Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

DESIGN GUIDELINES FOR SIMULATION BUILDING BLOCKS

Alexander Verbraeck

Delft University of Technology
Faculty of Technology, Policy and Management
Jaffalaan 5, 2628 BX
Delft, THE NETHERLANDS

Edwin C. Valentin

Systems Navigator
Delftechpark 38, 2628 XH
Delft, THE NETHERLANDS

ABSTRACT

Component based or building block based simulation model development is regularly mentioned as an interesting new development and a potential field of research. Most of the commercial simulation environments offer the users of their software functions to group model constructs and upgrade these to advanced model constructs that the users can use in future simulation studies. Unfortunately, the created model constructs are rarely reused and often stop being used after the first simulation study. In this paper we describe a list of guidelines to consider in the design of building blocks to enhance the reusability and the flexibility of the simulation building block to be used in multiple simulation studies, also by model developers who have not been involved in the design of the building blocks.

1 INTRODUCTION

The topic of domain specific building blocks and component based simulation is discussed regularly (Diamond et al. 2002; Barton et al. 2003; Barros et al. 2004). The authors conclude that the field of software development has benefited from the development of component based software, but that the field of simulation model development hardly uses component-based technologies. Still, most of the commercial simulation environments offer some technique to group model constructs together and upgrade them to a specific model construct or library element (Valentin and Verbraeck 2007).

The way that commercial simulation environments promote these techniques is as a semi-automatic model development. In their examples they show the capability as a kind of macro that automatically instantiates the constituting simulation elements into the simulation model (Rockwell Software 2000; Lanner 2003).

On the other hand, most commercial simulation environments provide sets of domain specific model constructs that can be used by other model developers to develop a simulation model in that domain. In these sets the model constructs are not just easy-to-use macros, but they provide added value to the development of a simulation model. For example, Arena has specific templates for the domains of contact centers and high speed packaging lines (Bapat and Sturrock 2003), Promodel has a specific version for hospitals (Harrell and Price 2003) and EnterpriseDynamics offers suites dedicated to modelling of airports and train networks (www.enterprisedynamics.com).

In several projects, we have encountered the difficulties and challenges that developers of sets of model constructs have in order to make sure that other people can benefit from the set and can reuse the components to compose their own simulation models. A first project was for detailed control of automatic guided vehicles and the second one was for passenger movements at airports. Section 2 describes the pitfalls for the development of a set of model constructs we encountered in these projects, and in literature.

During the simulation studies we have overcome most of these pitfalls by either spending more time, by adjusting the set, or by providing additional training to the model developers. In section 3 we show four key ways how to improve the usability and flexibility of sets of model constructs. The main proposal is a better and improved structure for model constructs which is described in section 4 together with guidelines how to build according to this structure. Finally, in section 5 and 6 we describe the benefits of following these guidelines and further research findings.

2 PITFALLS IN WORKING WITH CUSTOM SETS OF MODEL CONSTRUCTS

The following pitfalls have been identified by researchers that worked with model constructs of domain specific simulation environments.

- Scope of model developer is limited by model constructs (Sol 1982).
- Lack of trust results in less motivation to use domain specific extension (Balci 1997).
- Lack of insight in model constructs results in ignoring of domain specific extension (Kasputis and Ng 2000).

- Use of model constructs that are not suited for representation of certain system elements (Pater and Teunisse 1997).
- Model developers do not understand model construct (Barton et al 2003).
- Model developers do not know how to parameterize model construct (Barton et al. 2003).
- Difficult to compose simulation model, because important model constructs are missing (Barton et al. 2003).
- Mistakes of model developer are hard to overcome (Diamond et al. 2002).
- Model developers know something is wrong, but cannot identify what to do about it (Barton et al. 2003).
- Model constructs do not provide performance indicators wanted by the problem owner (Diamond et al. 2002).
- Model developers are limited by parameters and model constructs (Page and Opper 1999).
- A solution is identified that can not be represented by model constructs (Davis et al. 2000).
- Adjustments of model constructs required to represent solution are time consuming (Barton et al. 2003).

We have encountered several of the above mentioned pitfalls also during our projects for the development and use of sets of specific model constructs for AGV control and studying passenger flows at airports. As we knew these pitfalls could occur, we could quickly overcome them, but nevertheless they required extra attention from the developers and the model builders.

Besides the documented pitfalls we also encountered some new pitfalls which have not mentioned before in literature to our knowledge.

- Certain system elements can not be represented by model constructs.
- Model composition from developed domain specific model constructs is only applied for infrastructure system elements, not for control.
- Model developers can adjust internal logic of model constructs.
- It is difficult to compose a simulation model by persons other than developer(s) of the domain specific extension.
- Replacement of model constructs by new logic causes errors in model constructs that were linked or connected.

3 KEY ELEMENTS TO IMPROVE SETS OF MODEL CONSTRUCTS

3.1 Simulation Building Blocks

We think that a lot of the pitfalls can be resolved if the model constructs have a different structure that is prepared for reuse and maintenance, instead of only grouping a collection of model constructs together. The specific developed constructs that are following the new structure and architecture will be called a **simulation building block**.

Simulation building blocks are forthcoming out of the research of the BETADE-research group, a group of researchers at the Delft University of Technology that worked on defining, specifying and using building blocks in different modelling domains, including geo-information, web-services and discrete event simulation (www.betade.tudelft.nl). The BETADE-research group apply the following definition: "*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.*" (Verbraeck et al. 2002).

The term **self-contained** in the definition of a building block refers to the use of local information and local processes. Information is used within a building block that represents the state of the building block and affects behavior to external events. This information is used for the processes and functions the building block performs. Once the building block receives an external event to perform a function, it can execute this function with the information and process descriptions that are part of the building block. Within simulation building blocks this means that the building block keeps track of its own attributes and has all the knowledge and capabilities to represent the behavior of the system element it represents. We will enable this by storing data locally in the simulation building block and divide the functionality and services in a simulation building block over different elements with each their own part of the system representation. These elements will be referred to as **building block elements**.

Interoperable means that the building block is cooperating with other building blocks. This might seem to be contradictory with self-contained, but a simulation model cannot consist of one super building block. Different system elements are represented by different simulation building blocks. These system elements together form the system to be simulated. These system elements in reality exchange information and entities, the same occurs between simulation building blocks. However, due to the self-containment of simulation building blocks the way simulation building blocks interoperate needs to follow certain rules, i.e. by precisely defined interfaces. The ability to be interoperable starts from the idea that simulation building blocks are part of a set. This set consists of a family of building blocks which are composed of the same type of building block elements, e.g. an 'area' (a surface in the terminal) in the airport case study. In addition, the set of simulation building blocks consists of building blocks representing infrastructure or physical elements and building blocks for control or management. These simulation building blocks for control are represented as a process descrip-

tion and use pointers to be interoperable with simulation building blocks for infrastructure or physical elements.

Reusable for a simulation building block means that the simulation building blocks are instantiated more than once in a simulation model or the simulation building block is instantiated in simulation models of several simulation studies. Especially when a simulation building block is reused in a simulation model several times, it is necessary to be able to parameterize the simulation building block according to the shape and behavior of the system elements it represents. The reuse is also achieved by the use of building block elements inside a simulation building block which enables flexibility and ability to extend the set of building blocks with new building block elements instead of directly develop new building blocks.

A building block is **replaceable** if it can be removed from a system and another building block can take its place in the system. The system still works after the change, it might even act better, cheaper, more efficient or whatever objective it was aimed at. For example, replacing a CPU in computer by a newer model will result that the computer is still capable of executing the same software, but it can do it faster. The same applies for simulation building blocks. Replacing one simulation building block by another building block is a type of experimentation to evaluate system alternatives. The replaceability is achieved via the family of simulation building blocks that operate together with other simulation building blocks via standardized interfaces.

Encapsulating its internal structure means that whatever intelligent or secret mechanism is inside, the user does not need to know or find out. The model developer does not need to know what is inside a simulation building block. The internal working of a simulation building block is shown via the outside by the state of the simulation building block. The user interface will be the only thing the model developer will observe, but this can vary depending on the type of model developer, e.g. allow expert users a small peek into inner workings.

Building blocks are part of a system for a reason. This reason or reasons are **useful services or functionality** that the building block provides to a system. Every simulation building block in a simulation model should add something to the overall system representation; otherwise it could have left out of the scope of the simulation model. In addition, the service a simulation building block provides is to other building blocks in the system. A separation can be made between the type of services and functionality and how they are allocated to simulation building blocks, because no simulation building block in a domain specific extension provides all services and functionalities in the system. Most services or functionalities will be offered by building block elements inside the simulation building block. In some cases the desired service or functionality needs to be performed by the model developer by integrating the building block with model constructs of the generic simulation environment

Building blocks encapsulate their internal structure and are self-contained, yet they provide services to other building blocks and are interoperable. This means that somehow they exchange information. This information exchange cannot all be done through ad-hoc connections between building blocks, because then the building block is hard to replace by other building blocks, which will not have the same interfaces. Therefore, building blocks need to have **precisely defined interfaces**. A building block contains several types of precisely defined interfaces for different purposes. These purposes are exchange information and entities with other building blocks, parameterization by the model developer, and collection of statistics at the end of the simulation run. A part of the statistics and state representations of the simulation building block is also represented via an interface for visualization and animation.

3.2 External tools and instruments

Simulation models that are created with simulation building blocks contain a standardized way the system elements are represented. This standard format of the simulation model enables to automate model creation, parameterization for experiments and collection of performance indicators using tools outside the generic simulation environment, but dedicated for the simulation building blocks of the domain specific extension.

We define specific instruments in addition to the set of simulation building blocks. This could be an application that enables free drawing of a system in a MS-Visio® sheet and automatically translates the drawing into a simulation model using the defined simulation building blocks for that domain (Mayer et al. 2004).

Other applications are parameterization via dedicated interfaces, algorithms to check the data that is entered into the simulation model or a dedicated database that collects all standardized performance indicators of the defined simulation building blocks and presents that into a report (van der Hee 2002).

3.3 Support and training

The way that model developers will perform simulation studies with the set of simulation building blocks and the additional instruments will be different compared to a normal simulation study and compared to the normal process of performing a simulation study (Banks 1999). In the case studies we encountered that we underestimated the need of training and evaluation of the capabilities of the simulation building blocks before the model developers started the simulation study. Additional support material to be able to understand and use the simulation building blocks and the additional tools is required and possible types of support are described in this chapter.

3.4 Design approach

Finally, the new concept results in a new way of developing the domain specific extension. We encountered the need for a different perspective to the domain to make sure that the domain specific simulation environment can be used in different simulation studies. A new approach for development of domain specific extensions is also needed, because the new structure for simulation building blocks triggers trade offs for developers of domain specific extensions.

4 GUIDELINES FOR SIMULATION BUILDING BLOCKS

4.1 Self contained

The characteristic of a building block that it is self contained relates to information belonging to the simulation building block. The state of a simulation building block is a result of the values of its attributes. For example, a control simulation building block should send a message to an AGV if the control simulation building block decided that the destination of the AGV needs to change, instead of directly adjust the destination via the technical options that the generic simulation environment offers.

Guideline 1: data belonging to a building block should not be accessed by other building blocks directly, but only via defined interfaces.

The characteristic of being self contained does not just deal with the attributes of the simulation building blocks to resemble its state, but also to the functions or processes within building blocks. For example, the area model construct used to represent infrastructure for passengers in an airport had at least the following functions: resource limitation to passengers allowed; prioritization of passengers queuing for the area; determination duration passengers remain in area; collect statistics of passenger in area; relation to other areas for shortest path algorithm.

Each combination of functionalities resulted in a new model construct to represent an area according to the specific requirements of the system. An example of an area with several functionalities is the check-in area. At different airports the check-in areas were all slightly different and thus different model constructs were developed of the check-in area resulting in large sets of variants.

Software component developers overcome the large sets of variants by applying the concept of Product Line Engineering (Weis and Lai 1999). In this concept the functionalities of components are divided in small objects. One of these objects contains the core of the component that will be the same for all variant components. The other objects are adjusted to represent the different variants. Figure 1 shows at the left side three software components, before Product Line Engineering is applied. In these three components you can observe three objects (purple, yellow and green). The yellow object is the core of the component, the green and purple objects are variants for respectively function X and function Y. Product Line Engineering enables to find the commonality in the component variants and combine them together. Figure 1 shows at the right side the same three software components, but now the engineering concept is applied and the maintenance and usability is focusing on the core of the component and its objects for variation.

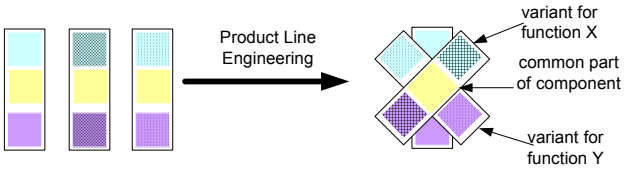

Figure 1: Product Line Engineering; find commonality in alternative components

The concept of Product Line Engineering could be applied in each simulation building block, where one or more functionalities can be observed as a functional variant. Product Line Engineering is worth applying if variants can be replaced by model developers to represent a slightly different behavior. We model these functional variants as specific objects, which will be a part of the simulation building blocks. We will refer to these specific objects as building block elements.

The simulation building block will consist of one of the instances of building block elements for each functionality. In the example of Figure 2 the yellow simulation building block will be instantiated with one of the blue and one of the purple building block elements. The model developer thus can make a decision for one of the building block elements. The selection for one of the building block elements will have effects to the functionalities that the simulation building block represents.

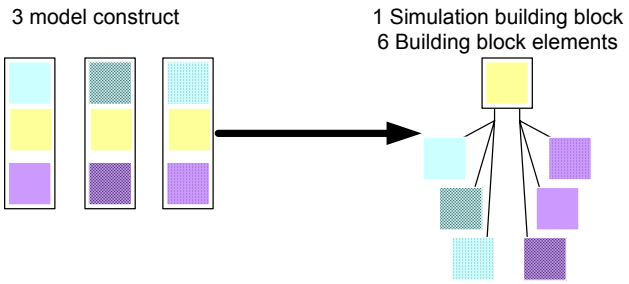

Figure 2: 3 model constructs versus 1 simulation building block with 6 building block element to represent 9 system elements.

Guideline 2: a simulation building block consists of building block elements to represent functions and services.

The building block elements that are described above can use a variety of attributes of the simulation building blocks to perform the function or service. The attributes used and information to be exchanged in these building

block elements are available within the simulation building block. A developer of simulation building blocks could decide to introduce the reduction of data exchange of building block elements similar as described in guideline 1, i.e. data exchange requires use of defined interfaces. However, this will strongly reduce the understandability of the inner logic of the building block element and therefore is seen as unnecessary overhead.

Guideline 3: data belonging to a building block element can be accessed by other building blocks elements of that building block without using the interfaces of the simulation building block.

4.2 Interoperable

The IEEE (1990) defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged." The components that the IEEE talks about are simulation building blocks instantiated in a simulation model. The ability of simulation building blocks to exchange information only applies for simulation building blocks that belong together, i.e. simulation building blocks member of a domain specific extension. The set of simulation building blocks in a domain specific extension can be structured according to different views.

One view is the identification of families of simulation building block. We define **family** as a set of simulation building blocks that represent a type of system element in different variants. Another view to the structure of simulation building blocks is the difference between infrastructure or physical building blocks and control or management building blocks. The third view to the structure is the view for fixed control or control in shape of process steps.

The areas in the domain specific extension for airports are a clear example of a family of building blocks in the set of model constructs. The simulation model was configured by instantiation of various member of the area family, for example the walking area, the conveyor, the shop, the boarding area and the check-in counter. Possibly other several families can be part of the simulation environment. For example, a second family in the domain specific extension for airports could be the mechanism of allocating airlines to check-in counters.

Guideline 4: system elements that appear in different variants and processes in a system are represented by a family of building blocks and building block elements.

In simulation studies often two types of experiments are performed to improve the system performance. The first option is to extend the availability of resources, the second option is to improve the way that resources are used. With resources we refer to things such as machines, vehicles or people. We refer to these items as infrastructure or physical elements of the systems. The infrastructure or physical elements perform processes for other elements in the system. For example, the AGVs perform a process for a load unit or an area performs a process for a passenger.

The processes, services and functions performed by infrastructure is determined by control or management of the system. For example, a management system allocates which load unit an AGV will transport. Simultaneously a control system makes sure that the AGV can safely move over infrastructure track. The control system triggers when the AGV can go and the AGV notifies the control mechanism when it reaches the end of the allowed track.

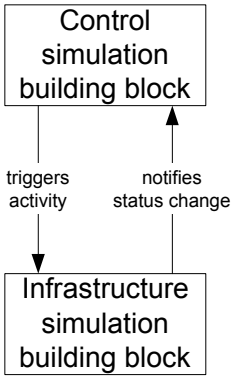

Figure 3: Simulation building blocks that separate the control and infrastructure

Guideline 5: building blocks are of different types, most common to have building blocks for infrastructure and for control.

The way that the infrastructure building blocks are controlled can be very diverse in different systems. A part of the required flexibility in control will be achieved via building block elements within control building blocks. The scripts in the AGV case study provided an alternative way for controlling the infrastructure. The control consisted of a sequence of processes instead of control that consisted out of one model construct. This approach resulted in more flexibility for the control of AGVs, with the ability to model control matching to every possible layout of tracks.

Guideline 6: complex control mechanisms are represented using control building blocks linked together to represent a flow.

Interoperable means that the building blocks work together and exchange information. The building blocks that are instantiated in the simulation model can only interact if they are aware of each others existence. The simulation environments offer several technical ways of achieving awareness that other building blocks exist in the simulation model and what they are named. The easiest way is to point to the other building blocks via their name. This process has been applied to the simulation models within the AGV case study and resulted in several errors that model constructs were not available or did not receive the correct name. We learned that the awareness between the

model constructs should be flexible and not fixed in the design of the model construct.

We introduced flexibility in defining the pointer to other building blocks in the airport case studies. For example, in the model construct to allocate flights to a check-in area, we listed the check-in areas using a pointer. These parameters were verified and error messages were generated if the pointer was invalid defined by the model developer. In that way the awareness of model constructs of other model constructs was easier to enter, check and update.

Guideline 7: building blocks should be aware of each others existence via parameters containing pointers saved in attributes.

4.3 Reusable and replaceable

Reuse or replacing a simulation building block in a simulation model can be easily achieved for building blocks that are part of the same set. A check-in counter in the simulation study of the airports could easily be replaced by a check-in counter with another function or the waiting area could be replaced by shops to entertain passengers during waiting. Replacing of elements in the simulation model can be achieved by simulation building blocks of the same family or by replacing building block elements of the instantiated simulation building blocks. This works exactly like defined in guideline 4.

The use of building block elements also improves the ability to extend the set of building blocks. System A and B have been represented with configuring simulation building blocks with alternative configuration of building block elements. System C could again require an adjustment or improvement to one of the simulation building blocks that is not available to be represented with any of the building block elements available. A new building block element will resolve the problem. The domain specific extension includes the new building block element and the model developer then can develop simulation models that represent sets that have not been considered in the original design of the domain specific extension.

Guideline 8: extension of a domain specific extension can be achieved by introducing new building block elements for existing simulation building blocks.

Replacing is not a direct result of using families of building blocks, but because within the family the building blocks have the same interface. With this interface we mean the way building blocks interact with other building blocks. If a control building block expects a check-in area to send triggers and receive notifications in a certain way, then it is important that the building block that replaces the original check-in area handles the same type of interaction via the same interfaces. This interface is the way that building blocks receive the triggers to do things. Standardized interfaces to enable replaceablity applies for building block elements in exactly the same manner as for simulation building blocks, because building block elements also need to be able to replace without any further activities or complexity for the model developer.

Guideline 9: simulation building blocks and building block elements of the same family follow the same interface requirements.

4.4 Encapsulating its internal structure

The internal structure of simulation building blocks and its underlying building block elements are hidden behind a user interface of the simulation building block in which the model developer enters the values for parameters. This user interface completely hides the code on the inside and avoids that the model developer can see what is inside or even can destroy the logic of the simulation building block or building block element.

Guideline 10: simulation building blocks hide inner working.

We observed that advanced experts want to dive into the set of model constructs and see what is hidden behind the interface. We conclude from these two observations that the user interface is a good way to hide the inner working, but that depending on the type of model developer an opening might be desired. Especially in the development of the simulation building blocks and building block elements the developers want to test and verify by evaluating the detailed logic of the simulation building block or the building block element.

Guideline 11: advanced model developers have to be able to unhide the inner logic and see how the processes and attributes are implemented.

4.5 Providing useful services and functions

The services and functionalities that simulation building blocks provide are represented by building block elements. A simulation building block can have several building block elements for the different services or functionalities it provides.

The service or functionality that should be represented by a simulation building block can be exceptional compared to the normal representation of the system element in the domain. In these cases it might not be worth to extend the domain specific extension with new building block elements. An alternative approach is to instantiate the desired functionality of the building block via custom model constructs of the generic simulation environment. We suggest therefore that a model developer should have the possibility to use generic model constructs to represent a certain system element or part of a system element.

Guideline 12: system elements should be represented by building block elements that can be extended with custom instantiation of model constructs of the generic simulation environment.

A second alternative approach is to enable simulation building blocks to interact with model constructs of the ge-

neric simulation environment. The ability to interact with the generic model constructs of the simulation environment result that the domain specific extension really is an extension to the simulation environment and not a partial replacement.

Guideline 13: a building block can connect to model constructs of generic simulation environment.

4.6 Well defined interfaces

A simulation building block contains interfaces that serve different aims, during the model development, during the simulation run and during the analysis of the simulation experiment. The interface with the simulation model developer, used mainly during model development, is to support understanding of the possible use of the simulation building block and to enable the model developer to easily parameterize the simulation building block. The interfaces used during the simulation run are interfaces to other simulation building blocks to exchange information and to trigger functions and processes of the other building blocks in the simulation model. Finally, the interfaces to be used during analysis are used by the analyst during the simulation run and after the simulation run is completed. The interface is the visualization of the state of the simulation building block including its current operation during the simulation run. The interface contains reporting of statistics gathered inside the simulation building block after the simulation run is completed.

We first describe the interface used by the model developer for parameterization, which is primarily build up on the features of the generic simulation environments to offer dialogs and consequently hide the inner working. Simulation building blocks require a similar parameterization to represent a system element of a specific system, see Figure 4.

The user interface will contain fields in which the model developer can enter the parameters for processes or statistical distributions, the availability of resources or the initial state of a simulation building block. The value that the model developer enters in the user interface of the simulation building block will be used during the simulation run and for the initialization of the simulation building block.

Guideline 14: the model developer is allowed to adjust parameters of a simulation building block via a dedicated user interface.

The user interface has an additional advantage that the developers of the simulation building block should use to the maximum. In a generic model construct the user interfaces are kept generic. For example, a process model construct uses the term “Process duration” while the process model construct refers to the process to repair break down. Terminology in the user interface clarifies the use and capabilities of the simulation building block in a domain and supports a model developer to make the correct choices for configuring.

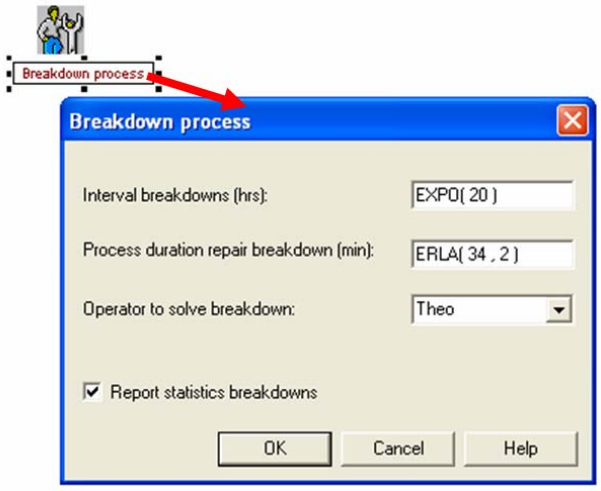

Figure 4: User interface for a simulation building block

Guideline 15: terminology in the user interface provides insight in the suitability of a building block and importance of its parameters.

Model developers will use the user interface of the simulation building block to enter their data. At the moment that they are feeding their data into the simulation building block, automatically a check can be performed to verify the use of the simulation building block by the user.

Guideline 16: parameters in a user interface of a simulation building block can be provided by model developers in a limited range of valid values.

The parameters that a model developer will use to configure an instance of a simulation building block in a simulation model will be different between projects. In some projects with a domain specific extension all parameters will be custom defined, in another simulation study it is sufficient to have some domain average values.

Guideline 17: parameters in a user interface of a simulation building block should have logic default values.

Documentation is an important instrument for model developers to understand how a simulation building block should be used. However, model developers will not spend hours to read a user manual. They need to have easy access to relevant support and explanation that links to the simulation building block they are working with.

Guideline 18: User interface of a simulation building block should provide relating support to model developer

The building block elements that represent functions and services of the simulation building block are hidden for the model developer inside the simulation building block. There can be different technical solutions to enable the model developer to change building block elements depending on the used generic simulation environment, but whatever technical option the generic simulation environ-

ment offers, the model developer will do so via the user interface.

Guideline 19: User interface of a simulation building block can be used by model developer to select building block elements.

Handling interaction between building blocks is one of the main features required for successful replaceability of simulation building blocks in a simulation model. A building block in a simulation model cannot be replaced by another building block if the two building blocks use different ways of interacting with the rest of the building blocks in the simulation model. Figure 5 is a schematic representation of the interaction issue as it appeared in the AGV case study. At the left hand side the interaction between different model constructs to the Terminal Manager (orange star) is shown. In the right picture an alternative Terminal Manager model construct is used, enforcing that the cross changes the interaction. The model construct can thus not be replaced without changing the interaction originating from other model constructs.

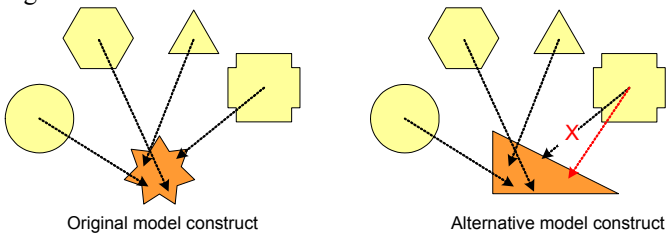

Figure 5: Unstructured interaction model constructs

In software development the issue with interaction has been tackled in two steps. The major step is to create awareness during the development of the software applications. Software developers enforce themselves to make sure that their components can follow a structure agreed with all developers. This way of developing software components is known as Design By Contract or Programming By Contract.

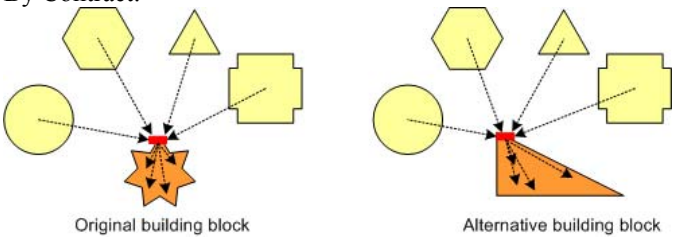

Figure 6: Structured interaction simulation building blocks

Figure 6 shows the effect of the interfaces. Other simulation building blocks no longer interact with something inside the model construct, but send their trigger or event to the interface of the simulation building block. Internally the simulation building block will redistribute the triggers to the appropriate building block element. The original building block on the left hand side in Figure 6 can be replaced by an alternative building block (right hand side of Figure 6) without any changes required to the other simulation building blocks in the simulation model.

Guideline 20: a simulation building blocks has a defined interface that receive triggers from other simulation building blocks in the simulation model and redistribute the trigger internally.

The interfaces have an additional function besides redistributing. They can also be used for checking the state of a simulation building block before continuing with a trigger. For example, an AGV cannot receive the trigger "pick up load" if it is currently in maintenance. Another check that the interface can perform is capture triggers that are not supposed to be received by a simulation building block.

Guideline 21: the interface of a simulation building block contains evaluations of the state of the trigger and the building block to determine whether the building block can handle the trigger.

The last type of interfacing of a simulation building block is the interfacing it provides to the analyst to evaluate the behavior of the system. This interface is used during the simulation run or afterwards to visualize the states and key performance indicators.

Visualization during the simulation run is provided by animation elements that are part of the simulation building block. Figure 7 shows animation using a set of pictures. Depending on the state of the simulation building block another picture is shown. Other options are counters and texts that show for example the number of entities sealed at the building block or the number of breakdowns that appear.

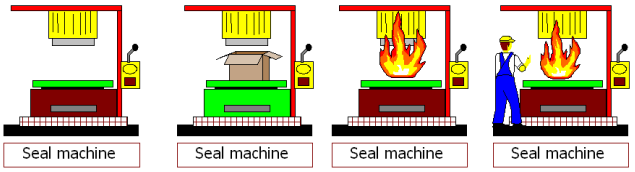

Figure 7: Four status visualizations of a building block

Guideline 22: a simulation building block contains pictures, numbers and other elements to support visualization of the state and key performance indicators during simulation run.

5 APPLICABILITY

The guidelines for simulation building blocks have been successfully applied in the development of different domain specific simulation extensions and new sets of simulation building blocks for supply chains, food factories, hospitals and container terminals.

While we implemented these domain specific sets of simulation building blocks we identified for almost each guideline a trade off for different ways that we could add more flexibility or increase ease of use. For example, the deci-

sion for the number of building block elements to use inside a simulation building block to represent its internal functions to follow guideline 2.

6 CONCLUSION

The guidelines helped in making building blocks more flexible and enabled reuse over different simulation studies. Together with the additional tools introduced in section 3.2, better documentation introduced in section 3.3 and an alternative design approach of section 3.4 we can develop better sets of simulation building blocks and also enable others to develop better simulation building blocks.

The trade offs that we encountered during the applicability of the building blocks require further analysis that will result in an improvement of the applied design approach and guidelines.

REFERENCES

Balci, O., 1997. Principles of simulation model validation, verification, and testing. *Transactions of the Society for Computer Simulation International*, 14(1):3-12.

Banks, J., 1999. Introduction to simulation. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P. A. Farrington, H. B. Nembhard, D. T. Sturrock , G.W. Evans, 7-13. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bapat, V., D. T. Sturrock, 2003. The Arena product family: enterprise modeling solutions In: *Proceedings of the 2003 Winter Simulation Conference* ed. S. Chick, P.J. Sanchez, D. Ferrin, D. J. Morrice, 210-217. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Barros, F. A., A. Lehmann, P. Liggesmeyer, A. Verbraeck, B. P. Zeigler, 2004. http://www.dagstuhl.de/04041/, [accessed May 10, 2008].

Barton, R. R., P. A. Fishwick, R. G. Sargent, J. O. Henriksen, J. M. Twomey, 2003. Panel: simulation – past, present and future. In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sanchez, D. Ferrin, D. J. Morrice, 2044-2050. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Diamond, R., J. O. Henriksen, C. D. Pegden, A. P. Walker, C. R. Harrell, W. B Nordgren, M. W. Rohrer, A. M. Law, 2002. The current and future status of simulation software (panel). In *Proceedings of the 2002 Winter Simulation Conference* ed. E. Yücesan, C.H. Chen, J.L. Snowdon, J.M. Charnes, 1633-1640. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Harrell, C.R., V. Lange, 2001. Healthcare simulation modelling and optimisation using Medmodel In *Proceedings of the 2001 Winter Simulation Conference*, ed. B.A. Peters, J.S. Smith, D.J. Medeiros, M.W. Rohrer, 233-238. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hee, R.van der. 2002. *Building blocks for Real-Time Supply Chain,* Master Thesis, Delft University of Technology, Netherlands

Kasputis S., H. C. Ng, 2000. Composable simulations. In *Proceedings of the 2000 Winter Simulation Conference* , ed. J.A. Joines, R.R. Barton, K. Kang, P.A. Fishwick, 1577-1584. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lanner Company. 2003. *WITNESS User Guide*. Lanner Group Company Limited, Redditch.

Mayer, I. S., W. Bockstael-Blok, E. C. Valentin, 2004. Containers a Drift: Visualization-simulation of an inland container terminal design. *Simulation and Gaming*, 35(1):29-52.

Page, E. H., J. M. Opper, 1999. Observations on the complexity of composable simulation. In *Proceedings of the 1999 Winter Simulation Conference* ed. P.A. Farrington, H.B. Nembhard, D.T. Sturrock, G.W. Evans, 553-560. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pater, A. J. G., M. J. G. Teunisse, 1997. The use of a template-based methodology in the simulation of a new cargo track from Rotterdam harbor to Germany In: *Proceedings of the 1997 Winter Simulation Conference* ed. S. Andradottir, K. J. Healy, D. H. Withers, B. L. Nelson, 1176-1180. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Rockwell Software, 2000. *Arena Professional – Reference guide*, Rockwell Automation, Sewickley.

Sol, H.G, 1982. *Simulation in information systems development*, Doctoral Dissertation, Rijksuniversiteit Groningen, Netherlands.

Valentin, E. C., A.Verbraeck, 2005. Requirements for Domain Specific Discrete Event Simulation Environments. In: *Proceedings 2005 Winter Simulation Conference* ed. M.E. Kuhl, N.M. Steiger, F.B. Armstrong, J.A. Joines, 654-663. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Verbraeck A., Y. Saanen, Z. Stojanovic, B. Shishkov, A. Meijer, E. Valentin, K.van der Meer, 2002. Chapter 2: What are building blocks? In: *Building blocks for Effective Telematics Application Development and Evaluation* ed. A.Verbraeck, A.Dahanayake, 8-21.

AUTHOR BIOGRAPHIES

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 Technology, 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

EDWIN C. VALENTIN is a research fellow at the Delft University of Technology and simulation consultant at Systems Navigator BV in The Netherlands. His main interest is in the creation of domain specific development environments for discrete event simulation. Edwin has implemented such environments at Nestlé, Sandd, and the British Home Office. His email address is: edwin.valentin@systemsnavigator.com