Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

PROGRESS IN MODULAR SIMULATION ENVIRONMENTS

Charles R. Standridge
James F. Kelly
Thomas Kelley
Jack Walther

Department of Industrial Engineering
Florida A & M University - Florida State University College of Engineering
2525 Pottsdamer Street
Tallahassee, FL 32310, U.S.A.

ABSTRACT

How each simulationist can design and implement software tailored for each particular simulation project is addressed by modular simulation environments. The requirements of such environments are derived from the needs of four distinct types of users. Inter-tool modularity deals with how data flows between a non-homogeneous set of software tools that can be changed on an ad hoc basis. Both simulation specific and widely applicable software tools may be used. The organization and management of simulation inputs and results to achieve this goal is important. Intra-tool modularity has to do with supporting simulation project tasks in a modular fashion. Modular modeling is well established. Possibilities for modular animation and the modular use of widely applicable tools, specifically spreadsheets, are discussed. An example modular simulation environment is given.

1 INTRODUCTION

Ideally, each simulationist would be able to select the set of software tools to use on each simulation project (Standridge and Centeno, 1994). The selection would be based on the particular requirements of that project. The tool set would contain both simulation specific tools such as model builders and simulation engines as well as tools with wide applicability such as word processors, statistical analysis packages, and spreadsheets.

Modular simulation environment concepts seek to provide the standard by which ad hoc collections of software tools can be used together to perform a simulation project. These concepts specify how simulation related data flows between tools in a general way so that heterogeneous software can work together. The concepts address how tools can be tailored for use on particular problems and how general purpose tools can be tailored for application in simulation.

Modular simulation environments are based on an approach similar to that of computer operating environments such as Microsoft Windows for personal computers and X-Windows for Unix-based work station computers. The windowing systems provide the design, structure and mechanisms for tool integration. Windows and X-Windows provide a user interface standard for software tools as well as standard mechanisms for sharing information between tools.

Some benefits of modular simulation environments are as follows:

1. High flexibility for end user selection of simulation software. This supports simultaneous use of simulation software from multiple software providers as well as locally developed tools.
2. Use of widely applicable software with which the end user is already familiar.
3. Inclusion of tools such as spreadsheets, word processors and presentation graphics generators that have not traditionally been a part of simulation environments.
4. Use of standard windows techniques for sharing information between tools.
5. Management and selective use of simulation inputs, results, and models.
6. Definition of a standard for simulation environment structure and user interface. Because of its flexibility, end users may generally adopt such a standard and tool builders may find developing software compatible with its requirements helpful.

This paper describes the design and initial implementation of modular simulation environments as well as an example application. Requirements for modular simulation environments are based on their four different types of users. These user types are derived from those proposed by Standridge and Centeno

(1991). The flow of information between tools is described. The use of modular modeling concepts to tailor simulation specific and general purpose software tools for particular applications is discussed. Modular modeling for network simulation languages is reviewed (Standridge, 1995).

2 SIMULATION ENVIRONMENT OVERVIEW

Traditional simulation environments have three major components: (1) a fixed set of software tools prescribed by the environment designers and implementers, (2) a database management system that transparently to the environment users controls the flow of data between the software tools, and (3) a user interface that gives access to all environment capabilities. Each software tool uses existing information in the database and adds the results of its own operations to the database.

TESS (Standridge, 1985; Standridge and Pritsker, 1987) was an early, pre-graphical user-interface, simulation environment built on this strategy. Software tools provided for building SLAM II network models, editing sets of SLAM II control statements, constructing animations, making statistical computations, graphing data, and reporting data. Simulation results could be collected automatically from SLAM II, GPSS/H, and MAP/1 simulations. A command language served as the user interface. Selective querying of the database was supported. A database subprogram library allowed computer programs to store and retrieve data from the TESS database. The database manager organized simulation inputs and outputs.

Balci and Nance (1987, 1992) have used a similar strategy in designing and implementing a simulation model development environment. This work has lead to the visual simulation environment (Balci et al., 1995). A visual user interface supports the development and simulation of visual models that may be hierarchical and object oriented. The collection of tools includes a visual editor, a library of existing component models, a static model analyzer, a model dynamics tester, a simulator, an analyzer for simulation results, a multi-media learning support system, and an evaluator for accessing the credibility of a simulation study.

Centeno and Standridge (1991) describe an information based simulation environment constructed using the same approach. Its distinguishing features include the description of a manufacturing system of interest as information stored in the database. Alternative models can be generated by querying the database. In addition, a knowledge-based user interface is proposed to guide the user through the steps of a simulation project.

Modular simulation environments are distinct from these existing environments in that the set of software tools are not pre-determined, the use of widely applicable software is encouraged, and the user-interface relies on the standards set by existing graphical computer operating system interfaces. Database management must accommodate the open-ended nature of the environment.

3 USER TYPE REQUIREMENTS

Modular simulation environment capabilities are based on the needs of four user types whose relationship is given in Figure 1. The user types represent typical roles. In any particular situation, the same individual may take on multiple roles or many individuals may participate in the same role.

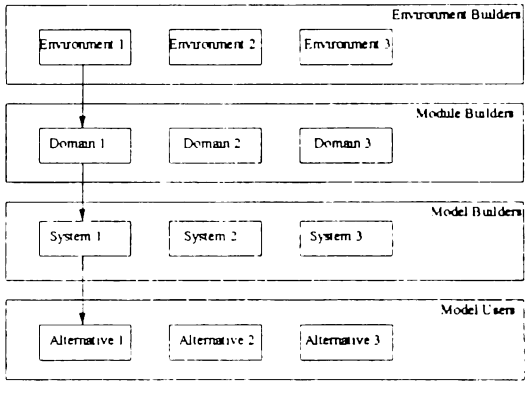

Figure 1: Modular Simulation Environment User Types

An environment builder gathers the software tools that form a simulation environment. The builder can construct multiple environments in this way. For example, an environment builder could select the SLAMSYSTEM simulation environment, the PROOF animation tool, and the EXCEL spreadsheet to comprise the environment.

In addition, an environment builder tailors the engine of any tool selected. SLX (Henriksen, 1995) and YANSL (Joines and Roberts, 1994 and 1995) are simulation engines. For example, the environment builder could construct a new transaction creation mechanism in SLX that reads the time of the next arrival and its attributes from a file as opposed to the traditional specification of a random or constant time between transaction creation. The extensibility capabilities of SLX would be used to define the new creation statement and encode the logic. The new

statement could then be used as would any other statement.

A module builder deals with one environment. The environment is tailored multiple times, once for each application domain of interest.

Consider a module builder concerned with discrete parts manufacturing. Using the capabilities of the simulation language selected by the environment builder, the module builder could construct modules for modeling individual work stations, material handling between work stations, serial lines and the like. Spreadsheet macros for processing observations of performance measure values could be written. Such macros could compute confidence intervals using the method of replicates, help find the truncation point in a steady state simulation, and graph simulation performance measure observations. Interfaces to these capabilities would "speak the language" of the domain. Thus, the simulation environment would more closely "use the language" of the application domain.

A model builder is concerned about multiple systems in one domain. Using the domain specific modeling modules provided by the module builder, a model builder constructs a simulation model of any particular system in the domain. Other tools tailored by the module builder for the domain can be further tailored by model builder for the system. For example, a tool for graphing time series of observations could be labeled as graphing the number of parts in a buffer. The range of the number of batches considered in computing a confidence interval from one series of observations could be specified.

A model user evaluates multiple alternatives about one system. The alternatives are described using input data to the simulation model constructed by the model builder. Other tools tailored by the model builder are used to examine, compare, and analyze performance measures resulting from simulations. For examples, spreadsheet macros can be used to graph performance measure values and perform statistical analysis.

4 INTER-TOOL MODULARITY

One type of modularity in a simulation environment has to do with how data flows between tools. This is referred to as inter-tool modularity. It must accomplished in such a way that new tools can be added to the environment on an ad hoc basis. No modification of existing tools can be required though the ability to export data and capabilities for tailoring existing functionality can be exploited.

Figure 2 shows the tool set of the first demonstration modular simulation environment. Existing commercial tools are shown in capital letters. Both simulation specific and widely applicable tools are included. Tools developed to achieve inter-tool modularity are shown in lower case.

Modular Simulation Environment Demonstration		
SLAMSYSTEM		PROOF
EXCEL		ACCESS
Inputs	Result Collection	Result Processing
Tool Attachment	Tool De-attachment	

Figure 2: Demonstration Modular Simulation Environment Tool Set

SLAMSYSTEM is used for modeling and simulation activities. It is interesting to note that another simulation environment can be used within a modular simulation environment. PROOF provides animation capabilities. EXCEL spreadsheets are used for model input value entry and for processing simulation results. ACCESS databases can be used to organize, manage, examine, and present simulation results. A text editor helps with input value entry.

Template-based tools have been developed specifically for organizing, specifying, and optionally entering model input values within a modular simulation environment. Multiple data sets can be input to a model. A template or format is defined for each input data set. A list of templates needed for a particular simulation is maintained. A template tool provides for defining the format of one input data set, listing the input data sets needed for simulation input, and entering values.

Simulation result collection can be specified. Generic tools for organizing and collecting results are provided. The capability to extract values from each simulation language of interest must be coded. This has been done for SLAM II.

Results processing has to do with making simulation results ready for input to other software tools. It includes joining together results collected from the simulation of multiple alternatives. This facilitates the comparison of alternatives.

Finally, a new tool can be attached to the environment. This involves describing the input data formats it accepts and the format of the output data it

produces if these are non-standard. Standard formats include text (ASCII) files, spreadsheet files, and database files. Existing tools may be removed.

Multiple data sets can be input to a model. A template or format is defined for each input data set. A list of templates needed for a particular simulation is maintained. A template tool provides for defining the format of one input data set, listing the input data sets needed for simulation input, and entering values.

Figure 3 shows the flow for input data. The TEMPLATE tool is used to describe the data. Values can be entered using any one of a variety of tools, including a spreadsheet such as EXCEL, a text editor such as NOTEPAD, or the TEMPLATE tool. This illustrates how a variety of tools can be employed to accomplish the same purpose in a modular simulation environment. Which tool is employed depends on each particular user.

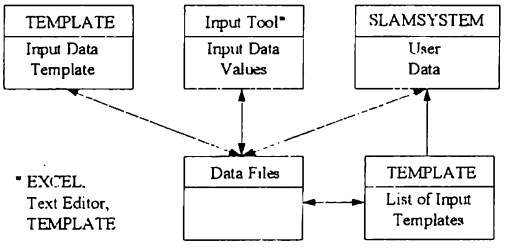

Figure 3: Flow of Simulation Input Data

The TEMPLATE tool also is used to define the data input requirements of a simulation model. This definition is a list of the templates for which values must be supplied in order to run the simulation. Running a simulation requires creating a list of the specific input data sets, one corresponding to each required template. This list defines the simulation input for one particular simulation run. Again, this input data list is created using the TEMPLATE tool. The input data list and the input data files are supplied to SLAMSYSTEM in its standard form for user data.

Within an input data set, values can be organized into rows and columns. Each row gives new values for the variables represented by the columns. Alternatively, each row can correspond to a different variable. A prompt at the beginning of the row specifies the input requirements.

The TEMPLATE tool is used to define the simulation results collected. Multiple result sets can be gathered for each simulation run. A template is given for each set. The template shows the name of the performance measure as known in the simulation result set and the corresponding name in the simulation language. For example, the performance measure, #INBUFFER, could correspond to the SLAM II function NNQ(1). A list of the simulation result sets to be gathered is created using the TEMPLATE tool.

Figure 4 shows the flow of information for the collection of simulation results. Each result template is defined using the TEMPLATE tool. A list of the all of the result templates desired for each simulation run is prepared. Based on the result template list, the result collector interacts with a simulation run to gather the performance measure values of interest. The result collector consists of a generic part and an interface to a particular simulation engine. Results can be stored in an ACCESS database for further processing.

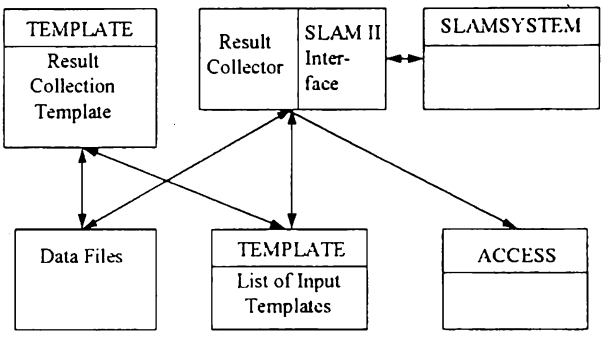

Figure 4: Simulation Result Collection

Result processing has to do with preparing simulation performance measure values gathered by the result collector for processing by other tools. Values of different performance measures from one alternative (different result sets) or values of the same performance measure from different alternatives can be organized together. EXCEL and PROOF can be used to analyze and display performance measure values.

Result processing is summarized in Figure 5. The TEMPLATE tool is used to specify which simulation results from which alternatives are of interest. The result processor produces a file of these result which be input to another tool. Data files collected from the simulation can be used directly in other tools if no re-organization of the simulation results is needed. This show the flexibility of a modular simulation environment is meeting user requirements.

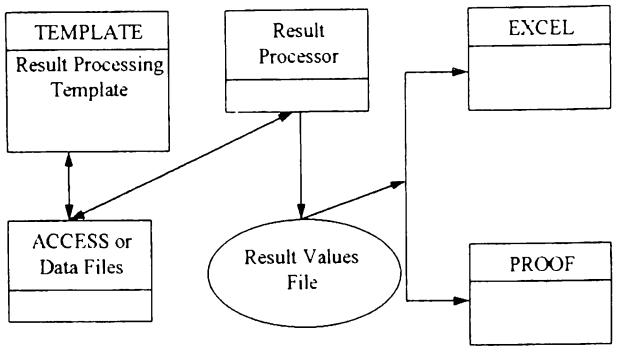

Figure 5: Simulation Result Processing

The organization of input values and simulation results is a fundamental aspect of achieving inter-tool modularity. Input data are labeled as follows:

Scenario -- What alternative is being described.
Variable -- Identifying name of an input quantity.
Value -- As specified by the model user

Note that there are multiple scenarios, versions, possible for each input data set. For example, there could be an input data set for the routes of parts through a job shop. Each scenario would represent a different possibility for routing jobs through the shop.

In addition, that there are multiple input data sets. Thus, a simulation alternative is defined by a model user by specifying the scenario of each input data set to be employed.

Performance measure values resulting from a simulation are labeled as in the following way.

Scenario -- What system alternative was simulated.
Type of values -- Time-persistent, observed, trace, end of simulation only.
Replicate -- ID number
Variable -- Identifying name of an performance measure.
Time or time interval -- When observed
Value -- As observed

For example, the following data sets could be collected: queue lengths, resource utilization, part time in the system, and throughput. The first two are time persistent, the third observed, and the last end of simulation only.

For result processing, the utilization of a particular resource and the corresponding queue length could be joined in a single data set. Alternatively, the queue lengths corresponding to a particular resource from each of two scenarios could be joined for comparison of the two alternatives.

5 AN EXAMPLE MODULAR SIMULATION ENVIRONMENT

Abrams, Standridge, et al. (1995) describe a simulation model of local caching policies. Files are retrieved via the World Wide Web to support distance learning. Caching allows the files to remain local, avoiding multiple retrievals of the same file. Caching policies tell what files to replace when a new file tries to enter a full cache.

A modular simulation environment supports this model. There are two input data sets: simulation options and experiment specification. Typical simulation options are the number of replications of each experiment, internet transmission time for files, and the name of the file listing the names of the files giving the files transmitted on the internet. This input data set in organized with a different variable and its value on each line. The experiment specification includes parameters such as the cache size, the replacement policy, and the parameters of the replacement policy. It is organized in a tabular format with one experiment per row. Columns correspond to experiment parameters.

The TEMPLATE tool is used to describe the two files. The Windows notepad editor is used to enter values. This shows the flexibility of a modular simulation environment. Either the TEMPLATE tool, the notepad editor, or a spreadsheet could have used to enter the values. The model users are computer scientists who are comfortable using the notepad editor.

The primary simulation results are gathered at the end of the simulation run (type end of simulation only) into one set. Results include the experimental parameter values and performance measures such as the hit rate, the percent of files found in the cache when requested.

Statistical analyses, such as regression and ANOVA, are used to determine which of the experimental parameters significantly affect the hit rate. These statistical analysis capabilities exist within a spreadsheet. Since only one result set is collected, it is stored by the results collector in a file format acceptable for direct spreadsheet input.

6 INTRA-TOOL MODULARITY

The second type of modularity has to do with how each tool does its work. Modular modeling concepts have long been discussed and are well implemented. See Zeigler (1990), Cota and Sargent (1992), and Sanderson et al. (1992) for basic definitions, concepts, and example implementations.

Standridge (1995) discusses modularity of network simulation languages and describes a modular network language, ModNet. This paper defines the characteristics of a module and the communication between modules. Module parameters are passed to a receiving module via a construct that resembles calling a subprogram in a general purpose programming language. Performance measure values are made public, that is accessible to all modules. Signals are public quantities that indicate that an event of note has taken place in a module. Other modules may respond to that event.

ModNet places bounds on the common entities of a network language, transactions and resources. Transactions may not cross module boundaries. Transactions may not be cloned. The need for cloning is satisfied by using a different module. A resource is defined and controlled by one module. This module may pass a resource name as input to another module to use. For example, a module may define and control a worker resource. The worker may perform tasks at two distinct work stations. Each work station is represented by a module. The name of the worker resource is passed to each work station module by the defining and controlling module.

ModNet is being implemented in the SLX simulation engine. An environment builder uses the extensibility properties of SLX to define the fundamental modeling capabilities available in ModNet. A module builder develops ModNet modules from which a model builder constructs models.

Consider the possibilities for the use of modularity concepts in other tools. A model user would like a spreadsheet interface and functionality tailored to the system under study. This could be constructed as follows. The environment builder is unlikely to have the capability of changing the spreadsheet engine, its core functions. The module builder would write spreadsheet macros that generate graphs and perform statistical computations generic to the domain of interest. For a example, a spreadsheet macro could graph the number of parts in a buffer versus simulation time. Another macro could use the method of batches to estimate a confidence interval concerning the mean number of parts in the buffer. The model builder could further tailor these macros to refer to the specific buffers in the model and specify the number of batches used in the confidence interval computation. Thus, the model user could simply ask for the graph of the number in the drill press buffer and the confidence interval concerning the mean.

Figure 6 shows one possible organization for animating a simulation. An animation is generated by a script that specifies a time ordered list of changes to a scene starting with an initial frame. Changes include modification to object attributes such as color or movement of objects. This specification allows the animation engine to produce a set of frames that show the time dynamics specified by the script.

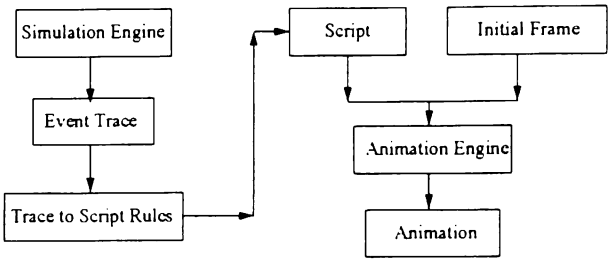

Figure 6: An Organization for Simulation Animation

The script can be based on the event trace produced by a simulation engine. A set of rules can be used to map the events of the simulation into a script that produces the animation of a simulation. In the case of a simulator such as ProModel, these rules are implicit in the specification of the model. In other words, the rules are embedded in the simulation engine. In the case of a simulation environment such as SLAMSYSTEM, the rules must be explicitly provided by a user.

Modular animation is achieved as follows. An initial frame is associated with each model module. The trace to script rules are specified. Thus, the animation for each instantiation of that module is specified. The combination of the animation of each module results in the animation of the model.

Animation of each module can be made optional. For example, an animation of the main module only gives a high level overview of the model. The animation can be expanded to include only those modules referenced by the main module to increase the level of detail. An animation of all modules shows all of the detail of the model.

7 SUMMARY

Current progress in the area of modular simulation environments has been discussed. Requirements for these environments arise from the four distinct types of users that take part in a simulation activity. Inter-tool modularity has to do with how data flow between heterogeneous tools. Intra-tool modularity is concerned with performing tasks in a modular fashion. Modular modeling is well established. Concepts of modularity can be applied to other tools such as spreadsheets and animators. The organization of simulation model inputs

and outputs in a modular simulation environment is discussed. An overview of a modular simulation environment for a simulation of local caching of World Wide Web files is given.

REFERENCES

Balci, O. and R. E. Nance. 1987. Simulation model development environments: a research prototype. *Journal of the Operational Research Society* 38:753-763.

Balci, O. and R. E. Nance. 1992. Simulation model development environment. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 726-735. IEEE, Piscataway, New Jersey.

Balci, O., A. I. Bertelrun, C. M. Esterbrook, and R. E. Nance. 1995. A Picture-Based Object Oriented Visual Simulation Environment. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 1333-1340. IEEE, Piscataway, N.J.

Cota, B. A. and R. G. Sargent. 1992. A modification of the process interaction world view. *ACM Transactions on Modeling and Computer Simulation* 2: 109-129.

Centeno, M. A. and C. R. Standridge. 1991. Modeling manufacturing systems: an information-based approach. In *Proceedings of the 24th Annual Simulation Symposium*, ed. A. H. Rutan, 230-239. IEEE Computer Society Press, Los Alamitos, California.

Henriksen, J. O. 1995. An introduction to SLX. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 502-509. IEEE, Piscataway, N.J.

Joines, J. A. and S. D. Roberts. 1994. Design of object-oriented simulations with C++. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 157-165. IEEE, Piscataway, N.J.

Joines, J. A. and S. D. Roberts. 1995. Design of object-oriented simulations in C++. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 82-89. IEEE, Piscataway, N.J.

Sanderson, D. P., R. Sharma, R. Rozin, and S. Treu. 1991. The hierarchical simulation language HSL: a versatile tool for process-oriented simulation. *ACM Transactions on Modeling and Computer Simulation* 1: 113-153.

Standridge, C. R. 1985. Performing simulation projects with the extended simulation system (TESS). *Simulation* 45: 283-291.

Standridge, C. R. and A. A. B. Pritsker. 1987. *TESS: The Extended Simulation Support System*. New York: Halsted Press.

Standridge, C. R. and M. A. Centeno. 1994. Concepts for modular simulation environments. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 657-663. IEEE, Piscataway, N.J.

Standridge, C. R., 1995. Modular modeling for network simulation languages: concepts and examples. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 736-743. IEEE, Piscataway, N.J.

Ziegler, B. P. 1990. *Object oriented simulation with hierarchical modular models.* New York: Academic Press.

AUTHOR BIOGRAPHY

CHARLES R. STANDRIDGE is an associate professor in the Department of Industrial Engineering at the Florida A&M University - Florida State University College of Engineering. He led the development of the Simulation Data Language (SDL) and of The Extended Simulation Support System (TESS) for Pritsker Corporation. His current research interests are modular simulation environments and health system engineering. He is co-developing a simulation model of local file caching of internet files as well as models of new family therapy processes.

JAMES F. KELLY is a graduate student in the Department of Industrial Engineering at Florida A&M University - Florida State University College of Engineering. He is writing his masters thesis on inter-tool modularity in modular simulation environments.

THOMAS KELLEY is a graduate student in the Department of Industrial Engineering at Florida A&M University - Florida State University College of Engineering. He is writing his masters thesis on intra-tool modularity in modular simulation environments focusing on spreadsheets and animation.

JACK WALTHER is a graduate student in the Department of Industrial Engineering at Florida A&M University - Florida State University College of Engineering. He is writing his masters thesis on intra-tool modularity in modular simulation environments focusing on the implementation of ModNet in SLX.