Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

AN APPROACH FOR LOOSELY COUPLED DISCRETE EVENT SIMULATION MODELS AND ANIMATION COMPONENTS

Michele Fumarola
Mamadou Seck
Alexander Verbraeck

Section Systems Engineering
Faculty of Technology Policy and Management
Delft University of Technology, The Netherlands

ABSTRACT

Animation techniques are used during simulation studies for verifying and validating the model, and for communication purposes to external parties. Including animation in a simulation model, often results in models that contain many references to animation specific code. Moreover, in the specific case of discrete event simulation, challenges arise due to the difference in time-bases with animation techniques that mostly have a discrete notion of time. Typical approaches to developing animation for simulation models provide tightly coupled simulation and animation components that are fine-tuned to provide acceptable results. Apart of the disadvantage of having entangled model and animation code, this also has the disadvantage of reduced flexibility that one would desire in order to be able to use other animation techniques: for instance going from 2D to 3D with modern visualization libraries. In this paper, we will present our approach for loosely coupled discrete event simulation models and animation components.

1 INTRODUCTION

Most modern simulation packages for material flow and production processes contain an animation component to support the communication of output data to the users i.e. for verification and validation of the simulation models, and presentation to external parties such as customers (Rohrer 2000). Traditionally, 2D canvasses have been used to animate the icons representing the different entities in the simulation. In recent years however, a shift has been made from 2D canvasses to more appealing 3D renderings. Although these 3D renderings are mostly meant for marketing purposes, studies have shown that 3D renderings can also increase insight into the dynamics of the simulation model, thus increasing the support for understanding the output data and validating the simulation models (Akpan and Brooks 2005a, Akpan and Brooks 2005b).

Using animation to visualize the output of discrete event simulation is particularly challenging due to their inherent difference in the way time is managed. Discrete event simulation models' states are described in terms of the consequences of events (Nance 1981). The flow of the simulation clock would therefore perform jumps of variable sizes, depending from one event to the next, and processing the event at hand. Animation techniques, on the other hand, aim at presenting a continuous flow of events, discretized in small time steps only limited by computational capabilities. The flow of the animation clock therefore mimics the wall clock, in certain cases only differing in a scaling factor to speed up or slow down the animation. While simulators aim at performing a simulation run as fast as possible using a logical clock, animations are meant at running at a speed desired by the viewer or user, with as little jerky movements as possible. This inherent difference causes conflicts that leads

to synchronization issues between simulator and animation, and unwanted slowdowns or speedups of the animation.

Traditional approaches for implementing animation components of simulation packages result in tightly coupled environments. Both parts must be aware of specifications of the other part in order to work together in harmony. To achieve this, simulation modelers must often be aware of the visualization part and the other way around: animation designers must cooperate closely with modelers. This results in models containing specific references to the animation, such as references to animation classes in Java libraries (Jacobs, Lang, and Verbraeck 2002), specialized procedures (Caputo, Di Gironimo G, and Marzano 2006), or fully integrated simulation and animation objects (Pegden 2007). This tight coupling can however have multiple disadvantages for flexibility and clarity, clearly lacking separation of concerns. Tightly coupled simulation environments are difficult to adapt to new visualization engines: as visualization engines are rapidly improving, having the flexibility to use a new generation engine can often be interesting. In tightly coupled environments, models contain often too much information meant for the animation, which makes complicated models even harder to understand.

2 REQUIREMENTS ANALYSIS

Throughout the introduction, we have brought forward some issues that arise whenever a discrete event simulator needs to be coupled to animation components. To construct this coupling concessions have to be made to the quality of either the animation (e.g. non-fluent animation) or to the quality of the simulation model (e.g. entangled simulation model and animation code). In most cases, the result that has to be presented (end result) to the final users (i.e. clients) has priority, so that simulation modelers end up by fine-tuning their model code to have an acceptable animation. In commercial packages such as Rockwell Arena[1], and Applied AutoMod[2] this coupling has already been made by providing existing blocks to users. We, on the other hand, focus on the developers of these building blocks.

Prioritizing the end result leads to inflexible environments and models that lack clarity. Models tend mostly to be big (in terms of lines of code) and complicated: the extra code needed for animation can be considered as a burden. The extra code is often also written with a specific animation component in mind, making the complete environment inflexible, bound to be used with that specific animation component. This is not desired whenever new, more advanced animation components are available. It would therefore be desirable to have a modeling environment where no attention has to be spent on the animation and where new animation components can be easily added. Nonetheless, it would be naïve to think that we could have a simulation environment that understands what the modeler is implementing and creates appropriate animations on-the-fly (although that should be a future we should aim for). However, a clear demarcation of simulation model and animation code can already be considered a step towards both flexibility and clarity.

The requirements for a simulation environment with animation capabilities can therefore be summarized as follows:

1. Modelers (developers) and animators should be able to work independently, so that the simulation model does not need to explicitly reference to the animation. This will keep both model and animation code clear and untangled.
2. The animation component should be able to cooperate with discrete event simulators.
3. The environment should be flexible enough to be able to change animation environment easily.

3 RELATED WORK

Animating simulation models has been studied abundantly thus resulting in a vast collection of literature that cannot and will not be summarized here. We will however focus on the requirements we have set

[1] http://www.arenasimulation.com/
[2] http://www.appliedmaterials.com/

to discuss existing research and further narrow our scope by solely concentrating on discrete event simulation models.

In Wainer and Liu (2009) recent work has been discussed on visualization of DEVS models (Discrete Event System Specification formalized in Zeigler and Praehofer (2000). Apart from providing an extensive overview of the animation of DEVS models, they present their work on their CD++ environment coupled to various (popular) 3D visualization environments. To achieve this, they create various types of messages from their simulation runs, which they save and pass to the animation environments. Both their research and the related work they discuss (and similar work such as Rekapalli and Martinez (2007) and Kamat and Martinez (2005)), do not completely satisfy the requirements we set earlier. This is mainly due to coupling of model and animation code, for which the first still contains parts to control the latter.

Focus on separation of concern is given in Ribault et al. (2010) wherein they acknowledge the issue of mixing modeling and instrumentation or data-processing code. Their approach uses aspect oriented design to keep the model code clean to the bare essential. As they have a larger scope than sole animation, they do not fulfill our requirements except the first one. Their approach however, could be worthwhile to explore in order to exploit other advantages presented in their paper.

Other related work is presented by Sarjoughian and Singh (2004) whose use of the MVC pattern together with the façade pattern makes model and view as independent as possible. The choice of the façade patterns helps achieve independencies but still requires specific calls from the model to steer external components. Lastly we must mention recent efforts on web-based simulation (discussed in Page et al. (2000) with current advances presented, among others, in Zhang and Gracanin (2008), Mulligan and Gracanin (2009), and Byrne, Heavey, and Byrne (2010)) as the separation of model and view is indeed one of the advantages of web-based approaches.

4 OUR APPROACH

The approach presented here uses a model probe and a data collector such as a visualizer. The model probe does not put requirements on the data collector that therefore can assume different forms, such as a statistical analysis tool or animation. We will present both components separately and discuss how both parts can be combined.

4.1 Model Probe

The discrete event models used in our approach are defined in the Discrete Event System Specification (DEVS) formalism (Zeigler and Praehofer 2000). In this formalism, two types of models are present, namely atomic models and coupled models. An atomic model has input and output ports to connect with other models and has various transition functions that define the state changes based on events. The state variables are model specific and information considered relevant by the modeler. Coupled models, on the other hand, are solely composed of interconnected atomic models or other coupled models. The input and output ports of coupled models are directly connected to the input or output ports of contained atomic or coupled models. Furthermore, the contained atomic and coupled models can also be connected with eachother by using their input and output ports. Coupled models do not contain state variables of their own. The DEVS formalism has been implemented in the Distributed Simulation Object Library (DSOL) (Jacobs 2005) as DSOL ES-DEVS [3](Seck and Verbraeck 2009). In this Java-library, every atomic or coupled model has to be derived from the appropriate classes. Figure 1 clarifies how a conceptual DEVS model relates to its implementation in DSOL ES-DEVS by presenting a partial class diagram.

In traditional DEVS models, the flow of information goes solely through the ports, including data for statistics gathering and animation. In approaches such as discussed by Wainer and Liu (2009), a trace file is generated by outputting data at points of interest. To fulfill our first requirement however, we would want a clear demarcation of model and animation code, preferably having no reference

[3]Distributed under a BSD-style license and available at http://simulation.tudelft.nl

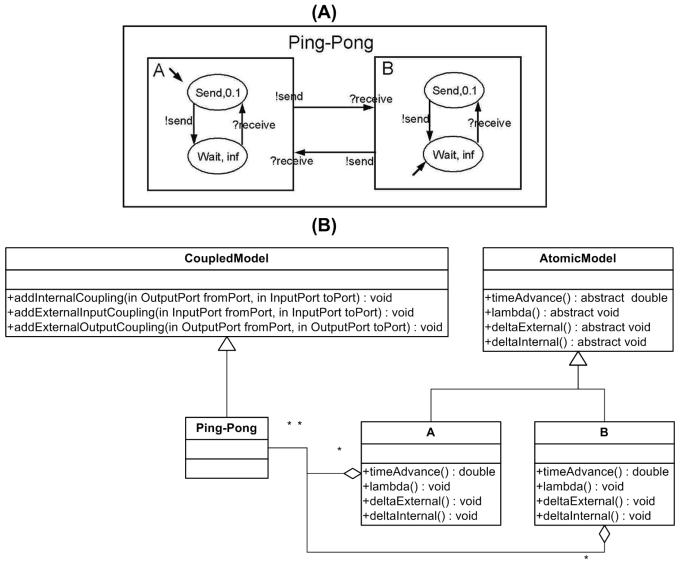

Figure 1: The Ping-Pong example model (Zeigler and Praehofer 2000) (A) implemented in the DSOL ES-DEVS library (B).

to animation components. This avoids entangled model and animation code that makes the whole more difficult to understand and maintain. To attain this, we have constructed the model probe: an approach that exploits Java's reflection capabilities and the publish-subscribe mechanism.

The abstract atomic model and coupled model classes are both derived from the AbstractDEVSModel-class that exploits the Java's reflection mechanism to collect every variable that is part of a given instantiation of a class. The collection happens as follows:

```
public static Set<SerializableField> getAllFields(final Class<?> clazz,
    final Set<SerializableField> result)
{
    Field[] fields = clazz.getDeclaredFields();
    for (int i = 0; i < fields.length; i++)
    {
        result.add(new SerializableField(fields[i]));
    }
    if (clazz.getSuperclass() != null)
    {
        return getAllFields(clazz.getSuperclass(), result);
    }
    return result;
}
```

The state can be deduced by taking the subset of the variables contained by this object. This subset must be taken as the object also contains variables that are useful for the internal logic of the DEVS implementation. Careful consideration is therefore needed to differentiate state variables from variables inherited from parent classes. This problem can however be tackled by taking the complete collection of variables and subtracting variables inherited from parent classes, those that therefore are not part of the state variables of the model. In Java this can be achieved as follows:

```
private void createStateFieldSet()
```

```
{
    Set<SerializableField> fieldSet = getAllFields(this.getClass());

    if (this instanceof AtomicModel)
    {
        fieldSet.removeAll(AbstractDEVSModel.atomicFields);
    } else if (this instanceof CoupledModel)
    {
        fieldSet.removeAll(AbstractDEVSModel.coupledFields);
    } else
    {
        fieldSet.removeAll(AbstractDEVSModel.abstractDEVSFields);
    }

    AbstractDEVSModel.stateFieldMap.put(this.getClass(), fieldSet);
}
```

Once the set of state variables has been determined, events can be fired that inform subscribed objects at each state change. To publish the state, different implementations are needed for atomic models and coupled models. Atomics models need only to publish their own state, thus firing the local set of state variables. Coupled models, on the other hand, have a state that is contained in the atomic and coupled models. Due to the hierarchical structure, this can be achieved by having a recursive function as follows:

```
public void publishState()
{
    for(AbstractDEVSModel adm : this.modelComponents)
        adm.publishState();
}
```

4.2 Animation

The animation component, which needs to be coupled to the simulation environment, is able to receive event changes by subscribing to the appropriate event generator. Once this has been done, the choice remains on what to do with these events. Although we have chosen to use the event changes for animation, one can choose to run statistical analyses with the data, or other undetermined purposes. In either case, we can define two ways of using the data: inside or outside the simulation environment.

Simulation environment with an embedded animation component are mostly compiled into a single executable. This practically means in our case implementing the animation component in Java. To achieve this, a class can be specified that receives the events and displays them in whatever way is desired to the user. This type of animation component can be useful for quick and easy ways of presenting the behavior of the model, such as in the debugging and verification stage.

Outside animation components grant much more flexibility for the animation component as it is completely decoupled for the simulation environment. To achieve this, a straightforward solution would be a network socket connection. In the simulation environment, an internet component has to be defined that receives the state changes and transforms them using a communication protocol. At the other end of the network connection, the same protocol will therefore be used to interpret the messages for animation purposes. The implementation of the animation can herewith be real-time or post-run, depending on the requirements for the animation. In either case, this fulfills our second requirement to be able to animate discrete event simulation. The third requirement is finally fulfilled by defining the intermediate language between simulation and animation component that is used to

communicate the state changes. This allows us to define new animation components as long as they can interpret the intermediate language.

In Figure 2 we can see how a typical setup can look like with animation components outside the simulation environment. The simulation model can run on a single server with a subscribed object that creates messages to send over a network. The visualizers, or other interested data collectors, can run on various different computers and connect to the server through a network. This loosely coupled architecture can be interesting whenever the simulation model or visualization component is computationally heavy.

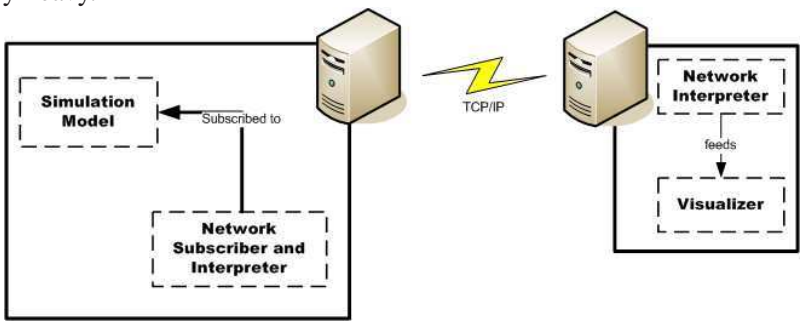

Figure 2: A diagram of the loosely coupled architecture.

5 EXAMPLE: 3D ANIMATION OF AN AUTOMATED CONTAINER TERMINAL

Container terminals provide a hub between marine and land transportation of standardized containers. Container terminals focus on two main services: load and unload container from and to vessels, barges, trucks and trains; and providing storage facility for containers. To face the continually increasing demand of these services, container terminal operators are on a constant lookout for increasing the efficiency of their container terminals. Automation has gained popularity in recent years as it provides the possibility of increasing the volumes that a container terminal can handle. However, thorough analysis is needed to design automated container terminals, which is mainly done by constructing simulation models.

To simulate automated container terminals, we built a component based model in DEVS modeling framework (Fumarola, Seck, and Verbraeck 2010) using the DSOL ES-DEVS library. To visualize the output of the simulation, we used a 2D canvas in Java to output raw data from the model, and we developed a 3D virtual environment to output a more refined animation that can be used for presentation purposes. The construction of both 3D visualization and simulation model is performed based on CAD drawings that conform to specific requirements. The creation of 3D environments and simulation models based on CAD drawings can be conducted in several ways. Whyte et al. (2000) presents three distinct ways that can be used to create solely 3D environments, thus not mentioning simulation models. The approaches are the library approach, the direct translation, and finally the database approach. The library approach uses a library of components that contains blocks that can be used in a 2D drawing and 3D models that represent the appropriate mapping. On the other hand, the simple translation purely transforms the CAD drawing to a 3D model. The CAD drawings have to be made directly in 3D which can therefore be use directly in a visualization environment. Lastly, the database approach uses a central database with a description of an object from which a 3D model and a CAD drawing can be extracted. Taking into consideration that we are not only interested in a 3D environment, but also in constructing a simulation model, we can argue that a library approach is best fit for constructing the components that both the 3D environment as the simulation model have in common (e.g. the cranes and vehicles) and use a form of direct translation for parts that are only needed for visualization (e.g. road lining). This approach allows also the re-use of existing components, which reduces time and efforts needed to construct the 3D environment. This approach has been realized by creating an intermediate XML file that extracts the needed information from a

CAD drawing. This files is used as input to the simulation server and the visualization clients. Both server and client will hereafter query the component library to construct the simulation model and visualization. This architecture is schematized in Figure 3.

To use the data from the model probe for animation, we have chosen to implement a post-run animation. This is described in Strassburger et al. (2005) and basically functions as a recorder of events that are used for animation in a later stage. Again in Strassburger et al. (2005) however, mechanisms have been defined to implement temporally parallel coupling between simulation and animation with variable buffer size. These mechanisms could be implemented in our case as well. The rationale behind the choice for a post-run animation is given by the advantages this approach brings to the users. By having a recording of a complete simulation run, users can navigate easily through the whole run. Fast forward, skip and repeat become easily manageable operations that possibly support users in the process of understanding the simulation output. However not a strict requirement, simulators should in this case be able to complete a run in a relatively short amount of time.

The animation component is loosely coupled with the simulation component through a TCP/IP connection. It receives messages containing the state changes of the simulation run. The animation component has therefore to construct the animation from these state changes. This can for instance be achieved through linear interpolation, or any other higher degree function that can describe the trajectory, based on the context of the animation. This allows us to animate cranes and vehicles, and to change the positions of containers. As all states are saved, it is also possible to skip to a specific instance of the simulation run as well as repeating passed events.

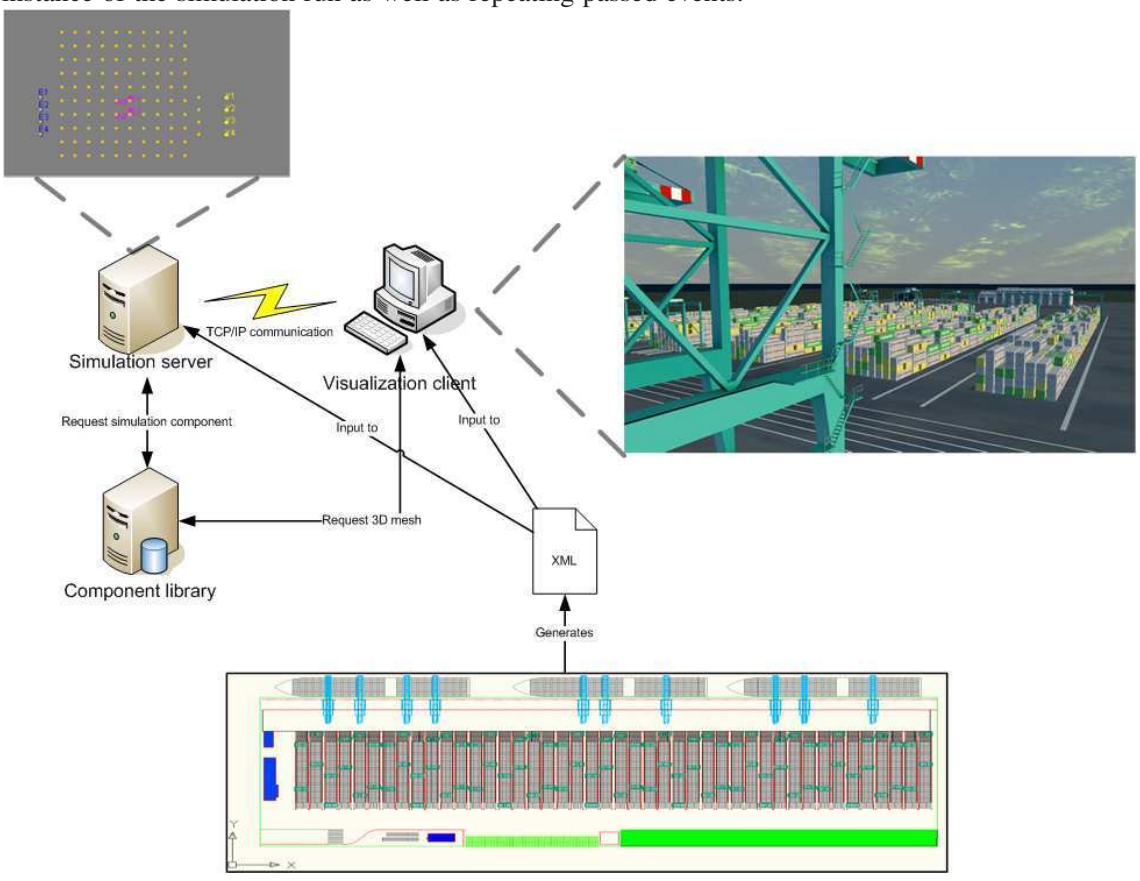

Figure 3: The schematized view of the architecture: we generate an intermediate XML from a CAD drawing; the XML file is used as input for the simulation server and the visualization client; the simulation server and visualization client request the needed components from the component library; the simulation server has a build-in 2D Java canvas visualization; the visualization client has a more sophisticated 3D virtual environment with animation.

6 CONCLUSIONS

In our research we have encountered the need of having a clear separation between simulation models and external components, such as animation components. Furthermore, we required the flexibility of being able to change animation components as seen fit in simulation projects. To achieve this, we first developed a model probe that works on every model developed in our DSOL ES-DEVS library without the need of the modeler to intervene on its workings. Hereafter, we studied how the flow of data produced by the model and captured by the model probe can be used for animation purposes. This resulted in an approach satisfing the requirements set forth in Section 1.

To illustrate our approach, we have shown how it has been used for animating a model for automated container terminals. The raw visualization of the output data of the model has been visualized in a simple 2D Java canvas, while an elaborate animation has been constructed in a separate 3D virtual environment. This showed the applicability of our approach and the flexibility that can be achieved.

REFERENCES

Akpan, J. I., and R. J. Brooks. 2005a. Experimental investigation of the impacts of virtual reality on discrete-event simulation. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 1968–1975. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Akpan, J. I., and R. J. Brooks. 2005b. Practitioners perception of the impacts of virtual reality on discrete-event simulation. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 1976–1984. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

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

Caputo, F., G. Di Gironimo G, and A. Marzano. 2006. A structured approach to simulate manufacturing systems in virtual environment. In *XVIII Congreso International de Ingenieria Grafica*.

Fumarola, M., M. Seck, and A. Verbraeck. 2010. A DEVS component library for simulation-based design of automated container terminals. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering.

Jacobs, P. 2005. *The D-SOL simulation suite - enabling multi-formalism simulation in a distributed context*. Ph. D. thesis, Delft University of Technology.

Jacobs, P., N. Lang, and A. Verbraeck. 2002. D-SOL; a distributed Java based discrete event simulation architecture. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J. L. Snowdon, , and J. M. Charnes, 793–800. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Kamat, V., and J. Martinez. 2005. Dynamic 3D visualization of articulated construction equipment. *Journal of Computing in Civil Engineering* 19:356–368.

Mulligan, G., and D. Gracanin. 2009. A comparison of soap and rest implementations of a service based interaction independence middleware framework. In *Proceedings of the 2009 Winter Simulation Conference*, ed. M. Rossetti, R. Hill, B. Johansson, A. Dunkin, and R. Ingalls, 1423–1432. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Nance, R. 1981. The time and state relationships in simulation modeling. *Communications of the ACM* 24:173–179.

Page, E., A. Buss, P. Fishwick, K. Healy, R. Nance, and R. Paul. 2000. Web-based simulation: revolution or evolution. *ACM Transactions on Modeling and Computer Simulation* 10:3–17.

Pegden, C. 2007. SIMIO: a new simulation system based on intelligent objects. In *Proceedings of the 2007 Winter Simulation Conference*, ed. S. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. Tew, and R. R. Barton, 2293–2300. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Rekapalli, P., and J. Martinez. 2007. A message-based architechture to enable runtime user interaction on concurrent simulation-animations of construction operations. In *Proceedings of the 2007 Winter Simulation Conference*, ed. S. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. Tew, and R. R. Barton, 2028–2031. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Ribault, J., O. Dalle, D. Conan, and S. Leriche. 2010. OSIF: a framework to instrument, validate, and analyze simulations. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering.

Rohrer, M. 2000. Seeing is believing: The importance of visualization in manufacturing simulation. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. Joines, R. Barton, K. Kang, and P. Fishwick, 1211–1216. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Sarjoughian, H., and R. Singh. 2004. Buidling simulation modeling environments using systems theory and software architecture principles. In *Proceeindgs of the Advanced Simulation Technology Symposium*.

Seck, M., and A. Verbraeck. 2009, July. DEVS in DSOL: Adding DEVS operational semantics to a generic event-scheduling simulation environment. In *Proceedings of SCS 2009 Summer Computer Simulation Conference*. The Society for Computer Simulation International.

Strassburger, S., T. Schulze, M. Lemessi, and G. Rehn. 2005. Temporally parallel coupling of discrete simulation systems with virtual reality systems. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 1949–1957. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wainer, G., and Q. Liu. 2009. Tools for graphical specification and visualization of DEVS models. *Simulation* 85:131–158.

Whyte, J., N. Bouchlaghem, A. Thorpe, and R. McCaffer. 2000. From cad to virtual reality: modelling approaches, data exchange and interactive 3d building design tools. *Automation in Construction* 10:43–55.

Zeigler, B. P., and H. Praehofer. 2000. *Theory of modeling and simulation*. Academic Press.

Zhang, X., and D. Gracanin. 2008. Service-oriented-architecture based framework for multi-user virtual environments. In *Proceedings of the 2008 Winter Simulation Conference*, ed. S. Mason, R. Hill, L. Monch, O. Rose, T. Jefferson, and J. Fowler, 1139–1147. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

MICHELE FUMAROLA is a Ph.D. researcher in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology. He received his M.Sc. in computer science from Hasselt University in Belgium. His current research interests are simulation and serious gaming. His email address is <m.fumarola@tudelft.nl> and his website is <http://www.tudelft.nl/mfumarola>.

MAMADOU SECK is an assistant professor in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology. He received his Ph.D. degree and his M.Sc. from the Paul Cézanne University of Marseille, and his M.Eng. from Ecole Polytechnique Universitaire de Marseille, France. His research interests include modeling and simulation formalisms, dynamic data driven simulation, human behavior representation and social simulation, and agent directed simulation. His e-mail address is <m.seck@tudelft.nl> and his website is <http://www.tudelft.nl/mseck>.

ALEXANDER VERBRAECK is a full professor in Systems and Simulation in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Tech-

nology, and a part-time full professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete event simulation for real-time control of complex transportation systems and for modeling business systems. His current research focus is on development of open and generic libraries of object oriented simulation building blocks in Java. His e-mail address is <a.verbraeck@tudelft.nl> and his website is <http://www.tudelft.nl/averbraeck>.