Component-based Distributed Simulations. The Way Forward?

Alexander Verbraeck
Delft University of Technology
Faculty of Technology, Policy and Management
Systems Engineering Section
P.O. Box 5015, 2600GA Delft, The Netherlands
a.verbraeck@tbm.tudelft.nl

Abstract

Parallel simulation and distributed simulation sometimes appear to be two different worlds. Where parallel simulation aims at increasing the speed of a single model by distributing it over more processors, distributed simulation looks at ways to link entire models or federates that run on different computers. In the case of distributed simulation, the models themselves are hard to distribute, and often they each run on one processor as a monolithic model. This paper advocates to build the more traditional simulation models in such a way, that they can be easily distributed. As simulationists, we can learn from component-based theory from the software engineering field to prepare our models for distribution, and parts of our models for reuse. The results of several projects show that componentizing simulation models can have many advantages. The results also show that it is not easy to create models in a componentized way, and that current methods for simulation model building should be adapted.

1. Parallel and distributed simulation

Parallel simulation and distributed simulation are related, but they also point to two very different motivations for wanting to distribute a model [15]. Parallel simulation aims at distributing a simulation model over a number of tightly coupled processors to speed up the run and/or spread the memory demand of the simulation execution. One of the major application areas is simulation of (tele)communication networks, which often need a lot of computer power to analyze [32], [33], [34].

The term 'distributed simulation' is often used to indicate that a number of stand-alone simulators or simulation models need to be integrated to enable joint execution. The major goal is to integrate the models from different sources into one larger model, without having to recode the model logic of each of the individual models. The HLA framework [24], developed under the guidance of DMSO and now an IEEE standard, has been the major architecture to use for distributed modeling. It has been noted, however, that distributed modeling is used mainly in the military field, and hardly in industry [44]. Nance and Sargent [28] describe the role and place of parallel simulation (p.167) and distributed simulation (p.168), and its limited use in the commercial field, and place these developments into a historical perspective. There is a limited number of studies described in literature about larger scale non-military applications of distributed simulation. Some of the notable exceptions are automobile factory simulations as described in [40] and [42], power grid simulations in [19], transport applications in [7] and [8], and enterprise simulations in [36]. So, slowly industry is increasing their use of distributed simulation. On the other hand, almost none of the existing commercial simulation software environments supports distributed modeling and simulation, nor do they support the HLA framework [43]. One of the few packages that did support HLA was the object oriented simulation language Modsim III [22], which is unfortunately not on the market anymore.

In the panel discussion described in [43], one of the major problems mentioned is the fact that common off-the-shelf simulation packages do not have the capabilities to interoperate. They usually *do* have possibilities to communicate, and wrappers are used to link one model to the other. This is in most cases not enough, and wrapping simulation environments can lead to unstable code and models that are hard to maintain, because the wrappers contain some logic of their own in addition to the model logic. Therefore, we should strive for simulation environments and simulation models that take the possibility to operate in a distributed mode as a starting point [20], [25], and look for ways to divide our simulation models into smaller parts, so it becomes easier to distribute individual models over more computers, or to group model parts in a different way than 'one model per computer'. Object orientation and component-based development immediately spring to mind as possible technologies to use when it comes to partitioning of software or models. Therefore, we will first briefly introduce the field of object-oriented development and

Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)
1087-4097/04 $20.00 © 2004 **IEEE**

component-based development, as we know it from the software-engineering field.

2. Object-oriented development

Object orientation in software development has always been a major promise for reuse and distribution. When programmed well, objects are 'closed' to the outside world due to their encapsulation [27], and access to the internal state of the object is only possible through selected 'public' methods. Object orientation is not only useful for software engineering, it can also be used in the simulation and modeling field. With the increased use of the Unified Modeling Language (see e.g. [35]), object orientation became more and more popular in the software engineering field, and also found its way to the modeling and simulation field. Many applications are developed in an object oriented way, although there are also claims of major problems with object orientation [26], [56].

Some simulation languages, such as Modsim III [22] and eM-Plant [17], [45] are based on the object-oriented paradigm. Actually, one of the earliest dedicated simulation languages, SIMULA, was fully object oriented [6], [13]. When it comes to simulation modeling, however, the object-oriented paradigm often falls short. Many of the applications of simulation are logistics simulations, where some kind of items (e.g., goods, persons, or vehicles) move through a number of processes that have limited capacity and that take some time. These processes with limited capacity are modeled as resources, which have attached queues for waiting entities when the required capacity of the resource is not available. The process interaction formalism, where the 'life cycle' of each entity type is described, is the most popular formalism for logistics simulations [4]. When modeling these logistics processes in an object-oriented way, we have to describe the 'active' servers rather than the 'passive' resources. The moving items are reduced to completely passive entities that just carry their information from one active server to the other. Model developers that 'see' the entities move from one server to the other find the object-oriented way of modeling difficult to use. Furthermore, the conceptual framework for most object-oriented simulations is the event scheduling formalism [4], rather than the process interaction formalism. Scheduling an event in the future is also more difficult to understand for a logistics person than describing the amount of time a process takes.

The developers of most of the commercially available simulation environments have tried to make a blend between object orientation and process interaction. Internally, the simulation software is built around event scheduling, allowing easy maintenance of the simulation software. For the model user, an entity / process view with queues and resources is presented for entering and changing the model. Languages like Arena [23] offer such a way of modeling, and even eM-Plant, which is an object-oriented simulation environment [45], has objects in its library that represent queues and processes 'through' and between which items can flow.

3. Component-based development

Object orientation does not guarantee reuse, nor does it guarantee that objects easily communicate with each other. The field of *component-based development* (CBD) has addressed this issue for the software-engineering world by aiming at reusable *building blocks* with a clearly defined *interface*. A research program at Delft University of Technology in the Netherlands called BETADE [52] has tried to transfer the ideas of components from the software world to other domains like modeling and simulation. We will use the word 'building block' rather than component for the applications in simulation and modeling, to distinguish it from the software components from the software engineering CBD field.

3.1. Definition of building blocks

What is a building block? Is it equal to an object or a component, and if not, what is the difference? The working definition that has been used in the BETADE research program is [52]:

> *A building block is a self-contained, interoperable, reusable and replaceable unit, encapsulating its internal structure and providing useful services or functionality to its environment through precisely defined interfaces. A building block may be customized in order to match the specific requirements of the environment in which it is 'plugged' or used.*

A clarification of the difference between a building block and components or other related terms is needed. In the area of automated transport systems, the representation of an automated guided vehicle will be a building block, as it is self-contained, interoperable, reusable and replaceable. It offers a clear service in the system, and for the system level, it hides its internal structure. In this example, an object will be a wheel: an essential, unique element of the solution of the transport problem, but not offering services on the system level. In this sense, the notion of *aggregation* or *composition* as we find in object orientation [21] and the Unified Modeling Language (UML), see e.g. [35], is clearly more important than the 'standard' object oriented software engineering concepts such as inheritance or polymorphism. Combining smaller elements into larger ones is the key.

The idea of components is derived from software-engineering environments (see e.g., [1], [2], [10], [11],

[16], [41], [55]), and builds upon the notion of distributed objects [14], [29]. However, the concept of a building block can also be used other environments, such as modeling and simulation. We state for the relationship between a building block and a component that a component is the implementation of a building block in a software environment. The interface (functionality) of the building block and the component are therefore different representations of the same thing.

This clear relationship between a component and a building block – from which the latter is clearly the broader notion – is supported by the following comments ([18], p.155): "A business component is the software implementation of an autonomous business concept (entity or process). It consists of all the software artifacts necessary to represent, implement, and deploy a given business concept as an autonomous, reusable element of a larger distributed information system. Important characteristics of the business component concept are its strong concepts of software artifact ownership, of black box/white box across the development lifecycle, of architectural viewpoints, of distribution tiers; and of its use as a unit of modularization, of deployment, of reuse, and of management."

3.2. Characteristics of building blocks

In this section, some properties of the building blocks are described. Building blocks are:

- self-contained (nearly independent of each other),
- interoperable (independent of underlying technology),
- reusable,
- replaceable,
- encapsulating their internal structure,
- providing useful services or functionality to their environment through precisely defined interfaces,
- customizable in order to match any specific requirements of the environment in which they are used (plugged).

In order to provide functionality, models are constructed by combining building blocks. More precisely stated, new – conceptual – building blocks can be formed by aggregation, which is combining lower level building blocks into new higher level building blocks. We embrace the object oriented paradigm and we want to be able to use the building blocks repetitively in our applications or models [9], [21]. Instances of building blocks will be combined to construct simulation models. The instantiation means that the attributes in the instance of the building block get a value in order to give the building block the exact functionality that the user wants within that application or model. Possibly, a number of attribute variables have a default value so the user does not have to worry about the values of all attributes of the building block.

3.3. Implementation of building blocks

The third aspect refers to the implementation aspects of a building block:

> *A building block is independent of its implementation and can therefore only described by its conceptual model and therefore by its function and interface with other building blocks. This means that independent of the implementation, a system consisting of a mixed implementation of different building blocks. For instance a combination of a simulation and a real-time application building block should function without problems.*

Even when humans fulfill certain activities, the system should remain functioning. So, building blocks go beyond software engineering or classical simulation studies. They can also be used to describe or design complex distributed business processes, such as the order handling at a bank.

3.4. Properties of building blocks

The following list of properties – inspired by [29] – of a building block can be given:

- *It can have different levels of granularity*. Building blocks can be composed of other building blocks and form so larger building blocks with more functionality. Building blocks have an extension that is dealt with in the former property.
- *It fulfils a clear function*. A building block is logically and physically cohesive, and thus denotes a meaningful structural and/or behavioral chunk of a larger system. It is not just some arbitrary grouping.
- *It has a well-specified interface*. A building block can only be manipulated through its interface. The functionality of the building block is addressed by its interface. In that way it is exposed to the outside world.
- *It exists in the context of a well-defined architecture*. A building block represents a fundamental building block upon which systems can be designed and composed. This definition is recursive: a system at one level of abstraction may simply be a building block at a higher level of abstraction.
- *It is nearly independent of other building blocks, but it rarely stands alone*. A given building block collaborates with other building blocks and in so doing assumes a specific architectural context. This architectural context is driven in large part by the implementation we chose.

- *It can be used in unpredictable combinations*. Building blocks can be used in ways that were totally unanticipated by the original developer. Typically, building blocks can be combined with other building blocks based on their interfaces.
- *It is an interoperable object*. A building block can be invoked as an object across address spaces, networks, languages, operating systems, and tools.
- *It may have several states*. A building block has some function, and by performing the function, its state may change from state, e.g. from passive to active.
- *It has limited domain specificity*. A building block is defined as being more or less generic.
- *It is a replaceable part of a system*. A building block is a substitute for any other building block, which realizes the same interfaces. This aspect helps during development, where parts of a system can be stubbed, sketched, and then replaced by mature, robust implementations. It also supports the evolution of a system, once deployed by making it possible to upgrade and evolve parts of the system independently.

4. A comparison with software components

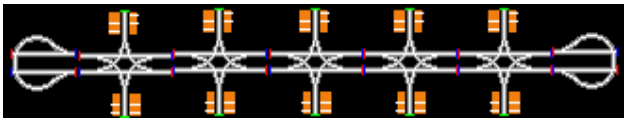
In [10], a number of definitions are presented that were discussed on a workshop on Component Based Software Development (CBSD). They represent typical examples of the definitions found in the software industry:

1. A component is a nontrivial, nearly independent, and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces. (Philippe Krutchen, Rational Software);
2. A runtime software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at runtime. (Gartner Group);
3. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition. (Clemens Szyperski, Component Software);
4. A business component represents the software implementation of an "autonomous" business concept or business process. It consists of the software artifacts necessary to express, implement, and deploy the concept as a reusable element of a larger business system. (Wojtek Kozaczynski, SSA).

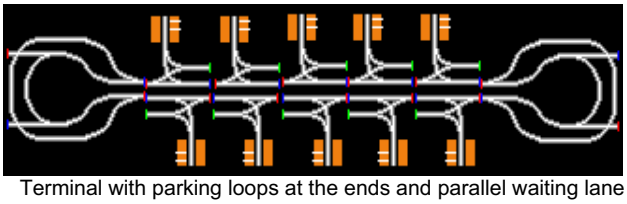
Key concepts in all these definitions is the (set of) interfaces as a means of characterizing components, and an architecture (structure) that provides the integration context for the components. The component must (be adaptable to) fit the integration context. A component can be either active or passive and a component can be seen as a black box or white box.

5. Component simulation 1: OLS

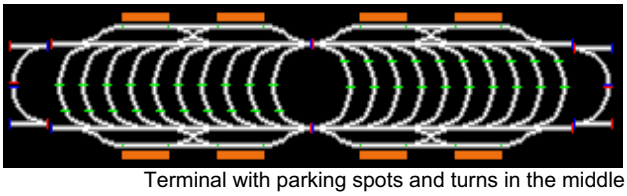
A first attempt at building block based development where building blocks can be easily interchanged, maintained, and replaced, has been carried out in a project called OLS, which is the Dutch acronym for Underground Logistic System. This research project is being carried out to survey the possibilities for underground logistic systems that use automated guided vehicles (AGVs) to transport goods on a large scale. One of the first practical applications that is being worked out is an underground logistic system called OLS Schiphol, between Amsterdam Airport Schiphol, the Flower Auction Aalsmeer, and a new high-speed rail terminal near Hoofddorp in the Netherlands. As usual in the design of a system like this, many design options were open at the start of the project. Because the number of options was so large, it was thought that a *library of easy to use simulation building blocks* would help to quickly create different versions of the system.



Compact terminal with complex crossings to save space



Terminal with parking loops at the ends and parallel waiting lane



Terminal with parking spots and turns in the middle

Figure 1. Three of the many developed OLS terminal designs, created from an eM-Plant library of simulation building blocks

After preliminary studies, several design choices were made for the OLS project. It was decided to partition the system into autonomous units as much as possible because of scalability and flexibility. Hierarchical building blocks were created (bottom-up) to be able to quickly make different infrastructure layouts [51]. After some attempts, it became apparent that infrastructure

building blocks that contain information on the service they can provide, and that can function autonomous, are very easy to use. eM-Plant [45] served as the object-oriented simulation language in which the infrastructure (terminals, tracks, docks, parking areas, maintenance areas) was built; see Figure 1 for three of the many terminal examples.

For maintaining maximum flexibility in the terminals it was chosen to use free ranging AGVs. In that way, layout changes, dock changes, and problems with the unavailability of docks or passing halted vehicles, can be solved far easier than with AGVs that drive on fixed tracks. When looking at the control system of the AGV more accurately, two subsystems can be distinguished: the AGV driver and the AGV control. The driver takes care of controlling the motors, brakes, and steering servos. The AGV driver also processes signals from sensors on the AGV. The vehicles were also implemented in the simulation language eM-Plant, using building blocks for both the AGV driver and the AGV control. Because it was known that the objects would eventually be distributed, the AGV driver and the AGV control object were implemented as two different objects with a clear interface. From the start of the project, CORBA was used to define the interface between different building blocks in the OLS project [3], [30].

In a second phase of the project, the same object-oriented building blocks for the vehicle control were used as part of different simulation models on different computers. The purpose was to analyze and test the distribution of the objects to get a feeling for the problems that might occur with the communication when the actual AGV (i.e. the AGV driver object) resides on another physical location – in this case the on-board computer of the AGV – than the AGV control object. As part of this project, more building blocks of the simulation model were distributed over different models and/or computers.

The final step in this project was the replacement of the simulated AGV driver with the real AGV. As the interfaces between the building blocks remained the same, this was nothing more than taking out one building block, and replacing it with another building block [50].

In the OLS project, many (simulation and real-time control) experiments were carried out with the library of infrastructure elements, control blocks, AGVs, and docks. The flexible and easy to change building blocks in the eM-Plant simulation library that was designed for this project enabled fast changes whenever the project required so. It became clear that self-contained building blocks with clear interfaces that use asynchronous communication are key to success in flexible real-time vehicle control [37]. The robust communication between building blocks using CORBA gave almost no problems.

Suppose that the final control information system for OLS is implemented in a traditional language like C++ using the simulation model as an example for the implementation. Usually, testing control software is quite difficult, because the effects can be quite grave when the software is not functioning properly. In this case, however, it has been suggested to the OLS management to give the newly built control software exactly the same (CORBA) interface as the simulated control system. In that way, the control system can be tested in a simulated environment with the simulated vehicles. The clear interface and the open CORBA standard used between the vehicles and the control system allows for exchanging real and simulated information in a distributed setting on both sides. A major advantage is that a problem that occurs in a real control situation can be replayed in a simulation mode, and the simulations can also be used to test different solutions before they are implemented in the real system. In experiments that were carried out, fast software development iterations in the design have been used a number of times now. Often, changes in the system took only the replacement of one building block by an adapted version. Interfaces were almost never impacted. To give an example: the interface between the AGV control and the AGV driver remained constant during the experiments for more than two years. Only recently, a new definition for curves based on B-splines has been proposed, that will ask for two simple changes in the building block definitions on both sides. Again, because the building block interfaces have been defined well, it is immediately known where this interface change will have an impact and where not, and the changes that need to be made are well isolated from the rest of the complex control system.

Although the OLS project was originally not aimed at research on building blocks or on component based development, it has become apparent that libraries with self-contained building blocks with clear interfaces have an added value in projects where flexibility and maintainability are required to test many design options.

6. Components simulation 2: supply chains

A second application area for which a library of simulation model components has been developed and tested is the area of supply chains [38], [39]. Based on literature, a supply chain architecture has been developed, that was implemented in a library of simulation building blocks in the simulation languages Arena [23] and eM-Plant [45], and in the general programming language Java. When building a supply chain model with these simulation building blocks, entire organizations are placed in the model, for which the policies can be entered in a consistent way. This tremendously speeds up the model building process and the analysis of alternatives; as such a model is much simpler to build, maintain and operate than a model built in the standard simulation environment.

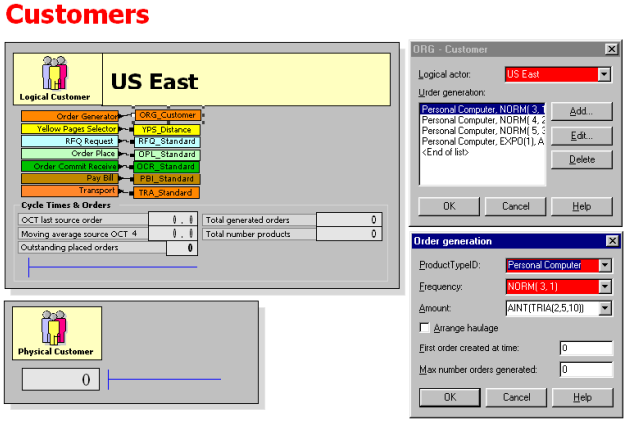


Figure 2. Customer simulation building block in Arena, when it has been dragged from the component library

In the customer or market simulation building block of Figure 2, the 'actor' that is entered in the simulation model by dragging a building block instance into the model is shown on the left hand side. In the building block, several policies can be seen in the rectangles, as well as automatically calculated performance indicators – in this case several counters and the order cycle time. The policies can be replaced with a rich set of policies from a policy library. Each policy has a number of properties, as can be seen in the right hand side of Figure 2. In this case, the order generation policy of the customer is opened, and it can be seen that the customer orders 'personal computers'. Four different orders are placed, each with its own order amount and interval. The properties of the top order are shown at the bottom right. Researching the effect of larger or more frequent orders from a certain market has been reduced to changing expressions or distribution parameters in policies. The policies themselves can also be easily replaced. Using building blocks like these saves the model builder a lot of time. The model also becomes much simpler to maintain, and the building blocks that are used have already been validated. Performance indicators have been built in to show the most important properties, and all animation functions have been defined. Several of the building blocks also show graphs and histograms of their most important indicators, such as the stock levels per product, the use of resources, and the financial situation. An example is shown in Figure 3, where the animation is shown for the 'bull-whip effect' in a simulation model with one market, five retailers, two distribution centers, one manufacturer, and three suppliers [49]. All graphs come from the building blocks that have been defined. The graphs at the bottom that show the demand for three types of organizations have been added by hand.

7. Discussion and conclusions

In addition to the two application areas described in sections 5 and 6, simulation component libraries have also been developed for airport applications [53], and for transportation applications in ports [7], [8]. In all four application projects, it turned out to be quite difficult to:

- *design* a good and consistent set of building blocks: especially the aggregation level and granularity were not easy to determine, see [47];
- *describe* the building blocks that have been created: there is no good language available for consistent component description, which supports the modeler's search process, see [5];
- *develop* the simulation model based on the building blocks: all simulation methods start from the assumption that models have to be built from scratch, and do not assume that there is a library of components available; development methods have to be changed, see [47];
- *teach* other persons than the simulation expert to use a set of building blocks: see [48];
- *make use of* standard simulation languages; although Arena and eM-Plant have been used in the two case-studies described more extensively here, preparing the models for distributed use turned out to be an extremely hard task, see [43].

Therefore, *the way forward* seems to be to research component-based development in relation to simulation and see where the two worlds can meet. Component-based simulation looks like a favorable method to join parallel and distributed simulation, and to carry distributed simulation into industry.

Acknowledgements

The research presented here has been carried out as part of the BETADE research program funded by TU Delft between 2000 and 2004. Parts of the paper have been described more extensively in [52] and build upon projects that have been carried out and published by the BETADE researchers. The author also wishes to thank TRAIL research school and Connekt for sponsoring the research described in the case studies. The concepts of this paper have also been sharpened during the extensive and interesting Dagstuhl seminar 04041 on component-based simulation in January 2004 [5].

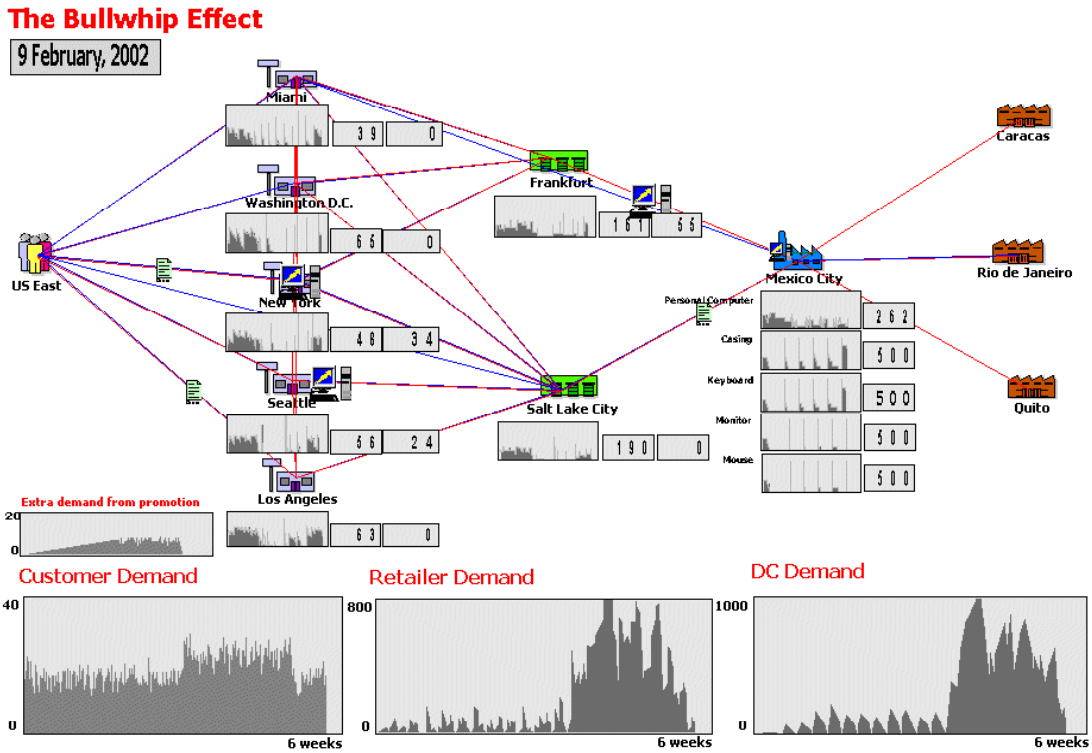


Figure 3. Example of almost automatically created output screen with the standard supply chain building blocks

8. References

[1] P. Allen, S. Frost. *Component-Based Development for Enterprise Systems*. Cambridge University Press, 1998.

[2] P. Allen. *Realizing e-Business with Components*. Pearson Education, 2001.

[3] S. Baker. *CORBA Distributed Objects*. ACM Press / Addison-Wesley, 1997.

[4] O. Balci, "The implementation of four conceptual frameworks for simulation modeling in high-level languages". In: *Proceedings of the 20th Winter Simulation Conference*. ACM Press, New York, 1988, pp. 287-295.

[5] F.J. Barros, A. Lehmann, P. Liggesmeyer, A. Verbraeck, B.P. Zeigler, *Dagstuhl Seminar on Component-based Modeling and Simulation*. http://www.dagstuhl.de/04041/

[6] G.M. Birtwistle, *Discrete Event Modelling on Simula*. Springer-Verlag, 1987.

[7] C.A. Boer, A. Verbraeck and H.P.M. Veeke. "Distributed Simulation of Complex Systems: Application in Container Handling. In: *Proceedings of the 2002 SISO European Simulation Interoperability Workshop*. SISO, Orlando, Florida., 2002.

[8] C.A. Boer, A. Verbraeck, A. de Waal, B. van Eck, and J. Seager. "Distributed e-Services for Road Container Transport Simulation". In: [54], pp. 541-550.

[9] G. Booch. *Object Oriented Design with Applications*. Benjamin / Cummings, 1991.

[10] A.W. Brown, K.C. Wallnau, "The current state of CBSE", in: *IEEE Software*, pp. 37-46, September/October 1998.

[11] A.W. Brown. *Large-scale Component-Based Development*. Prentice-Hall, 2000.

[12] S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (eds.). *Proceedings of the 2003 Winter Simulation Conference*. IEEE, 2003.

[13] O.-J. Dahl and K. Nygaard, "Simula – an ALGOL based simulation language". In: *CACM* 9:9, 1966. pp. 671-678.

[14] P. Fingar, D. Read, J. Stikeleather. *Next Generation Computing: Distributed objects for business.* SIGS, 1996.

[15] R.M. Fujimoto, "Distributed Simulation Systems". In: [12], pp. 124-134.

[16] G.T. Heineman, W.T. Councill. *Component-Based Software Engineering – Putting the pieces together*. Addison-Wesley, 2001.

[17] M.U. Heinicke and A. Hickman, "Eliminate Bottlenecks with Integrated Analysis Tools in eM-Plant". In: J.A. Joines, R.R. Barton, K. Kang, and P.A. Fishwick, (eds.), *Proceedings of the 2000 Winter Simulation Conference*. ACM/IEEE, 2000, pp. 229-231.

[18] P. Herzum, O. Sims, *Business Component Factory – A comprehensive overview of component-based development for the enterprise*, Wiley / OMG Press, 2000.

[19] K.M. Hopkinson, K.P. Birman, R. Giovanini, D.V. Coury, EPOCHS: Integrated Commercial Off-The-Shelf Software for Agent-Based Electric Power and Communication Simulation. In: [12], pp. 1158-1166.

[20] P.H.M. Jacobs, N.A. Lang, A. Verbraeck. "D-SOL; A distributed Java based discrete event simulation architecture". In: [57], pp. 793-800.

[21] I. Jacobson, M. Ericsson, A. Jacobson. *The Object Advantage – Business process reengineering with object*

technology. New York: ACM Press / Addison Wesley, 1995.

[22] G. Johnson, "Networked Simulation with HLA and MODSIM III". In: P. Farrington, H. Nembhard, D. Sturrock. and G. Evans (eds.) *Proceedings of the 1999 Winter Simulation Conference*, ACM Press, New York, NY, pp. 1065-1070.

[23] W.D. Kelton, R.P. Sadowski, and D.T. Sturrock, *Simulation with Arena*. 3rd edition. McGraw-Hill, 2003.

[24] F. Kuhl, R. Weatherly, J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall PTR, 1999.

[25] N.A. Lang, P.H.M. Jacobs, A. Verbraeck. "Distributed open simulation model development with DSOL services". In: [54], pp. 210-218.

[26] T. Love, *Object Lessons: Lessons Learned in Object-Oriented Development Projects*. Signature Sounds Recording, 1997.

[27] B. Meyer, *Object-Oriented Software Cconstruction.* Prentice Hall;, 2nd edition, 1997.

[28] R.E. Nance and R.G. Sargent. Perspectives on the Evolution of Simulation. *Operations Research*, Vol. 50, No. 1, 2002, pp 161-172.

[29] R. Orfali, D. Harkey, J. Edwards, *The Essential Distributed Objects Survival Guide*, John Wiley & Sons, 1996.

[30] R. Orfali, D. Harkey, *Client / Server Programming with Java and CORBA*, 2nd edition. John Wiley & Sons, 1998.

[31] S. Pllana and T. Fahringer, "UML based Modeling of Performance Oriented Parallel and Distributed Applications". In: [57], pp. 497-505.

[32] *Proceedings of the 2003 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2003)*. SCS, San Diego, 2003.

[33] *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2003)*, IEEE/ACM, 2003.

[34] G.F. Riley and P.A. Wilsey (eds.), *Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS2003), IEEE, Los Alamitos CA, 2003.*

[35] J. Rumbaugh, G. Booch, and I. Jacobson. *The Unified Software Development Process*, Addison-Wesley Publishing, 1999.

[36] S.M. Saad, T. Perera, and R. Wickramarachchi , "Simulation of Distributed Manufacturing Enterprises: A New Approach". In: [12], pp. 1167-1173.

[37] Y.A. Saanen, A. Verbraeck, J.C. Rijsenbrij, "The Application of Advanced Simulations for the Engineering of Logistic Control Systems". In: K. Mertins, M. Rabe (eds.). *The New Simulation in Production and Logistics – Prospects, Views and Attitudes. Proceedings 9. ASIM Fachtagung "Simulation in Produktion und Logistik"*. IPK, Berlin, 2000, pp. 217-231.

[38] J.F. Shapiro, *Modeling the Supply Chain*. Duxbury Press, 2000.

[39] D. Simchi-Levi, P. Kaminsky, E. Simchi-Levi, *Designing and Managing the Supply Chain: Concepts, Strategies, and Cases*. 2nd edition. McGraw-Hill/Irwin, 2003.

[40] S. Straßburger. *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains*. SCS European Publishing House, Erlangen, Germany, 2001.

[41] C. Szyperski. *Component Software*, 2nd edition. Addison-Wesley Publishing, 2001.

[42] G. Tan, N. Zhao, and S.J.E. Taylor, "Automobile Manufacturing Supply Chain Simulation in the GRIDS Environment". In: [12], pp. 1149-1157.

[43] S.J.E. Taylor, B.P. Gan, S. Straßburger, and A. Verbraeck, "HLA-CSPIF Panel on Commercial Off-the-Shelf Distributed Simulation". In: [12], pp. 881-887.

[44] S.J.E. Taylor, *et al.* "Distributed Simulation and Industry: Potentials and Pitfalls". In: [57], pp. 688-694.

[45] Tecnomatix, *eM-Plant version 4.6 User Guide*. Tecnomatix, Stuttgart, 2000.

[46] E. Valentin, A. Verbraeck. "Simulation using building blocks". In: F.J. Barros, N. Giambiasi (Eds). *AIS'2002 - Proceedings of AI, Simulation and Planning in High Autonomy Systems*. SCS, 2002, pp. 65-72.

[47] E. Valentin, A. Verbraeck. "Guidelines for designing simulation building blocks". In: [57], pp. 563-571.

[48] E. Valentin, A. Verbraeck, H.G. Sol. "Advantages and Disadvantages of Building Blocks in Simulation Studies: A Laboratory Experiment with Simulation Experts". In: [54], pp. 141-148.

[49] R. Van der Hee, *Simulating and visualizing the real-time supply chain*. M.Sc. thesis, Delft University of Technology / University of Maryland, 2002.

[50] A. Verbraeck, C. Versteegt, "A bridge between the design and implementation of complex transportation systems - Linking simulation models and physical systems". In: D.P.F. Möller (Ed.), *ESS2000 - Simulation in Industry. 12th European Simulation Symposium*, SCS publications, Ghent, 2000. pp. 238-243.

[51] A. Verbraeck, Y.A. Saanen, E. Valentin. "Designing Effective Terminals and their Control Systems for the Underground Logistic System Schiphol". *Proceedings ISUFT 2000, 2nd International Symposium on Underground Freight Transportation by Capsule Pipelines and Other Tube / Tunnel Systems*. CD-Rom proceedings. Delft, 2000.

[52] A. Verbraeck, A.N.W. Dahanayake (eds.), *Building blocks for Effective Telematics Application Development and Evaluation*. Delft University of Technology, 2001.

[53] A. Verbraeck, E. Valentin. "Simulation Building Blocks for Airport Terminal Modeling". In: [57]. pp. 1199-1206.

[54] A. Verbraeck, V. Hlupic (eds.), *ESS'2003, Proceedings 15th European Simulation Symposium 2003 – Simulation in Industry*, SCS European Publishing House, Germany, 2003.

[55] K.C. Wallnau, S.A. Hissam, R.C. Seacord. *Building Systems from Commercial Components*. Addison-Wesley, 2002.

[56] B.F. Webster, *Pitfalls of Object Oriented Development*. Hungry Minds, Inc, 1995.

[57] E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes, (eds.). *Proceedings of the 2002 Winter Simulation Conference*, IEEE, 2002.