamp ordered and always satisfied the local causality constraint. The simulation cycles in phases where LPs are synchronized

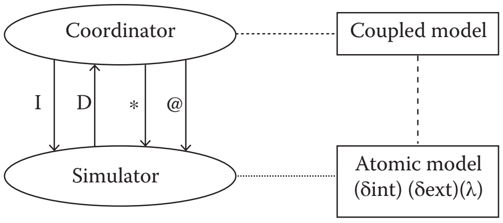


Figure 17.16 Message exchange in a simulation cycle. (From Al-Zoubi, K. and Wainer, G., *Intelligence-Based Systems Engineering*, 10, 129–157, 2011.)

at the beginning of each phase. Each LP (which is a CD++ engine) has its own unprocessed event queue. In this case, after initialization of every atomic model, the Root Coordinator (see Figure 17.17) starts the collection phase by passing a simulation message (@) to the topmost Coordinator (including Top Head/Proxy Coordinator) in the hierarchy, as shown in Figure 17.17. This message is propagated downward in the hierarchy. Next, a DONE message is propagated upward in the hierarchy until it reaches the Root Coordinator. Each model processor uses this DONE message to insert the time of its next change (i.e., an output message to another model, or an internal event message) before passing it to its parent coordinator. A coordinator always passes to its parent the least time change received from its children. Once the Root Coordinator receives a DONE message, it calculates the minimum next change, advances the clock, and starts the Transition phase by passing a simulation message (*). All the collected external messages are executed along with simultaneous internal events. Then, the Root receives a DONE message and starts another cycle again.

CD++ v3.0 simulator in RESTful Interoperability Simulation Environment

CD++ v3.0 simulator configuration

Because of the plug-and-play and scalable resource-oriented design characteristic of RISE middleware, simulation services are wrapped and

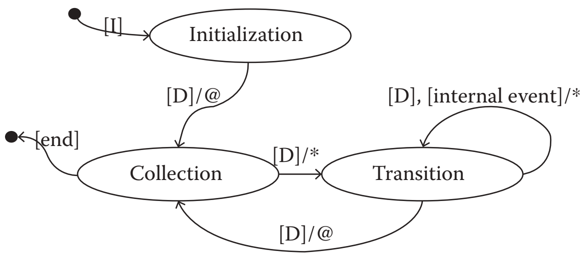


Figure 17.17 Root Coordinator Phases state diagram. (From Al-Zoubi, K. and Wainer, G., *Intelligence-Based Systems Engineering*, 10, 129–157, 2011.)

accessed through their URIs, allowing the middleware independent to any specific simulation engine. Adding a new simulation engine is easy and straightforward in RISE, basically two steps are needed: adding a new CD++ service name in {ServiceName} for RESTful WS URI pattern to recognize it, then adding the path for the new CD++ engine source code directory. In our case, CD++ v3.0 with improved functionality of CD++ is stand-alone version, not geographically distributed, so it can be viewed as single simulation engine. Simulation Manager is enough for manipulation of its execution. From a user's point of view, the modeler manipulates the entire active simulation via HTTP message via framework's URI (including its children's URIs), for example, uploading configuration files, executing simulation, and retrieving results. Figure 17.18 shows the message path when RISE gets other simulation events (e.g., external events), as the following: (1) It passes the simulation event (in XML message) to dedicated simulation manager. (2) Simulation Manager parses the XML message and passes it to Inter Process Communication queue through the local operation system. (3) The CD++ simulation engine (e.g., CD++ v3.0 in our case) executes it properly. Likewise, the RISE gets information from the simulation engine through the reverse way, which is in fact used in DCD++ between two LPs.

The XML configuration file of the CD++ v3.0 is similar as the one in DCD++. Besides multistate variables/ports, it has some other new optional features, like the <OnlyYMsgOP> for allowing to show only Y output messages in log files, and <DrawLog> for viewing 2D visualization results in *.drw file.

CD++ v3.0 simulator features

CD++ is a tool for the simulation of DEVS and Cell-DEVS models, and has been widely used to study a variety of models. CD++ v3.0 (Lopez and Wainer 2004) is an improved CD++ version for Cell-DEVS to allow

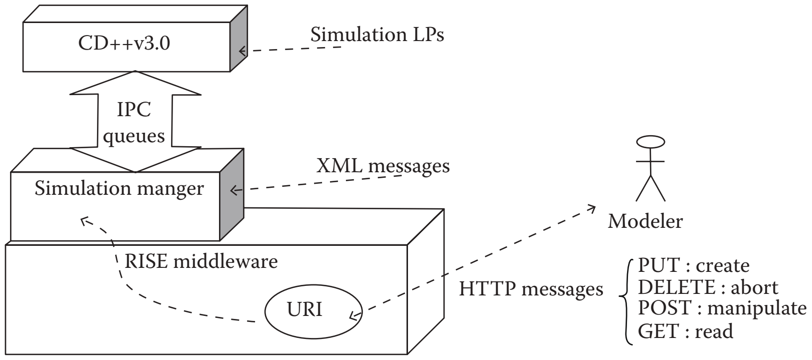


Figure 17.18 CD++ v3.0 execution session.

the cells to use multiple state variables and multiple ports for intercell communication, overcoming some limitations existing in original CD++ implementation, making CD++ more powerful.

One limitation of the original CD++ is that it only supports one state variable in each cell. CD++ v3.0 allows modelers that want to define multiple state variables per cell, avoiding creating extra planes to define as many layers as state variables needed before. Using CD++ v3.0, complex models can be easily integrated and written more clearly, reducing the development time. The format of definition of state variables is as follows:

```
StateVariables: pend temp vol
StateValues: 3.4 22 -5.2
InitialVariablesValue: init.var
```

The first line declares the list of state variables for every cell. The second line indicates the default initial values for each state variable. The last line gives the name of a file that stores some initial values for particular cells. State variables can be referred within the rules, by using "$" followed by its name. The identifier ": = " is used to assign values to state variables in a new section in the rules.

```
Rule: {(0,0) + $pend} {$temp : = $vol/2;} 10 {(0,0) > 4.5 and $vol < 22}
```

A second limitation is that original CD++ only uses one port for inputs (neighborChange) and one for outputs (out). Different I/O ports can provide a more flexible definition of the cell behavior. CD++ v3.0 supports the use of multiple I/O ports. The format of definition of that is as follows:

```
NeighborPorts: alarm weight number
```

The input and output ports share the names, and are generated automatically. An output port from a cell will influence exclusively the input port with that name in each cell of its neighborhood. I/O port can be referred by "~," and similarly, the assignment of a port is using": = ," for example, as this:

```
Rule: {~weight : = 1;} 100 {(0,1)~alarm ! = 0}
```

Visualization in Google Earth

After we got the simulation results retrieved from the WBS, we want to visualize them back in the GIS system. Here, we discuss the visualization framework for this purpose, with a particular implementation based on Google Earth using the KML files.

In general, as mentioned in Figure 17.7, this Visualization subsystem has the following four steps:

1. Results parser: it parses the simulation results preserving only output messages for visualization purposes, because the changes information of the cell values in the simulation results are stored in output messages.
2. Coordinate converter: the cells with the same state in an adjacent area can be merged together, to reduce the data size to be visualized.
3. Cell Merger: because coordinates systems used between GIS (information depicted in *.info) and Google Earth are different, to visualize the results, geography references conversion is needed.
4. KML generator: it generates processed simulated results into the desired visualization format (KML).

Once the KML file is created, it can be imported into Google Earth allowing the visualization of the simulated results evolving on a layer impressed over standard layers (e.g., satellite views, and street maps). In Google Earth, we can pause at any point and view layers forward and backward in time, which provides an interactive way to analyze the progress of a simulation.

In the Section "KML introduction," we will briefly introduce KML characteristics. Then we will present the class diagram of developed tool, followed by the details of each step mentioned previously. At last, we will show some results of the studied land use change example.

KML introduction

Google Earth (Google 2013) is a powerful visualization system supporting KML elements (place, marks, images, polygons, 3D models, textual descriptions, etc.) to manage 3D geographic data. Visualization using KML in Google Earth can enhance the communication of results of nontechnical users and provide instantaneous access to layered information. KML (OGC 2013) is based on the XML standard.

We analyzed KML structure in Wang and Chen (2012). KML uses the tags and its attributes to descript the geography information. In our implementation, we use a subset of the elements defined in KML, including <Document>, <Style>, <Folder>, <Placemark>, <Polygon>, and <Timestamp>. The following file segment shows an example. Line 1 is an XML header. Line 2 is a KML namespace declaration. Lines 3–24 state the <Document>, which is our Cell-DEVS simulation unit shown in Google Earth. Lines 5–7 state a Style, which descript a polygon style. We can define different colors for different cell states. Lines 8–23 represent a <Folder> element, which stands for a layer in our Cell-DEVS model. If the model has more than one layer, <Folder> and </Folder> pairs will also appear many times. The <Folder> element contains many <Placemark> elements (lines 9–21). In our design, a <Placemark> element depicts a cell region, which could be a square or irregular polygon. Because the <Polygon> element should have a closed loop, the first coordinate is the same as the last one specification (see lines 15 and 17). In addition, the <Placemark> element contains the <Timestamp> (line 22) element that indicates the <Placemark> showing-up time.

```
1   <?xml version = ″1.0″ encoding = ″UTF-8″?>
2   <Kml xmlns = ″http://www.opengis.net/kml/2.2″>
3     <Document>
4       <Name>Visualization</Name>
5       <Style id = "CellState">
6         <PolyStyle> <Color>aabbggrr</Color> </PolyStyle>
7       </Style>
8       <Folder id = ″#id layer″> <Name>#id layer </Name>
9         <Placemark>
10          <Name>cell name</name>
11          <StyleURL>#CellState</StyleURL>
12          <Polygon>…
13            <OuterBoundaryIs> <LinearRing>
14              <Coordinates>
15                78.7707183652255, 35.809591457038, 0
16                78.770717636844, 35.8098483367093, 0…
17                78.7707183652255, 35.809591457038, 0
18              </Coordinates>
19            </LinearRing> </OuterBoundaryIs>
20          </Polygon>
21        </Placemark>
22        <TimeStamp> <When> 2013-01-01 … </When>
    </TimeStamp>
23      </Folder>
24    </Document>
25  </Kml>
```

It is worth to note that the corresponding <Placemark> element will appear on Google Earth until either at the end of the simulation or part of the region covered by another <Placemark> element at a later stamp time, which is explained in Figure 17.19. Therefore, when a cell state changes, we only need to put that <Placemark> element with the corresponding style at that specific time. It will stay in that style until next change occurs.

Class diagram of Visualization tool

Figure 17.20 shows the class diagram of this Visualization part, updated from Wang and Chen (2012) and Zapatero et al. (2011). First, LogParser parses the data generated by the simulator preserving output messages that represent state changes. Parsed results store in the supporting structure class LogInfo. The LogInfo will facilitate and keep changing in further

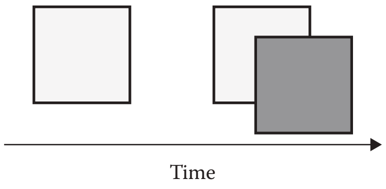


Figure 17.19 A placemark is covered by a later appearance of another one. (Wang, Y. and Chen, P. 2012. *A Cell-DEVS Geographic Visualization Framework Using Google Earth*, *Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.)

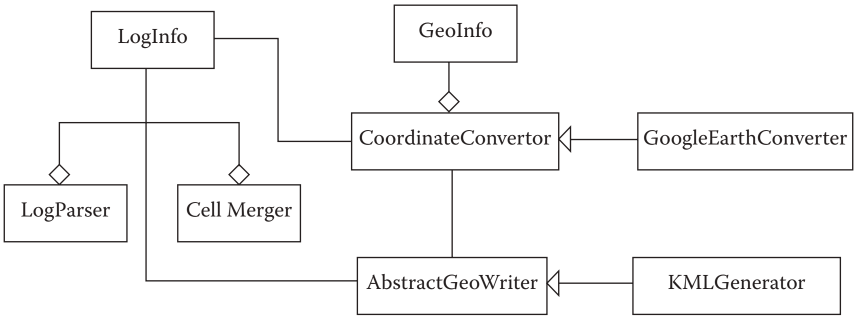


Figure 17.20 Visualization Class diagram.

processing. Then Cell Merger will merge the cells with same state in adjacent area in LogInfo, to reduce the data needed to be visualized, using presented merging algorithm (see details in the Section "Visualization process"). Note that the Cell Merger can be extended for other merging algorithms in the future. Next, CoordinateConverter changes the coordinate in LogInfo into the required way in the output visualization system, an abstract class providing an interface; in our case, its subclass GoogleEarthConverter specifies this step. Finally, AbstractGeoWriter translates LogInfo into the desired output visualization format. Similarly, AbstractGeoWriter is an abstract class providing an interface to translation methods. Here, we focus on the generation of KML files by means of the KMLGenerator class. KMLGenerator takes LogInfo information and the *.info file with the georeferences, processes them, and generates a KML file with georeferenced and timed representation of each simulated cell state change. The process consists of translating each output message into KML tags.

Visualization process

As we have seen the general process and class diagram of this GIS visualization process, now let us go deeper to see its four subprocesses depicted in Figure 17.7.

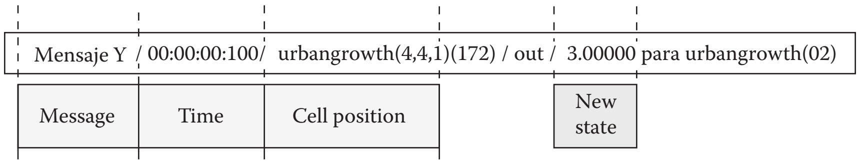

Mensaje Y / 00:00:00:100/ urbangrowth(4,4,1)(172) / out / 3.00000 para urbangrowth(02)
Message
Time
Cell position
New state

Figure 17.21 The output message format in simulation results (CD++ log file).

Step 1: Results parser

As mentioned previously, the changes information of the cell values in the simulation results are stored in output messages. For the visualization purpose, we need to parse the simulation results preserving only output messages, and transfer them into the proper format defined in LogInfo. Figure 17.21 shows the format of the matched message. The message always begins with Mensaje Y followed by the timing and cell position information. A new state value is after the/out/substring.

Step 2: Cell Merger

In each parsed message, each cell uses four positions to descript its geography information, thus we need to record all the four corners for future operation. Besides, each cell requires one <placemark> element in the KML, which makes the KML file size very big and highly redundant, resulting in a long time to render the file when loading to Google Earth. Therefore, it is important to reduce the number of KML elements to be generated. Cell Merger is responsible for solving this issue. The idea is try to merge the cells with same value in adjacent area together as much as possible. Figure 17.22 shows an example, instead of using 16 placemarks (Figure 17.22a) to represent every cell respectively, we could merge the cell with the same state value together and only need to use 4 placemarks (Figure 17.22b), and generate new polygons.

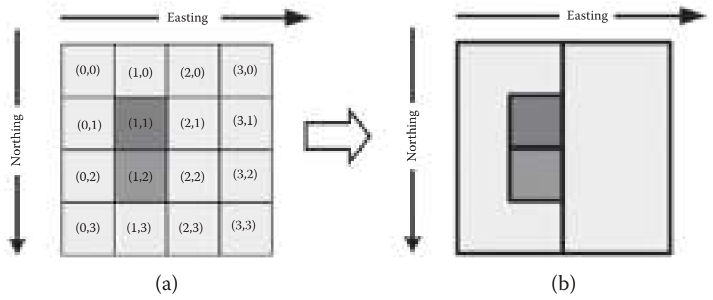


Figure 17.22 Merge cells with the same state into irregular polygons. (a) Before cells merging. (b) After cells merging. (Wang, Y. and Chen, P. 2012. A *Cell-DEVS Geographic Visualization Framework Using Google Earth, Internal Report,* Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.)

Different merging rules can be applied. Ideally, we can find all the adjacent cells for each cell; however, this kind of algorithm is time-consuming, hard to implement, and hard to retrieve the coordinate information. Therefore, we want to merge the cells with the same states as much as possible. In our implementation, we use a heuristic way to design the merging rules. The merging algorithm is shown as follows:

```
If (the current cell has the same state as its left neighbor)
    Merge into its left geometry;
Else if (the current cell has the same state as its upper
neighbor)
    Merge into its upper neighbor;
Else
    Create a new merging geometry;
```

Step 3: Coordinate converter

As we know, the land use change model is sensitive to georeferenced information and its initial data is got from GIS (GRASS), also geo.info stores the geographic information of the studied area. With this information, the geographic visualization could be achieved. The geo.info retrieved from GRASS uses Lambert Conformal Conic geography project system (LCCGPS) (GRASS GIS 2013), and this plane size is fixed by the coordinates of upper left, lower left, upper right, and lower right. Nevertheless, Google Earth uses a different kind of geography reference system called World Geodetic System 1984 (WGS84) (Google 2013). WGS84 uses longitude and latitude pairs to define the unique position on the Earth. So to visualize the simulation results on Google Earth, the geography information along with the initial parameters stored in geo.info has to be converted to longitude and latitude pairs required in Google Earth for each cell.

In the implementation, each cell has four corners with position of (East; North) (see Figure 17.23), and after cells merging step, we only need to know the corners points of all the boundaries in a merged geometry. We use the formulation mentioned in Ghilani (2010) to transform the coordinates from LCCGPS to WGS84.

Step 4: KML generator

After knowing the information needed for each <Placemark> element (all the coordinates of the merged polygon boundaries, Timestamp), we can generate KML file. Basically, this generation step will (1) generate the KML header, (2) write the KML body, and (3) conclude the KML. In the KML header, the state style (state values and associated colors) defined in the Cell-DEVS model is rewritten as polygon style used to paint each

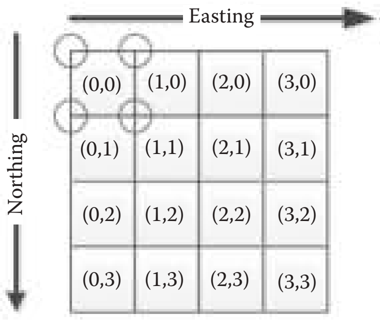


Figure 17.23 The square points of a cell in Cell-DEVS. (Wang, Y. and Chen, P. 2012. *A Cell-DEVS Geographic Visualization Framework Using Google Earth*, *Internal Report,* Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.)

<Placemark> element. For the body of KML, following simulation time, each <Placemark> element of the merged cells will list all the coordinates of its polygons; and a <Timestamp> element will be inserted to specify the showing-up time of that <Placemark> element. A <Placemark> element with a later <Timestamp> will cover fully or partially of the one in an early <Timestamp>.

Finally, the whole procedure generates geographical areas that emulate simulation in cellular spaces of a Cell-DEVS model. Now we get the KML file for our land use change example. Google Earth version 7.0.1 is used to validate our idea. After using the cell merging, 1157 geometries are needed, instead of 1787 original geometries. Figures 17.24 and 17.25 show some results obtained in Google Earth visualization.

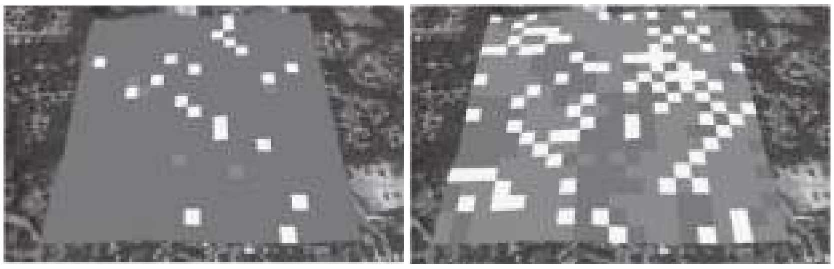


Figure 17.24 Google Earth visualization for layer 0 (land use). (a) Initial simulation time. (b) Middle simulation time. (Wang, Y. and Chen, P. 2012. *A Cell-DEVS Geographic Visualization Framework Using Google Earth*, *Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.)

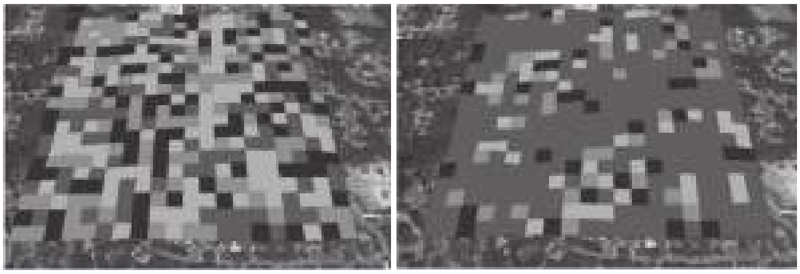


Figure 17.25 Google Earth visualization for layer 1 (population). (a) Initial simulation time. (b) Final simulation time. (Wang, Y. and Chen, P. 2012. *A Cell-DEVS Geographic Visualization Framework Using Google Earth*, *Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.)

Applications

The Section "Architecture" presented the architecture we used to integrate GIS and Cell-DEVS model using WBS, followed by detailed explanation of each steps in the Sections "Cell-DEVS modeling," "Geographic information system data collection," "Web-based simulation," and "Visualization in Google Earth." In this section, we will show two real case studies associated with more complex Cell-DEVS models (Fire Spread and Monkey Pathogen Transmission) in two different engineering fields (environmental engineering and biomedicine engineering) for testing our architecture and show its feasibility and flexibility for the M&S of environmental systems.

Fire Spread

Fire Spread is a 30 × 30 Cell-DEVS model used to study the spreading of fires in forests (Wainer 2006). This model allows foreseeing the propagation and intensity of the fire, which help us to imitate phases of the actual behavior of forest parcels in response to external events (e.g., heat, wind on the area), along with other external parameters. Different parameters affect the fire spread; in this model the following are taken into consideration in terms of the ratio of spread: (1) particles properties (amount of heat, minerals, and density), (2) type of fuel (includes the size of the vegetation), and (3) values involved with the natural environment (wind speed, territory inclination, and humidity). The behavior of each cell depends on its current state whose value is determined by a set of rules after satisfying a precondition of his neighborhood.

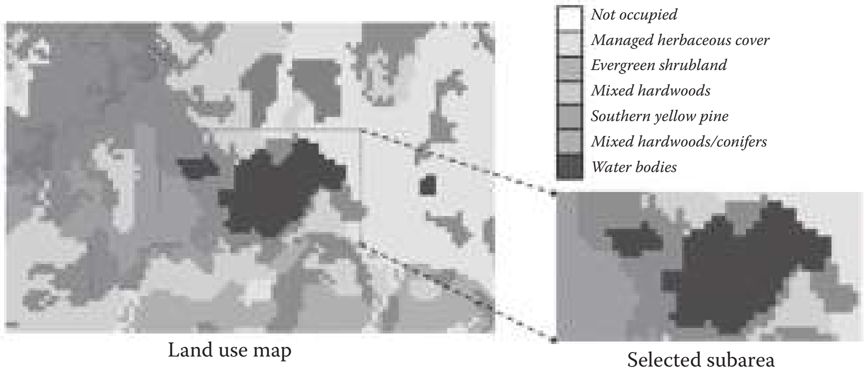


Figure 17.26 Selected subarea of land use map for fire spread simulation. (Zapatero, M. et al., Architecture for integrated modeling, simulation and visualization of environmental systems using GIS and Cell-DEVS, *Proceedings of the 2011 Winter Simulation Conference*, Phoenix, AZ, © 2011 IEEE.)

For GIS data collection, we chose the sample dataset *North Carolina, USA* from the GRASS GIS, which is a dataset readily available with the standard file GeoTIFF. Because of two water surfaces surrounded by flammable land, a section of the *landuse96_28m* map is chosen. This is useful to observe the differences in fire propagation according to the type of surface, giving us how the model sensitive to the land use information provided by GRASS. The propagation of fire is supposed to spread through the land but over water. Figure 17.26 shows the initial map and the selected subarea for simulation (Zapatero et al. 2011).

The simulation of this fire spread model is executed through WBS over RISE middleware. To realize interacting heterogeneous simulators and distributed models, we choose DCD++ for remote/distributed simulation. We first use the PUT method to create a new framework …dcdpp/firespread using distributed simulation engines under an authorized userworkspace in our RISE server, and we use the POST method to upload the initial files (*.xml with partition configuration, *.val with initial values of each cell, *.ma of Cell-DEVS fire spread model, etc.). Then we can run this simulation by using PUT to …/dcdpp/firespread/simulation. We wait for the simulation to finish, then GET simulation results files from …dcdpp/firespread/results. For distributed simulation, DCD++ allows to divide the model into different partitions to be executed on heterogeneous simulations. The model partition configuration can be specified easily in an XML file uploaded at the same time of other initial files POSTed to …dcdpp/firespread.

After retrieving the simulation log from RISE WBS middleware, we can import it into the developed GIS Visualization tool, generating a KML output file. This KML file is then loaded into Google Earth showing the following

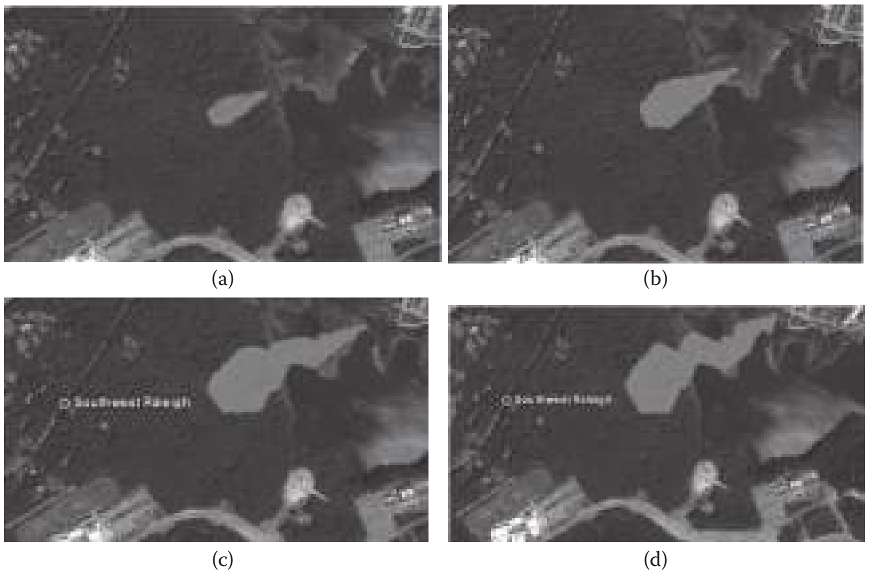


Figure 17.27 Fire model simulation results shown in Google Earth. Snapshots taken at times (a) 00:15:00, (b) 00:30:00, (c) 00:45:00, and (d) 01:00:00. (Zapatero, M. et al., Architecture for integrated modeling, simulation and visualization of environmental systems using GIS and Cell-DEVS, *Proceedings of the 2011 Winter Simulation Conference*, Phoenix, AZ, © 2011 IEEE.)

results shown in Figure 17.27, following the simulation time during 01:00:00 (Zapatero et al. 2011). It tells us that fire spreads around the lakes as expected, effected by the land use information obtained from GIS. It can also be seen that with the wind direction information extracted from GIS, the large urbanized areas (see the bottom right of Figure 17.27) are not impacted.

Monkey Pathogen Transmission

Besides the environmental system, GIS with advanced M&S techniques has also significantly influenced and improved the field of biomedicine. With the existence of several serious contagious diseases that may cause death, social panic, or even a national crisis, it is becoming essential to study infection transmission to prevent these outbreaks (Kennedy et al. 2009).

Monkey Pathogen Transmission (Wang 2012) is a Cell-DEVS model to study pathogen transmission for Macaques (a kind of monkey) in the region of Bali, Indonesia, by simulating their movement behaviors. Macaques on the island are known to carry a specific pathogen that is transmissible to neighbor macaques. Every monkey is able to host the pathogen that follows the life cycle (susceptible, latent infection, symptomatic infection, and acquired immunity). Their movement, sex, and surrounding environment effect how the pathogen is passed between each other. For example, macaques move at random, but tend to pass through waterways less frequently than forests, and female monkeys are unable to leave their birth *temples*, which have been present in the forests of Bali for centuries (Kennedy et al. 2009). This model is implemented in CD++ v3.0 specification with different state variables per cell (landscape, temple, sex, movement, and pathogen). It uses different phases in each movement cycle (intent, grant, constraint, and move). The state of each cell is determined by the values of its neighborhood followed by a set of rules (where the behavior of each cell is implemented).

This model can monitor the speed of progression and explore the effect of numerous variables on the pattern of disease transmission, allowing scientists understand the behavior of a disease and its movement from one subject to another. This model is sensitive to the landscape geographic information from GIS, requiring the river, coast, and forest information of the island Bali. In the implementation, we get the river and coast datasets for the region of Bali from an open-source website Cloudmade (Cloudmade 2013), and the forest dataset from Carleton University's Library GIS department (Carleton University 2013). Then, we combine the two datasets together using the GRASS GIS raster map calculator, and using our developed tool to get the landscape values form the map for our model. After data collection from GIS, we get the initial file *.val for the following processes. Figure 17.28 shows the initial map of Bali, Indonesia, with initial cell values for simulation (Al-Disi et al. 2013).

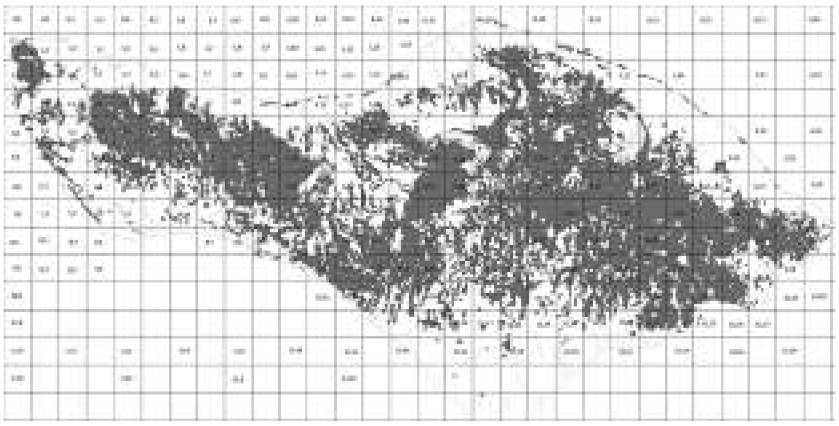


Figure 17.28 Initial cell values of Bali, Indonesia, after geographic information system data collection. (From Al-Disi, E. et al., *Visualizing Models for Biomedical Applications: Disease Transmission, Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, April 2013.)

The simulation of this pathogen transmission model is executed through WBS over RISE middleware. To use multistate variables and multiports, we use CD++ v3.0 for our simulation engine. We first use the PUT method to create a new framework …lopez/pathogen using distributed simulation engines under an authorized userworkspace in our RISE server, and we use the POST method to upload the initial files (*.xml with model configuration, *.val with initial values of each cell, *.ma of Cell-DEVS model, etc.). Then we can run this simulation by using PUT to … lopez/pathogen/simulation. We wait until the simulation finishes, then GET simulation results files from …/lopez/pathogen/results.

After retrieving the simulation log from RISE WBS middleware, we can see the simulation results in CD++ in a 2D way. Different tests have been performed with various parameters to show the effects of landscape, monkey occupancy, sex, river cross probability, and initial infection ratio on transmission patterns (see Figure 17.29). In each test, the right-most figure shows the changes of the pathogen (i.e., uninfected monkey, latent infection, symptomatic, and acquired immunity) with the movement of the monkeys. Generally, if a cell is uninfected and more than one of its neighbor is not uninfected, this cell will be infected in the next time.

We also can visualize simulation results in Google Earth. This time, we narrow down the scale of our model to a small region to verify our model scalability. Figure 17.30a shows the whole Bali Island, the small white square on the map is the region that was used to test the pathogen transmission. We reuse the previous steps easily, collecting information from GIS and retrieving the log file from RISE. Then we import the log file into our GIS Visualization tool, generating a KML output file. Finally, we load this KML file in Google Earth. The visualization results can be seen in Figure 17.30b.

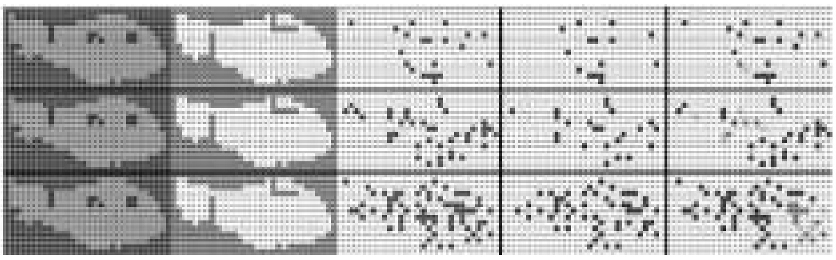


Figure 17.29 Three simulation results of Monkey Pathogen Transmission under different monkey occupancy (top with 10%, middle with 20%, bottom with 30%). (From Al-Disi, E. et al., *Visualizing Models for Biomedical Applications: Disease Transmission, Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, April 2013.)

(a) (b)

Figure 17.30 (a) Bali and (b) Visualization results in Google Earth. (From Al-Disi, E. et al., *Visualizing Models for Biomedical Applications: Disease Transmission, Internal Report*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, April 2013.)

Conclusion

We proposed a general WBS architecture by integrating Cell-DEVS modeling and GIS visualization to study environmental phenomena. We presented a workflow-like process with a land use change prototype to demonstrate the way to extract information from GIS (GRASS), run simulation remotely, and visualize results in Google Earth. The WBS is run on the WSs middleware of RISE, supporting different simulation engines. We also discussed the distributed simulation mechanism (DCD++) using RISE middleware, executing heterogeneous simulations on distributed computer systems. Two applications as case studies applying this proposed architecture were demonstrated: the wildfire spreading in the environmental engineering field and the monkey pathogen transmission in the biomedical engineering field. These applications verified and highlighted the flexibility of the proposed WBS architecture. Some future work may focus on the following aspects: (1) to expand this work with more standard DEVS protocol for distributed simulation synchronization algorithm, (2) to investigate efficient way to interface different components in the proposed architecture and apply related cloud computing technologies, and (3) to develop more applications adapting this architecture in other fields and perform more data and statistical analysis.

Acknowledgments

This chapter is the result of the recent efforts of many students and collaborators, including Mariano Zapatero and Rodrigo Castro (integration of GIS, Cell-DEVS and KLM), Yu Wang, and Peiwen Chen (designing a geographic visualization framework and a land use model), and Eman Al Disi, Joanna Lostracco, and Myriam Younan (implementing monkey pathogen transmission into the proposed architecture).

References

Al-Disi, E., Lostracco, J., and Younan, M. 2013. *Visualizing Models for Biomedical Applications: Disease Transmission. Internal Report.* Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. April 2013.

Alfonseca, M., De Lara, J., and Vangheluwe, H. 2001. Web II: web-based simulation of systems described by partial differential equations. In *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA. IEEE.

Al-Zoubi, K. and Wainer, G. 2009. Performing distributed simulation with RESTful Web-services. In *Proceedings of the 2009 Winter Simulation Conference*, Austin, TX. IEEE.

Al-Zoubi, K. and Wainer, G. 2011. Distributed simulation using RESTful interoperability simulation environment (rise) middleware. *Intelligence-Based Systems Engineering* 10, 129–157, Berlin/Heidelberg, Germany, Springer.

ANSI/EIA-632. 1999. *Process for Engineering a System*. Electronic Industry Association, Arlington, VA.

Badard, T. and Richard, D. 2001. Using XML for the exchange of updating information between geographical information systems. *Computers, Environment and Urban Systems*, 25(1), 17–31.

Band, L. 1986. Topographic partition of watersheds with digital elevation models. *Water Resources Research*, 22(1), 15–24.

Bencomo, S. D. 2004. Control learning: present and future. *Annual Reviews in Control*, 28(1), 115–136.

Bell, M., Dean, C., and Blake, M. 2000. Forecasting the pattern of urban growth with PUP: a web-based model interfaced with GIS and 3D animation. *Computers, Environment and Urban Systems*, 24, 559–581.

Bishop, I. D., and Gimblett, H. R. 2000. Management of recreational areas: GIS, autonomous agents, and virtual reality. *Environment and Planning B*, 27(3), 423–436.

Boer C., A. Bruin, and A. Verbraeck. 2009. A survey on distributed simulation in industry. *Journal of Simulation*, 3(1): 3–16.

Botkin, D. B., Janak, J. F., and Wallis, J. R. 1972. Some ecological consequences of a computer model of forest growth. *The Journal of Ecology*, 60(3), 849–872.

Byrne, J., Heavey, C., and Byrne, P. J. 2010. A review of web-based simulation and supporting tools. *Simulation Modelling Practice and Theory*, 18(3), 253–276.

Carleton University. 2013. Carleton University's Library GIS department. Available at http://www.library.carleton.ca/find/gis; accessed April 2013.

CloudMade. 2013. CloudMade. Available at http://cloudmade.com; accessed April 2013.

Desmet, P. J. J. and Govers, G. 1995. GIS-based simulation of erosion and deposition patterns in an agricultural landscape: A comparison of model results with soil map information. *Catena*, 25(1–4), 389–401.

Fujimoto, R. M. 2000. *Parallel and Distribution Simulation Systems*. New York, John Wiley & Sons.

Gimblett, R., Ball, G., Lopes, V., Zeigler, B., Sanders, B., and Marefat, M. 1995. Massively parallel simulations of complex, large scale, high resolution ecosystem models. *Complexity International*, 2, ISSN: 1320–0682.

GDAL. 2013. Geospatial Data Abstraction Library. Available at http://www.gdal.org/; accessed April 2013.

Ghilani, C. D. 2010. *Adjustment Computations: Spatial Data Analysis*. New Jersey, John Wiley & Sons.

Goble, C. A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., Borkum, et al. 2010. myExperiment: A repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, 38, Web Server issue (July 2010), W677–82.

Google, Inc. 2013. Google Earth. Available at http://earth.google.com; accessed April 2013.

GRASS GIS. 2013. Geographic Resources Analysis Support System. Available at http://grass.fbk.eu/; accessed April 2013.

Hitchins, D. K. 2008. *Systems Engineering: A 21st Century Systems Methodology*. West Sussex, United Kingdom, John Wiley & Sons.

Hu, X., Sun, Y., and Ntaimo, L. 2011. DEVS-FIRE: Design and application of formal discrete event wildfire spread and suppression models. *Simulation*. 88(3), 259–279, March 2012. doi: 10.1177/0037549711414592.

Huang, Y. and Madey, G. 2005, April. Autonomic web-based simulation. In *Proceedings of the 38th Annual Simulation Symposium* (pp. 160–167). IEEE Computer Society.

Kennedy, R. C., Lane, K. E., Arifin, S. N., Fuentes, A., Hollocher, H., and Madey, G. R. 2009. A GIS aware agent-based model of pathogen transmission. *International Journal of Intelligent Control and Systems*, 14(1), 51–61.

Khul, F., R. Weatherly, J. Dahmann. 1999. *Creating Computer Simulation Systems: An Introduction to High Level Architecture*. Upper Saddle River, NJ, Prentice Hall.

Leonard, J. 1999. *Systems Engineering Fundamentals: Supplementary Text*. Darby, PA, DIANE Publishing.

Longley, P. A, Goodchild, M. F., Maguire, D. J., and Rhind, D. W. 2005. *Geographic Information Systems and Science*. West Sussex, United Kingdom, John Wiley & Sons.

Lopez, A. and Wainer, G. 2004. Improved Cell-DEVS model definition in CD++. In *Cellular Automata*. Berlin/Heidelberg, Germany, Springer, 803–812.

Miller, J. A., Seila, A. F., and Xiang, X. 2000. The JSIM web-based simulation environment. *Future Generation Computer Systems*, 17(2), 119–133.

Myers, D. S. 2004. An extensible component-based architecture for web-based simulation using standards-based web browsers. Master's thesis, Virginia Polytechnic Institute and State University.

Narayanan, S. 2000. Web-based modeling and simulation. In *Proceedings of the 2000 Winter Simulation Conference*, Orlando, FL. IEEE.

OGC. 2013. KML. Available at http://www.opengeospatial.org/standards/kml/; accessed April 2013.

OSGeo Foundation. 2013. Geotiff format specification. Available at http://trac.osgeo.org/geotiff/; accessed April 2013.

Page, E. H. 1999. Beyond speedup: PADS, the HLA and web-based simulation. In *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation* (pp. 2–9), Atlanta, GA. IEEE Computer Society.

Page, E. H., Griffin, S. P., and Rother, L. S. (1998). Providing conceptual framework support for distributed web-based simulation within the high level architecture. In *Proceedings of SPIE: Enabling Technologies for Simulation Science II.*

Papazoglou, M. 2007. *Web Services: Principles and Technology*. Upper Saddle River, NJ, Prentice Hall.

Pullar, D. V. and Tidey, M. E. 2001. Coupling 3D visualisation to qualitative assessment of built environment designs. *Landscape and Urban Planning*, 55(1), 29–40.

Ribault, J. and Wainer, G. 2012. Simulation Processes in The Cloud for Emergency Planning. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 886–891. IEEE Computer Society.

Richardson, L. and Ruby, S. 2008. *RESTful Web Services.* Sebastopol, CA, O'Reilly Media, Inc.

Sage, A. P., and Olson, S. R. 2001. Modeling and simulation in systems engineering: Whither Simulation based acquisition? *Simulation*, 76(2), 90–91.

Sinha, R., Paredis, C. J., Liang, V. C., and Khosla, P. K. 2001. Modeling and simulation methods for design of engineering systems. *Journal of Computing and Information Science in Engineering*, 1(1), 84–91.

Strassburger, S., Schulze, T., and Fujimoto, R. 2008. Future trends in distributed simulation and distributed virtual environments: results of a peer study. In *Proceedings of the 2008 Winter Simulation Conference*, Austin, TX, eds. S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler, 777–785. Piscataway, NJ, Institute of Electrical and Electronics Engineers, Inc.

Tolk, A. 2010. Engineering management challenges for applying simulation as a green technology. In *Proceedings of the 31st Annual National Conference of the American Society for Engineering Management (ASEM)*, 137–147.

Van Der Knijff, J. M., Younis, J., and De Roo, A. P. J. 2010. LISFLOOD: A GIS-based distributed model for river basin scale water balance and flood simulation. *International Journal of Geographical Information Science*, 24(2), 189–212.

Wainer, G. 2002. CD++: A Toolkit to Develop DEVS Models. *Software: Practice and Experience*, 32(13), 1261–1306.

Wainer, G. 2006. Applying Cell-DEVS Methodology for Modeling the Environment. *Simulation*. 82(10), 635–660.

Wainer, G. 2009. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. Boca Raton, FL, CRC Press, Taylor & Francis Group.

Wang, S. 2012. *Macaque Pathogen Transmission using Cell-DEVS. Internal Report*. Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.

Wang, X. 2005. Integrating GIS, simulation models, and visualization in traffic impact analysis. *Computers, Environment and Urban Systems*, 29(4), 471–496.

Wang, Y. and Chen, P. 2012. *A Cell-DEVS Geographic Visualization Framework Using Google Earth. Internal Report*. Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. December 2012.

Ware, C. 2000. *Information visualization: Perception for design*. San Francisco, CA, Morgan Kaufmann Publishers.

Wiedemann, T. 2001. Simulation application service providing (SIM-ASP). In *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA. IEEE.

Zapatero, M., Castro, R., Wainer, G., and Hussein, M. 2011. Architecture for integrated modeling, simulation and visualization of environmental systems using GIS and Cell-DEVS. In *Proceedings of the 2011 Winter Simulation Conference*, Phoenix, AZ. IEEE.

Zeigler, B. P., Praehofer, H., and Kim, T. G. 2000. *Theory of Modeling and Simulation, 2nd Edition*. Academic Press, San Diego, CA.