Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds..

DOMAIN SPECIFIC MODEL CONSTRUCTS IN COMMERCIAL SIMULATION ENVIRONMENTS

Edwin C. Valentin

Systems Navigator
Delftechpark 38, 2628 XH
Delft, THE NETHERLANDS

Alexander Verbraeck

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

ABSTRACT

Commercial simulation environments offer model developers the ability to compose simulation models using generic or domain specific model constructs. Most simulation environments even offer the possibility to compose custom extensions to the simulation environment for faster development of simulation models for a specific domain. This paper evaluates the functionalities for usage and development of custom domain specific extensions that 10 commonly used simulation environments provide to model developers. The findings are scored against a set of criteria, showing that currently more than half of the most used simulation environments offer support to model developers regarding domain specific extensions.

1 INTRODUCTION

Simulation experts have discussed the advantages of domain specificity of simulation environments to support model developers in several panel sessions at previous Winter Simulation Conferences (Banks et al. 2001; Diamond et al. 2002; Barton et al. 2003). In these panel sessions they concluded that the ability of model developers to create their own model constructs for their own specific domains will enable faster model development, make it easier to perform simulation experiments, and reduce verification and validation efforts.

All commercially available simulation environments have acknowledged these advantages and provide some way of enabling domain specificity into their environment. The simulation environments offer specific support to model developers mainly for the domains in which the environments are widely applied. 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>.

The simulation environments also offer model developers functionality to develop their own model constructs. In this way the model developer can develop his or her own set of model constructs to help with the model development in a typical domain. Commercial simulation environments that offer the feature of developing custom sets of model constructs apply different ways and types of development and instantiating. This paper provides an overview of how domain specificity is offered in commercial simulation environments and how model developers can develop their own sets of domain specific model constructs.

Swain (2005) lists over 50 commercial simulation environments. We decided to describe the top-10 of simulation environments that are the most popular within the papers of the Winter Simulation Conference of 2006. Rockwell Automation (2007) provided an overview of the most referred commercial simulation environments at the conference of 2006, see Figure 1. We selected the top 10 simulation environments of this analysis.

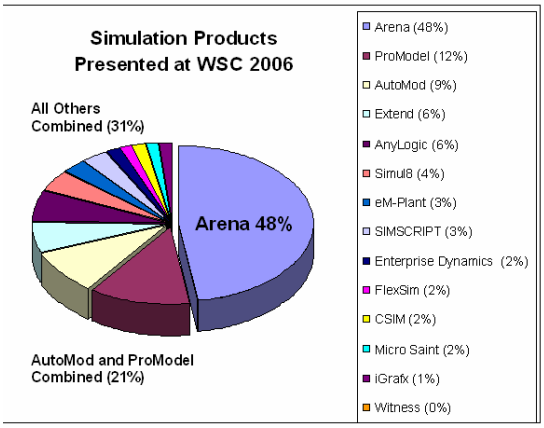

Figure 1: Most referred commercial environments (Rockwell Automation 2007).

All simulation environments use different terms to refer to model constructs and their sets. In section 2, we will

introduce some terms that we will apply in the rest of the paper to avoid confusion about terminology. In section 3 we describe for the selected simulation environments how model developers develop a simulation model in a commercial environment and which domain specific sets of model constructs are offered by simulation environments. In section 4 we describe how model developers can develop their own (sets of) model constructs. Section 5 contains a summary of concepts and our impression of advantage and disadvantages. Finally, section 6 concludes this paper linking our findings to the remarks of simulation experts in the panel sessions.

2 TERMINOLOGY

Arena uses modules and sets of templates, Automod refers to loads, resources or processes and systems and Enterprise Dynamics talks about atoms and libraries. All these different names can lead to confusion. Therefore we introduce the term "model construct". **Model constructs** are elements in a simulation model that represent a certain part of the system that is described by the simulation model.

Model constructs are **domain specific** if they are member of a set that the developer combines together to enable model development for more than one problem in that domain. Note: a set of model constructs can be domain specific to one person, while another model developer might miss representation of system elements that he/she assumes to belong to the domain.

A **simulation environment** is a set of one or more applications that support the model developer in instantiating a simulation model to represent a system and to execute the processes for a defined time frame (Nance 1993). A set of specific model constructs is an extension for a generic simulation environment. In this paper an extension for a commercial simulation environment for a specific domain is called a "**domain specific extension for a simulation environment**" abbreviated to "**domain specific extension**". A domain specific extension restricts model developers to implement simulation models of a specific domain, based on a conceptual model of that domain. The domain specific extension consists of model constructs that can be directly derived from the elements of the conceptual model.

A simulation model is instantiated in a simulation environment by creating an aggregate of model constructs and defining interactions between model constructs. These interactions can be named references, or links using arrows. The model constructs in the simulation model have a relation with the model construct in the domain specific extension. The strength of the relation between the model construct in the extension and the model construct in the simulation model determine whether the simulation environment applies inheritance and derivation of model constructs, or duplication while instantiating.

Inheritance is a concept from the domain of object orientation which forces model constructs to keep links to their parent, i.e. the model construct in the extension. Children, e.g. model constructs instantiated into a simulation model, behave the in the same way as their parents unless defined differently. Children have the same attributes as their parent. Changes that apply to the parent can also have effects on the children in the simulation model, depending on the way inheritance is implemented in the simulation environment.

On the other hand, simulation environments that apply duplication do not keep any relation between the model construct in the extension and the model construct in the simulation model. Changes to the model construct in the extension will not have any effect to the model construct in the simulation model. The use of inheritance in a simulation environment is important, because it has effects on the development process of model developers for their custom domain specific extensions.

3 USE OF MODEL CONSTRUCTS

3.1 Arena

Arena refers to its model constructs as modules and the extensions are called templates. Arena has three generic templates called "Basic Process", "Advanced Process" and "Advanced Transfer". The model constructs in these templates are composed from the underlying SIMAN language. The SIMAN language consists of blocks and elements and these model constructs are also available within Arena. As commercial extensions Rockwell Automation offers Contact Center for call centers and packaging for high speed packaging lines like bottlers.

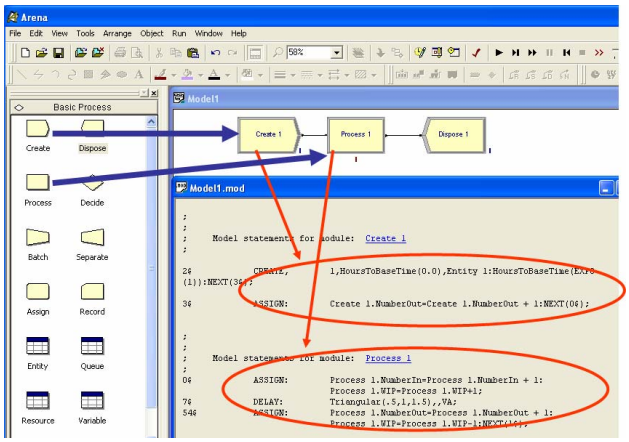

Figure 2: Model instantiation in Arena.

A model developer drags a model construct into the simulation model (see thick blue arrow in Figure 2) from one of the attached templates. At the moment the model developer checks the simulation model for syntax errors

prior to running, the SIMAN code that belongs to each model construct in the simulation model is generated (see thin red arrows and circle in Figure 2). Arena does not use inheritance as such, but generating the SIMAN code at the last possible moment ensures that always the latest version of the SIMAN code of a model construct is applied in the simulation model. A model developer does not have the capability to disconnect the link between the model construct in the model and the model construct in the extension.

3.2 Promodel

A simulation model in Promodel (Harrell and Price 2003) is instantiated by dragging a graphical representation of an element, location or resource into the simulation model. The model developer also has the possibility of defining these model constructs in a spreadsheet view before adding the graphical representation to the model (see blue arrows in Figure 3).

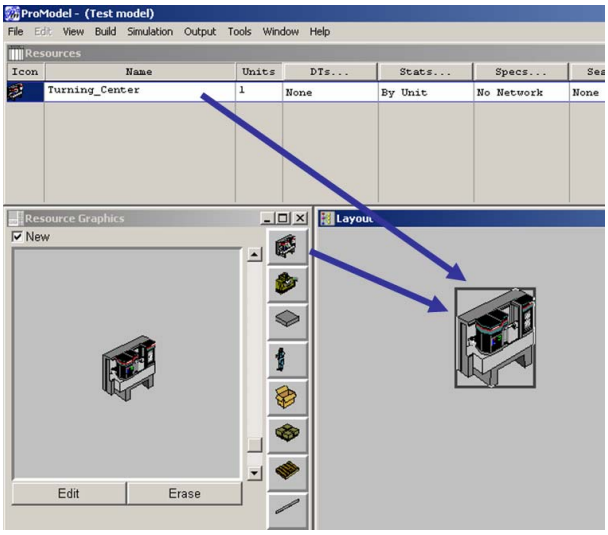

Figure 3: Model instantiation in Promodel.

Promodel does not seem to have one overview of model constructs to be added, at least not in version 4.2). The model developer can select another graphical representation for each resource, element or location to be instantiated into the simulation model, but that concerns only the graphical representation of these model constructs.

Promodel provides a separate application called MedModel that focuses on the modeling of hospitals (Harrell and Lange 2001). The main difference with the generic edition of Promodel is the availability of dedicated graphics and examples. The model constructs in this version are not different from the resource, element or location present in Promodel.

3.3 Automod

Automod is probably the simulation environment with most commercial domain specific extensions. Automod uses so-called "systems" that contain model constructs to be added to a simulation model. Every simulation model has a system containing generic model constructs such as a load or resource and a system for the process logic of these model constructs. In addition Automod offers systems for conveyors, automatic guided vehicles, warehouse systems, stacking cranes, overhead conveyors and wafer production centers (Rohrer 2003).

The model constructs of these systems contain the specific behavior of the equipment they refer to. The model constructs also contain dedicated events and model developers can add their detailed logic to be executed once a model constructs fires a defined event. Before the simulation model will be executed, Automod carries out a compilation of the equipment and the process logic defined by the model developer. During the compilation the latest available system information will be used to model the piece of equipment in the simulation model. A system can thus be adjusted and changes will be incorporated in the simulation model.

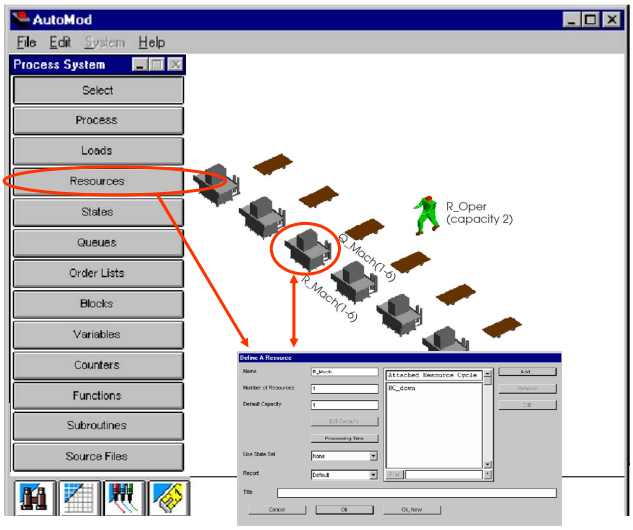

Figure 4: Model instantiation in Automod.

3.4 Extend

Extend provides domain specificity in two ways, using suites and libraries. Suites are commercial packages that are aimed to a certain domain and contain libraries dedicated for that domain, in addition to some dedicated user interfaces and representations of the simulation environment, applicable to that domain. Libraries are what we refer to as domain specific extensions, the sets of model constructs to enable instantiation of a simulation model. A library in Extend consists of a block that can represent a

piece of equipment or a process step for entities (Krahl 2003).
A block is instantiated by dragging from a library. A simulation model in Extend can be composed of model constructs from different libraries. Given the correct semantics it is possible to link the model constructs of the different libraries, as if they always belonged together. An example of model instantiation is provided in Figure 5. A model construct in Extend is developed using coding that is very similar to C. The latest version of the C-logic of the model constructs will be compiled just before the simulation model is executed. A model construct that is instantiated in the simulation model always will have the same C-logic as provided in the library.

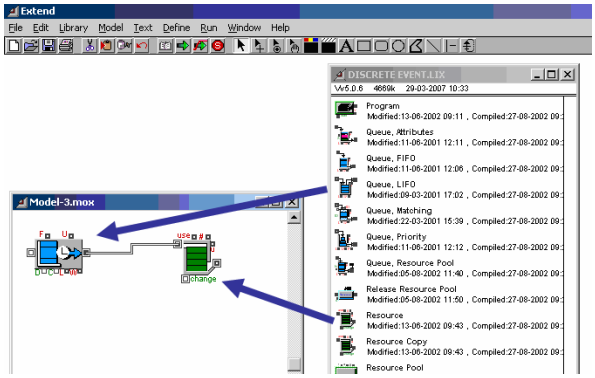

Figure 5: Model instantiation in Extend.

3.5 Anylogic

Anylogic is a simulation environment based on the programming language Java. In this simulation environment classes are defined as Java classes and objects of these classes can be instantiated by dragging them into a root view, automatically a representation will be added to the visualization view (see thick blue arrows in Figure 6). Anylogic provides several libraries of objects for domains in which they have performed successful simulation studies. Among these domains are transport, hospitals and logistics (XJ Technologies 2005).

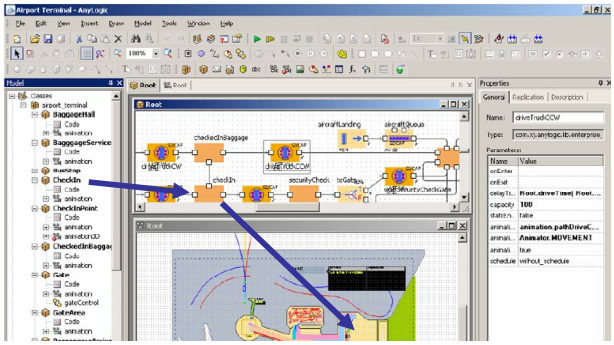

Figure 6: Model instantiation Anylogic.

The special feature of Anylogic is that the this simulation environment is not limited to libraries of objects for discrete event simulations. The simulation environment allows also the (re)use of object libraries that are using a different formalism, for example the Agent Based or system dynamics formalism.

3.6 Simul8

Simul8 is a workflow-based simulation environment, where work items are instantiated at work entry points, after which they claim resources, are processed at work centers, and stored in bins, before they leave the model at work exit points. Several advanced processing and transportation objects such as conveyors and tanks are also available (Haige and Paige 2004).

Each instance of an object is placed in the model by clicking it in the library toolbar, and placing it in the model. Many properties for each of the objects can be set through sub-screens of the main property screen of the object (Figure 7).

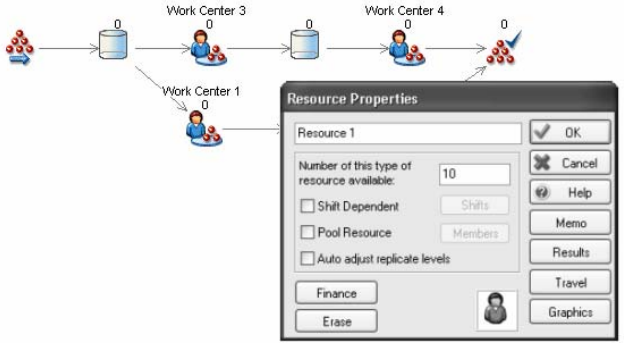

Figure 7: Model instantiation in Simul8.

3.7 eM-Plant

An SPP file of the simulation environment eM-Plant is a combination of a library and one or more simulation models that use model constructs of this library. A model construct in eM-Plant is called an object and it is either a descendent of a MU (movable unit), a frame (a screen in which you can compose a part of a model), or any of the provided basic objects for material flows. The model developer extends an initial SPP file, i.e. a library with the most generic model constructs, and adds to this file new frames as simulation models.

The model developer can instantiate available objects in the simulation model via either drag and drop from the object library, or via selection of the Application Object Library (see the thick blue arrows in Figure 8). The Application Object Library is a structured set of icons all representing one object from the object library.

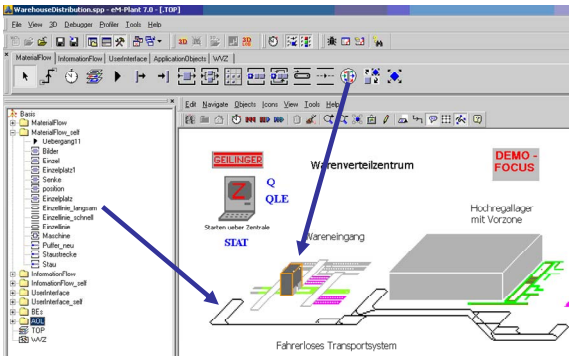

Figure 8: Model instantiation eM-plant.

Objects that are instantiated into the simulation model are children using inheritance of the parent object in the object library. The model developer can define in the instantiated object which events, methods or attribute values are inherited from the parent. The model developer has thus also the opportunity to change the behavior of a model construct that is instantiated in the simulation model.

3.8 SimScript

Model development in SimScript is done by typing code. SimScript uses text files that define objects. These objects are dynamically instantiated in the simulation model and represent an element in the system. The combination of a set of text files is a set of objects and thus a domain specific extension. Figure 9 shows for one of the objects its internal logic and shows that it uses the object "Port" to simulate the waiting of ships (Rice et al. 2004).

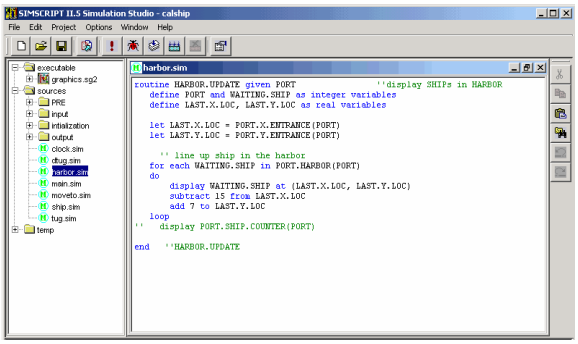

Figure 9: Model instantiation SimScript.

SimScript provides several solutions for domains in which they previously have performed projects. For example, the web-site of SimScript <www.simscript.com> shows that they provide solutions for 6 different aspects of healthcare modeling. Each of these solutions is a set of text files that can be used accordingly to dynamically instantiate a (part of a) hospital.

3.9 Enterprise Dynamics

Enterprise Dynamics is a simulation environment where models are composed of atoms. According to the documentation, everything in the environment is an atom and can be used to develop a simulation model or even a complete application that uses some of the underlying simulation technology. A simulation model in Enterprise Dynamics consist of one library in which the atoms are defined. The combination of library and simulation model is called a suite.

Enterprise Dynamics offers several suites for specific domains. For example, the Airport Suite consists of atoms for representation of passengers, baggage and airplanes and the Logistics Suite is developed for logistic systems like warehouse, high speed cranes and forklifts.

A simulation model in Enterprise Dynamics is composed by instantiating an atom into the 2D model view This instantiation can be performed from the library tree or via one of the icons in the menu. Figure 10 shows two ways of instantiating the SOURCE atom in a simulation model. An atom instantiated into a simulation model will inherit the behavior of the atom in the library and initially also shows the same parameter values.

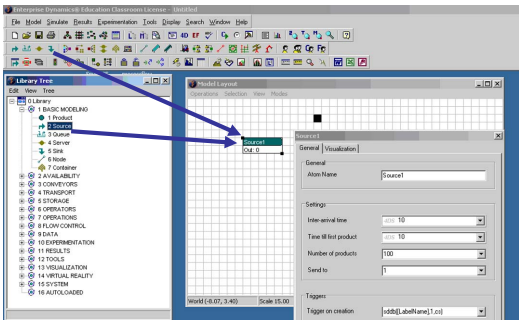

Figure 10:Model instantiation Enterprise Dynamics.

3.10 Flexsim

In Flexsim, models are created by dragging and dropping model objects from a library (left hand side of Figure 11) onto the 2D or 3D model workspace. After that, the modeler can set and modify the objects' characteristics. The standard library contains both discrete objects (including transporters, robots and cranes) and objects for fluids (tanks and pipes).

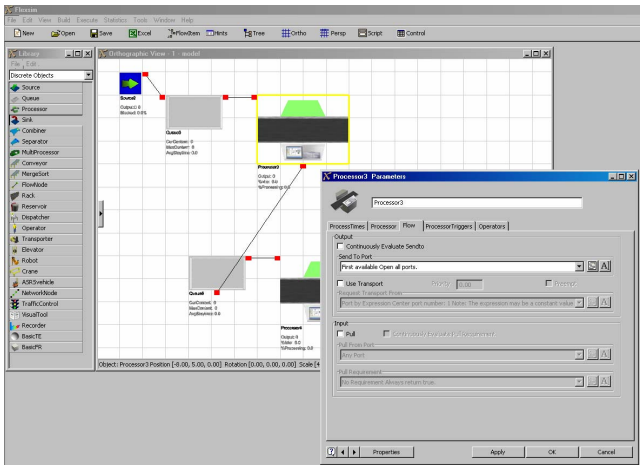

Figure 11: Model instantiation in Flexsim.

4 DEFINE CUSTOM DOMAIN SPECIFIC EXTENSIONS

4.1 Arena

Arena provides the possibility to model developers to develop their own custom domain specific extension. The development of a new template starts from scratch. Arena does not allow to extend templates developed by other model developers. In a new template, model developers make a list of new modules and provide these modules with an interface, logic, visualization and an icon. Figure 12 shows four of the parts that are used to develop a new template. The logic that is represented by a custom module can be developed from other templates, but Arena advices to develop a template from the basic templates that are provided and not extend custom developed templates.

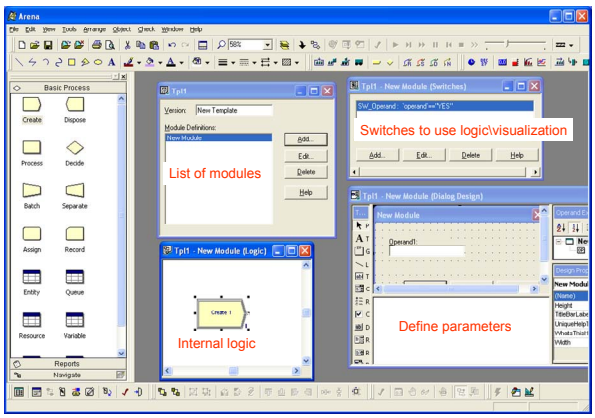

Figure 12: Develop new template in Arena.

Arena has a dedicated manual that explains how custom templates can be developed. This manual has a description of over 300 pages that helps a model developer to develop a new template. The manual extensively discusses all the different features and possibilities for a template developer.

4.2 Promodel

Promodel (version 4.2) does not offer possibilities to model developers to develop their own locations, resources or entities.

4.3 Automod

Automod (version 12) does not offer possibilities to model developers to develop their own systems. Advanced users of Automod have developed one standard logic file that include functionalities which they reuse in different studies, but this is just a workaround and not supporting new model developers to easily develop a simulation model.

4.4 Extend

Extend offers the ability to adjust every object and make duplicates of all existing objects in existing libraries. The behavior of an object can be defined in text logic. Figure 13 shows a part of the configuration of an object of the Discrete Event library. At the left hand side is a long list of all events and state variables that the object in Extend provides. In the left top corner a visualization of the object is shown once it will be instantiated into a simulation model.

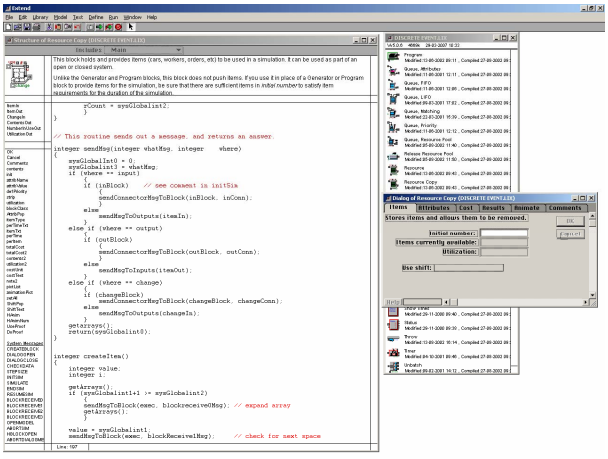

Figure 13: Adjust existing object in Extend.

If an object is changed by adjusting the logic or events of the object, this has effects on the behavior of the simulation models in which the object is instantiated. The attribute values will not be adjusted, unless a new attribute is added to the object. In that case the attribute will have the default value.

The Extend libraries consist of two files. If one of the files is not provided then a model developer can only use

the objects to instantiate the model but not adjust or view the contents of the objects.

4.5 Anylogic

The user's manual of Anylogic contains the quote: "A library is actually a project, which is compiled and packed into a Java™ archive. Any project can be made a library if you specify its property *Target file* and build it." (XJ Technology 2005, p388). This means that every model developer that instantiated one of more new objects in a simulation model can use these new objects as a library in a new simulation study.

Thanks to Anylogic's extension of the Java programming language, it is fully compatible with Java object libraries. If someone developed a Java library that provides some features that could be useful in a simulation model, then this library can be added to the use of the model developer.

4.6 Simul8

Simul8 has several features for creating and changing libraries. The first step that users can perform is the creation of sub-windows, where logic in the model is grouped. Secondly, components can be created. Components can be built from SIMUL8 standard objects, or from other components. Specific dialogs can be added to the component. When the component is finished, it can be saved as a new object on the toolbar, and used in any simulation. Component series can be easily distributed to other users to help them create their models. The Simul8 website offers several component libraries that can be downloaded. According to Simul8, components truly help co-operation between model developers, and ease the distribution of model knowledge from earlier models. Components can display or hide their inner logic (see Figure 14).

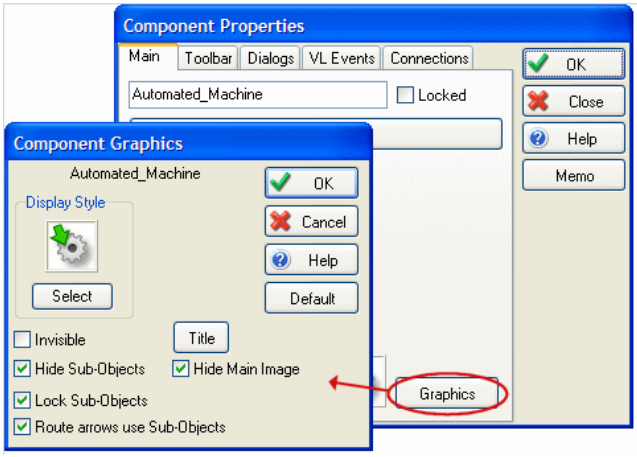

Figure 14: Hiding or showing, locking or unlocking sub-objects of a component in Simul8.

Another way to share modeling knowledge is through the use of templates. Templates are completely configured simulation solutions that modelers can use off the shelf, or expand and modify to suit their requirements. Templates can be easily shared, and the Simul8 website offers several templates from the company itself and from users.

4.7 eM-Plant

Defining new objects in the object library is almost an obligation when using eM-Plant. A new object is created by filling an empty frame with new methods or other existing objects in the object library. Figure 15 shows an object that consists of two methods and some lists. Each object that is developed can receive its own visual representation by making a drawing in one or more icons of the object.

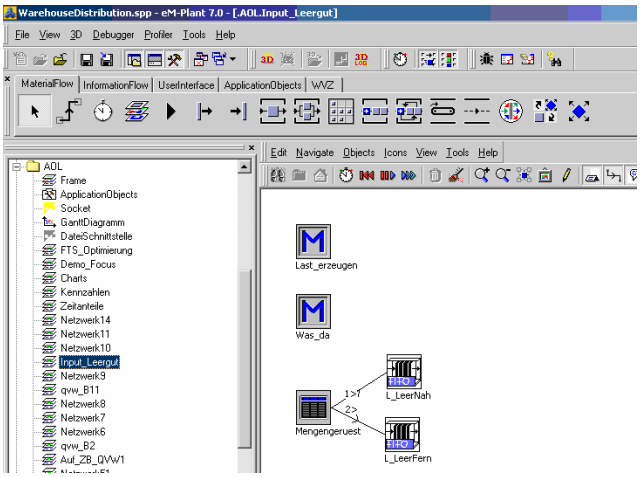

Figure 15: Instantiation new object in eM-Plant.

New objects that are part of the library can be derived from existing objects or duplicated. If an object is derived in the library it inherits all settings and logic, including the values in the user interface. If an object is duplicated, then it does not have an inheritance relation to the original, but it will keep an inheritance relation with the parent of the copied object. Whether inheritance is applicable or not can be observed in the user interface by small square blocks. Figure 16 shows the inheritance of a process duration after deriving and duplicating. If required a model developer can turn on the inheritance by selecting the appropriate box.

The objects in an object library can be structured for a model developer in the application object library. The model developer can use the library to develop several simulation models within the same SPP file. Unfortunately, it is not possible to keep the object definitions of the library separated from the simulation models. If a change is required for one of the objects, for example if an error has been identified, then this has to be adjusted separately in all files that use these objects.

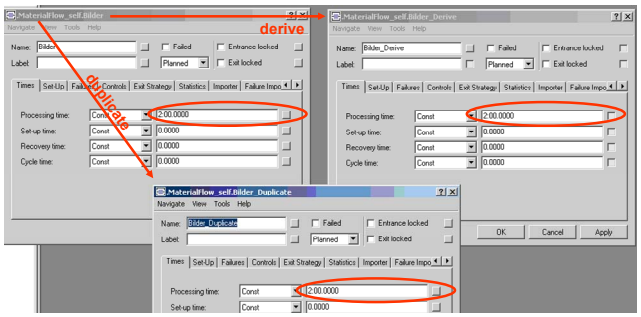

Figure 16: View status inheritance of parameters via boxes in eM-Plant.

4.8 SimScript

SimScript can use all files that are located in a certain directory as parts of the object library. A new library is thus created by moving some files that contain object definitions that already have been developed.

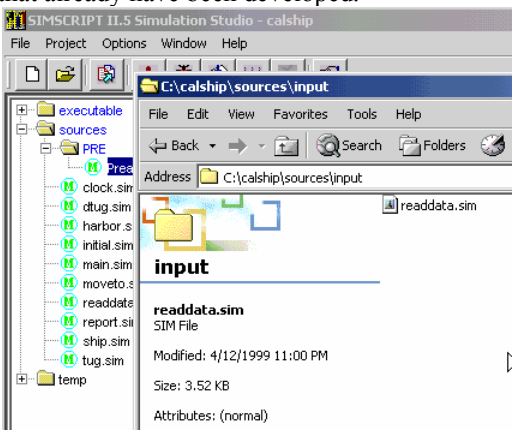

Figure 17: Custom classes to be added to SimScript.

4.9 Enterprise Dynamics

The atoms in the libraries of Enterprise Dynamics can be derived or duplicated and all settings of a module can be adjusted via the atom editor. If an object is derived, then the events inherit from the parent. The inheritance is shown by a small circle instead of the line of code of 4D-script for the event. Within the derived atom the developer can change the inheritance of the event and create custom 4D-script event logic for the event. Figure 18 shows a derived atom with one adjusted event with a specific 4D-script.

The user interface for atoms in Enterprise Dynamics is implemented by a user event that is triggered by double clicking on the icon in the simulation model. At that moment specific code is executed, for instance to open a file that contains the user interface. Enterprise Dynamics offers a special instrument to create user interfaces (see Figure 19).

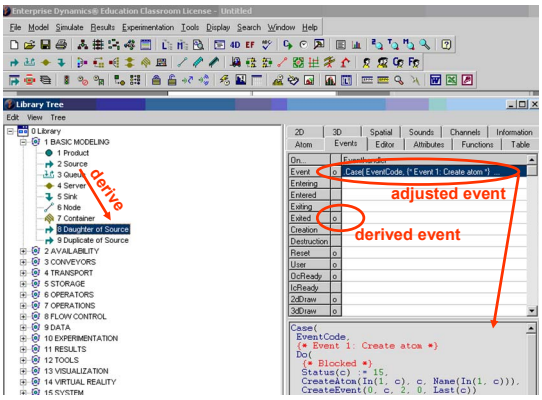

Figure 18: Atom editor and 4D-script in Enterprise Dynamics.

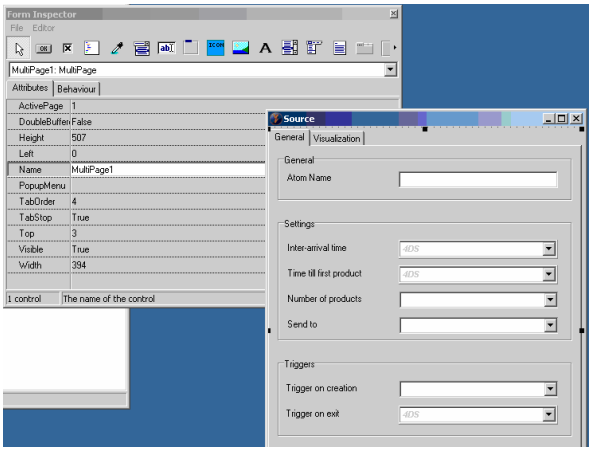

Figure 19: User interface builder Enterprise Dynamics.

4.10 Flexsim

It is possible to create user libraries in Flexsim. The custom library, or user library, mechanism provides flexibility in many areas. The most widely used functionality of the user libraries is to reuse customized objects in a model, but all kinds of additional features are available as well. All Flexsim libraries (including the standard library) are based on a tree model, where the composition of a new component is based on a combination of components and flexscript entries.

Flexsim only recently started with the official distribution of libraries; a container terminal library is now available for installation through the flexsim website.

5 OVERVIEW OF CAPABILITIES

We have identified 9 criteria to evaluate the suitability of commercial simulation environments for domain specific extensions. The evaluation of the criteria is whether or not the simulation environment provides the criterion. If the simulation environment matches the criteria, a weight factor is added to the score of the simulation environments.

Table 1 shows whether the simulation environments match the criteria and the weighted score for the 9 criteria.

The weight factors are based on the requirements for domain specific simulation environments as defined by Valentin and Verbraeck (2005) and the need for this feature to be provided by the simulation environment. The requirements are scored as very important (A, 8 points), strongly preferred (B, 4 points) or desired (C, 1 point). Obviously the key item is that model developers can develop domain specific extensions. Secondly, model developers need to be able to use the same concept for developing domain specific extensions as they apply in developing the simulation models, because a difference between the two concepts makes it hard for model developers to consider to develop domain specific extensions while they can have so many advantages during a simulation study. Enterprise Dynamics scores half, because are developed completely with 4D-script. Model constructs in the simulation model can be parameterized with this script, but is an extra not obliged to be used.

It is impossible for a developer of a domain specific extension to get the model constructs correct at once. Changes always need to be made to the model construct, but simulation models should not need to be redeveloped once a model construct is improved. The simulation environments Simul8, eM-Plant and Enterprise Dynamics score only half for this topic, because a model construct that is used in different simulation files needs to adjusted for each file.

Deriving is judged as being important, because it enables developers of model constructs to reuse already defined model constructs without having to start again.

Mainly for testing purposes it is handy that model developers can directly view the logic behind the model constructs and not have to go through a compiled version of the set of model constructs. The downside of this feature is that it offers model developers that are not aware of the internal working of the model constructs the possibility to make (undesired) changes. Therefore we inserted a criterion that the developer can protect the model constructs for changes of the domain specific extension.

In Table 1 we have not paid attention to all elements that are of importance in judging the suitability of commercial simulation environments to develop domain specific extensions. For example, item like support for defining structure, enable interfacing between model constructs, checks for semantic problems and the ability to protect use of the domain specific extension by model developers are not added. The reason is that none of the simulation environments offer these features, functionalities or documentation at this moment, and all would have scored zero points.

Table 1: Scoring of simulation environments for requirements (A=8; B=4; C=1. Maximum: 42)

Item	**Arena 11.0**	**Promodel 4.2**	**Automod 12.0**	**Extend 6**	**Anylogic 6**	**Simul8 2007**	**eM-Plant 7.6**	**SimScript 2.5**	**Enterprise Dynamics 7**	**Flexsim 4**	**Weight**
Model developer can create custom extensions	☑			☑	☑	☑	☑	☑	☑	☑	**A**
Modeling concept of model constructs is same as concept for instantiation of simulation models	☑				☑	☑	☑		½		**A**
Change model construct in extension after model development	☑			☑	☑	½	½	☑	½		**A**
Deriving of model constructs in extension				☑	☑	☑	☑		☑	☑	**B**
Change model construct in simulation model					½	☑	☑				**B**
Extension open for model developer				☑	½	☑	☑	☑	☑	½	**B**
User interface for parameterization of model constructs	☑			☑		☑	½		☑	☑	**B**
Ability to protect extension for model developer	☑			☑	☑	☑				☑	**C**
Commercial domain specific extensions exist	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	**C**
Score	**30**	**1**	**1**	**30**	**34**	**38**	**35**	**21**	**29**	**20**	

6 CONCLUSION

The analysis presented in table 1 shows that more than half of the commercial simulation environments offer support to developers and users for domain specific simulation environments. Several of these simulation environments claim to be object oriented and follow an object oriented approach, which makes concepts for reuse quite natural. In addition, a number of more "flow-oriented" packages offer very good functionalities as well.

Arena requires model constructs to be developed from scratch or by duplicating of existing model constructs. The advantage of Arena is that the same concept applies for the development of model constructs as for simulation models. Parts of simulation models can thus be reused in a model construct.

In the evaluation we only discussed technical features. It was surprising to us that none of the simulation environments offer any further support or guidelines in the design and development of the domain specific extensions. The simulation vendors seem to handle the development of domain specific extensions only as a technical feature to quickly instantiate a simulation model for a system. They do not seem to acknowledge that domain specific extensions are an excellent way to support simulation novices in new domains and thus enlarge the potential of discrete event simulation as was repeatedly mentioned during the panel discussions at previous Winter Simulation Conferences (Banks et al. 2001; Diamond et al. 2002; Barton et al. 2003).

ACKNOWLEDGMENTS

We want to express our gratitude to all referred simulation environments for providing us with (demo) systems and technical support while doing this research.

REFERENCES

Banks, J.; F. Azadivar; D. Ferring; J.W. Fowler; D.W. Halpin; A.M. Law; M. Manivannan; W.S. Murphy. "Panel session: the future of simulation" In: B.A. Peters; J.S. Smith; D.J. Medeiros; M.W. Rohrer (Eds.) *Proceedings of the 2001 Winter Simulation Conference*, pp.1453-1461, 2001

Bapat, V.; D.T. Sturrock. "The Arena product family: enterprise modeling solutions" In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, pp.210-217, 2003

Barton, R.R.; P.A. Fishwick; R.G. Sargent; J.O. Henriksen; J.M. Twomey. "Panel: simulation – past, present and future" In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, pp.2044-2050, 2003

Diamond, R.; J.O. Henriksen; C.D. Pegden; A.P. Walker; C.R. Harrell; W.B Nordgren; M.W. Rohrer; A.M. Law. "The current and future status of simulation software (panel)" In: E. Yücesan; C.H. Chen; J.L. Snowdon; J.M. Charnes (Eds.) *Proceedings of the 2002 Winter Simulation Conference*, pp.1633-1640, 2002

Harrell, C.R.; V. Lange. "Healthcare simulation modelling and optimisation using Medmodel" In: B.A. Peters; J.S. Smith; D.J. Medeiros; M.W. Rohrer (Eds.) *Proceedings of the 2001 Winter Simulation Conference*, pp.233-238, 2001

Harrell, C.R.; R.N. Price. "Simulation modelling using Promodel Technology" In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, pp.175-181, 2003

Haige, J.W.; K.N. Paige. *Learning SIMUL8: The Complete Guide, 2nd Ed*. Plain Vu Publishers, Bellingham, WA. 2004.

Krahl; D. "Extend: an interactive simulation tool" In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, pp.188-196, 2003

Nance, R.E. "A history of discrete event simulation programming languages" In: *ACM SIGPLAN Notices*, Volume 28, Issue 3, pp.149-175, 1993

Rice, S.V.; A. Marjanski; S.M. Bailey; H.M. Markowitz. "The SIMSCRIPT III Programming Language for Modular Object-Oriented Simulation". In: R.G. Ingalls; M.D. Rossetti; J.S. Smith; B.A. Peters (Eds.) *Proceedings of the 2004 Winter Simulation Conference*, pp.621-630, 2004

Rohrer, M.W. "Maximizing simulation ROI with AutoMod" In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, pp.201-209, 2003

Swain, J.J. "Power Tools for Visualization and Decision Making" In: *OR/MS Today*, Volume 32, Issue 6, 2005

Rockwell Automation. *Arena Shines at WSC—Again!* <www.arenasimulation.com>, visited 20-03-2007, 2007

Valentin, E.C.; A.Verbraeck. "Requirements for Domain Specific Discrete Event Simulation Environments", In: M.E. Kuhl; N.M. Steiger; F.B. Armstrong; J.A. Joines (Eds.) *Proceedings 2005 Winter Simulation Conference*, pp.654-663, 2005

XJ Technologies. *ANYLOGIC™ User's Manual*, St Petersburg, 2005

AUTHOR BIOGRAPHIES

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>.

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>.