04041 Abstracts Collection
Component-Based Modeling and Simulation
— Dagstuhl Seminar —

Fernando J. Barros[1], Axel Lehmann[2], Peter Liggesmeyer[3], Alexander Verbraeck[4] and Bernhard P. Zeigler[5]

[1] Univ. de Coimbra, PT
barros@dei.uc.pt
[2] Univ. Bundeswehr München, DE
[3] HPI Potsdam, DE
[4] TU Delft, NL
a.verbraeck@tbm.tudelft.nl
[5] Univ. of Arizona, Tucson, US

Abstract. From 18.01.04 to 23.01.04, the Dagstuhl Seminar 04041 "Component-Based Modeling and Simulation" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. System-theoretic definitions and foundations for model- components, specification of model components, hierarchical, model-based model development cost-benefit, quality, performance, relibility, and reusability aspects

A Component-Based Approach to Modeling and Simulation of Hybrid Systems

Fernando J. Barros(Univ. de Coimbra, PT)

The Heterogeneous Flow System Specification (HFSS) is a comprehensive formalism aimed to represent a large variety of systems including switching systems, hybrid systems and digital controllers. The HFSS is a component-based formalism enabling hierarchical and modular component composition. The explicit representation of structure makes possible to change it dynamically, providing support for describing component mobility. By offering a common ground to model a large variety of models, the HFSS formalism enables the representation of complex systems in a sound framework.

A Framework for configurable Fault Tolerance in HLA Simulations

Steffen Großmann (Fernmeldeschule des Heeres - Feldafing, D)

The absence of fault tolerance mechanisms is a significant deficit of most current distributed simulation in general and of simulation systems based on the high level architecture (HLA) in particular. Depending on failure assumptions,dependability needs, and requirements of the simulation application, a choice of different mechanisms for error detection and error processing may be applied.

In this presentation we introduce a framework for configurable fault tolerance in HLA simulations. Fault tolerance methods can be selected and configured for each individual federate. The administration and execution of fault tolerant federations is supported by a configurable runtime environment. We present an adaptation process for integrating fault tolerance to existing federations. Prototypes of certain parts of the framework have been implemented. Furthermore We address the question whether dependability properties should be part of component descriptions.

Features vs. Components - How to reconcile the two Structuring Mechanisms?

Maritta Heisel (Universität Duisburg-Essen, D)

The goal of the talk was to stimulate discussion on two questions:

- What are commonalities of and what are differences between software components and model components?
- Can the notion of a feature be useful for the evolution of model components?

In particular, we discussed the role of component models in component-based software engineering and its possible counterparts in the area of simulation.

We introduced the notion of feature as it was coined in telecommunication and pointed out the problem of feature interaction.

We compared the notions of components and features and pointed out how to use them in combination. In the discussion, it turned out that realizing evolution by adding features saves less effort in simulation than in software development.

A component-based simulation architecture

Jan Himmelspach (Universität Rostock, D)

If a model shall be executed in a parallel, distributed instead of a sequential manner, typically the entire simulation engine has to be exchanged.

To adapt the simulation layer more easily to the requirements of a concrete model to be run in a specific environment a component based simulation layer has been developed for James. A set of different simulator components demonstrates that a component-based design facilitates the exchange of simulators and their combination.

Joint work of: Himmelspach, Jan; Uhrmacher, Adelinde M.

Challenges of Component Interoperation in Military Simulations

Marko Hofmann (Univ. der Bundeswehr - Neubiberg, D)

Simulations of military combats belong to the most complex models in use today. In order to manage their complexity over a comparatively long time (up to two or three decades) component based approaches are among the most promising. However, experiences with combat simulation systems in several NATO countries have shown that interoperability is not only a matter of technical and syntactical issues. Semantic and conceptual problems of model interoperation are much more challenging and they defy easy solutions. Two of these challenges are the problem of pragmatic differences and conceptual mismatch. Pragmatics deals with the actions triggered within an information processing entity (human or artificial) after receiving and semantically understanding a given message. In simulation models pragmatics can be seen as the real world interpretation of the model dynamics. Because of the huge variety of possible abstractions from reality these pragmatics can differ significantly. Thus, even with standardized semantic glossaries meaningful interoperation can still fail. Raising this problem from the level of executable or formal components to the conceptual level, helps to overcome some of the pragmatic problems but it also puts a new challenge on interoperation: It is well known that some concepts that are used to model "military reality" are not compatible to each other, for example Lanchester differential equations and single shot probability models for attrition or as a second example grid terrain models and vector based models. To overcome these problems, it is - up to now - always necessary to fine tune the models (or model components) to each other. The only solution that can be imagined with technology available today seams to be the strict standardization both on the level of pragmatics and concepts. However, that would significantly reduce the degree of freedom in modeling modern warfare.

Component-Based Safety and Reliability Modelling

Bernhard Kaiser (Hasso-Plattner-Institut - Potsdam, D)

Development by reusable components is a promising approach in the Embedded Systems domain. In this domain safety and reliability analysis are essential parts of the development process.

This generates a demand for compositional safety and reliability models. It should be possible to attach safety models to the technical components identified during the system development phase. We present Fault Tree Analysis (FTA) as one of the most popular safety analysis models and point out the aspects that do not fulfil the above requirements: The current modularisation by independent subtrees is not appropriate for components with complex interactions. The FTA semantics is not able to represent some facets of software-controlled systems, such as time dependencies or multi-state components. Moreover, current FTA tools do not support the collection of repeating structures in libraries for reuse. We propose three steps to deal with these issues: First, a new decomposition for Fault Trees allows defining arbitrary components that are connected by ports. Next we add state/event semantics to FTs. The resulting State-Event-Fault-Trees are particularly suited for software- controlled systems. Finally we show how both concepts together allow identifying patterns, preconfigured solutions to recurring structures in safety critical systems. We complete the picture by some examples and an outlook on our FTA Tool UWG3.

Keywords: Components, Embedded Systems, Fault Trees, Safety, Reliability

Joint work of: Kaiser, Bernhard; Liggesmeyer, Peter

Simulation of Systems that Contain Simulating Components - Problems and their Solution, Application, Classification and existing Models

Eugene Kindler (Charles University - Prague, CZ)

The paper will concern simulation of man-made systems S containing computers that simulate. Let such a simulation be called nested simulation. The simulating computers often simulate system S in which they are, or a very similar system S*. In such a case we speak on reflective simulation. It is a special case of the nested simulation.

Problem of implementation of models in case of nested simulation: almost all known simulation systems have no tools for handling two more time axes. The solution: using programming languages that are object-oriented, block-oriented and process-oriented.

Special problem of implementation of models in case of reflective simulation: the programming error (called "transplantation"), which consists of mixing element of both the existing models (e.g. to put a component of one model into a queue existing in anther model) makes problems to be discovered. The solution: using SIMULA, as it is secure against transplantation.

Special problem concerning SIMULA: it seems to be so secure against transplantation that it might not allow any communication between different models. The solution: there are some tricks that allow the communication and are secure against transplantation.

The problems and their solution will be presented in details.

The author implemented some models or essentially participated at their implementation:

(1) Simulation models of queuing systems containing dispatcher(s) who control the systems and use simulation to anticipate the quality of their decisions.

(2) Simulation models of container terminals that apply simulation to anticipate possible crashes and dead-locks.

(3) Simulation models of conveyors with rollers, which use simulation models to solve various problems concerning complicated situation that can occur when there is a lot of objects moving at the conveyor.

(4) Simulation models of personal urban transport where the imagining of passengers was modeled by their proper simulation models.

(5) Simulation model of systems that use "fictive simulation" (fictive simulation is a very rewarding implementation of complex non-simulation routines so that instead of a conventional algorithm a simulation model of a fictitious system is applied that generate the same data as the algorithm).

(5) represents nesting simulation which is not reflective, (1) - (4) represent reflective simulation. The models will be described in details, some of them demonstrated at PC.

SIMULA allows to simulate systems that use several simulating computers. Moreover, these computers themselves can perform nested (reflective) simulation. Such a nesting can be iterated. Therefore the nested simulation demands to be classified. A proposal for classification will be presented.

Also after having known the ways to solve the problems mentioned above, the implementation of nested simulation models is a hard task and their programming must be decomposed into several "world viewings" that permit the programmer to formulate the general concepts used in such world viewings and to specialize them in a stepwise manner. For example, in case of (3) we accessed 12 steps of world viewings. We have an experience that sometimes the world viewing can be placed at several computers of a network. The world viewings represent an interesting methodology, namely in case of nesting models: beside other, three levels "eternal", "external" and "internal". The eternal level corresponds to the simulation studies, the external to the simulation experiments handled directly in the simulation study by the objects of the eternal level, and the internal level corresponds to the particular experiments with the nested models; the external level functions sometimes like the a certain eternal level for the nested models, but in other cases the models existing at the external level can contain their proper level that functions similarly as a certain eternal level for them.

Nowadays, Ostrava University together with Blaise Pascal University in Clermont-Ferrand work at a software system that would allow to transform conventional simulation software written in SIMULA to software able of reflective simulation. The software is almost finished, one must complete tools that make it robust against sophisticated programming steps allowed by SIMULA; they will surely not be frequent but they are logically possible.

Simulation Model Components for Multi-Agent Simulation

Franziska Klügl (Universität Würzburg, D)

Multi-Agent Simulation provides a rather new modelling paradigm. A system is conceptualized as a multi-agent system in a simulated environment. An agent can be seen as an entity that is capable of flexible, interactive behavior in reaction to environmental input or for pursuing its selected goal. A multi-agent model consists of a formalization of the autonomous agents themselves, the environment they are "living" in and their interactions. The prerequisite for the dissemination of this model concepts is also based on the facilities for structured design and model component reuse.

This is an interesting issue, especially because agents should be able to flexibly interact due to their intelligence or adaptation.

Thus hard-wired links will be found only in special cases of multi-agent simulation. Thus composing a model from model components can be an non-trivial task.

On the other side concepts from the Multi-Agent Systems focus on interoperability of independently developed agents. The FIPA specifications can form here the most relevant contribution for handling complete agents as components as they tackle the syntax and semantics of agent communication. Thus the reusability of an agent component can be based on an adhereance of interaction protocol, message understanding and actual service provision. For using simulated agents as components multi-agent simulation the situation can not totally be solved in an analogous way as with fine grained agent simulation the costs of integrating this infrastructures are too high. Components at the sub-agent level, like e.g. planners using for software-agent construction can also be used for building simulated agents. As a last possibility of identifying components in multi-agent simulation models partial models that are providing model elements that are related to agent-agent or agent-environment interaction are tackled. These can be treated as sets of depending components. A short example illustrates this.

Dealing with Uncertainties in Component-Based Models

Johannes Lüthi (FHS Kufstein Tirol, A)

Often, at the time when a model or model component is built, some of its input parameters are not known exactly. This can be for a variety of reasons, such as the system itself not having been completed, a benchmarking process not yet having been carried out, or that precise estimates for the parameters are very difficult to obtain. Another situation where uncertain parameters may occur is the use of bounding techniques for intermediate solutions within a hierachical model.

In these situations, interval values or fuzzy numbers can be used to express the uncertainties of the input parameters. Such parameter representations offer

the potential to use a model component over a longer period within the lifetime of the real system. Intervals as well as fuzzy numbers can easily be adapted to less uncertainty up to the point where real numbers can be used when the model is no longer subject to significant uncertainties.

In previous work, various solution techniques for models with analytical solutions have been adapted to interval and fuzzy number parameters. In this talk, an overview about such adaptations is presented. However, the main purpose of the talk is to raise questions if and how such methods can be applied to model components that are not used within a framework providing analytical/mathematical solutions.

I.e. (how) can formalisms for component-based modeling be adapted to handle e.g. intervals or fuzzy numbers? What is the impact on the composed model and its solution techniques? How can a simulation kernel deal with interval/fuzzy parameters? Are optimization techniques based on interval arithmetic still feasable? What are the options for verification and Validation of components with parameter uncertainties?

Addressing Cross-Cutting Concerns: Aspect-Oriented Programming in the .NET Component Framework

Andreas Polze (Hasso-Plattner-Institut - Potsdam, D)

Besides design and implementation of components, software engineering for component-based systems has to deal with component integration issues whose impact is not restricted to separate components but rather affects the system as a whole. Aspect-oriented programming (AOP) addresses those cross-cutting, multi-component concerns.

AOP describes system properties and component interactions in terms of so-called aspects. Often, aspects express non-functional component properties, such as resource usage (CPU, memory, network bandwidth), component and object (co-) locations, fault-tolerance, timing behavior, or security settings. Typically, these properties do not manifest in the components' functional interfaces.

Within our presentation, we discuss the usage of aspect-oriented programming techniques in context of the .NET framework. We study the fault-tolerance, migration, and presistency aspects and discuss the expression of non-functional component properties (aspects) as C# custom attributes. Our approach uses reflection to generate proxy objects based on settings of a special attributes for C# components.

We have implemented an aspect-weaver for integration of aspect-code and component-code, which uses the mechanisms of the language-neutral .NET type system. Therefore, our approach is not restricted to the C# language but works for any of the .NET programming languages.

Software, Security Implementation - and Modeling?

Thomas Santen (TU Berlin, D)

Component-based software engineering today is a relatively mature discipline. Classical correctness of an implementation with respect to a specification is a compositional property. As such, component-based software engineering accommodates correctness concerns well.

Emergent properties, such as performance, safety, or security, of software systems are, however, not compositional: composing a system of "secure" parts in general does not yield a secure system. My research is concerned with finding ways of adequately specifying security properties and accomodating reasoning about security in component-based systems.

The discussions during the workshop have shown that the relation between component model descriptions and model components is similar to the one of emerging property specifications to software components.

Establishing a component-based modeling process, therefore, issues similar to the ones apparent in, e.g., security software engineering, will arise: - matching of component descriptions - adequacy of a component description for a model component - validation and verification of a composed model is, in general, not a simple consequence of validated and verified model components.

Component-based Simulation using HLA

Steffen Straßburger (Fraunhofer Institut - Magdeburg, D)

Simulation components are building blocks describing static and dynamic properties of a system (attributes, behavior). They can range from a fine granular building block inside a COTS up to an executable simulation model. They should be self contained and have some well defined interface to communicate with other simulation components.

Simulation components can be hierarchically put together to form a simulation model which itself can be considered a simulation component.

Motivation for the use of component based approaches is the promise of time and cost reduction. The idea of reusing existing components instead of building a new monolithic single-purpose model each time sounds very appealing. The other promise builds on increased maintainability, because components can be tested individually and should provide a welldefined interface to other components (although here is a difference to component-based software engineering: simulation components are highly context sensitive and can therefore not easily be tested in standalone- fashion).

Simulation models are typically developed in COTS simulation systems. Characteristics of COTS are that they have closed architectures and that they do not provide access to source code of simulators. Their only interfaces to the outside world are typically C and socket interfaces.

The objective for component-based simulation must be to enable simulation components developed in both the same and in different COTS to interact. Towards this objective the methods known from component based software development are not straight forward applicable. However, component-based simulation can learn from the experiences in the software engineering field, esp. in ways to describe semantics and dependencies between components.

One of the major differences of simulation components compared to software engineering components is that they need not only exchange data at runtime but also need to synchronize their local simulation clocks. Therefore typical component architectures (COM, CORBA, EJB) are not directly applicable for simulation; instead the use of HLA must be considered.

The possible role of HLA in the area can be as follows. Inside a simulation system components are put together hierarchically to a simulation model. The simulation model itself can be considered a top-level component which is executed in a runtime environment (the simulator). In the same way this simulation model together with its runtime environment can form an HLA federate. This can be considered a component, too. Multiple federates are also formed together to builder a federation (our top-level component) which is executed in its runtime environment (the HLA runtime infrastructure).

HLA can thus form the missing link to allow component based simulation across different COTS.

Component-Based Modeling and Simulation Tools

Michael Syrjakow (Universität Karlsruhe, D)

A great challenge of computer-based modeling and simulation (M&S) is the still rapidly growing model complexity and heterogeneity. To be able to develop distrib-uted simulation applications of high complexity quickly and for a reasonable price powerful M&S tools are required. In this presentation an open component architecture for M&S tools is pre-sented which is independent from the underlying modeling technique and which allows the integration of legacy M&S tools. Beyond that, it can be flexibly extended by experimentation com-ponents which have gained great importance because experimentation goals like finding op-timal or sensitive model pa-rameters cannot be reached by hand any more. The main focus of this presentation lies on integration concepts for M&S tools. Beside methods for in-tegra-tion of tools based on the same modelling technique also concepts for coupling of het-eroge-neous M&S tools are presented. Such tools which allow the modeller to deal with so-called multi-paradigm models are heavily demanded today. The presentation ends with an overview of existing multi-paradigm M&S tools and an outlook for future de-velopments.

Conditions for model component reuse

Mamadou Kaba Traoré (Université Blaise Pascal - Aubiere, F)

As Modeling and Simulation activities grow from modeling in the small to modeling in the large, model reuse becomes a key issue. Indeed, modeling is an objective-driven activity, and using an existing sub-model to build a larger one requires that the smaller model be checked for being able to achieve objective-based requirements of the larger one. A key problem is the capturing of these requirements in a way that automated (or semi-automated) semantic-based verification processes can be performed. Trying to achieve this goal, we introduce the OOAC paradigm. Generalizing the concept of experimental frame introduced by the multifaceted modeling framework, and formalizing the model/frame duality, we try to link the composability and reuse problem to the frame applicability problem, i.e. when does a model be suitable for an experimental frame. Our answer falls into three satisfaction conditions:

Objectives satisfaction, Constraints satisfaction, and Assumptions satisfaction. We specify for each condition a boolean satisfaction function, as well as a way to define an interval of confidence in the case of partial satisfaction. A simple and academic case study is used all along the work to illustrate the propositions. We believe that the formal background proposed here gives a new insight in model reuse and much more hard-to-achieve concepts, though not yet implemented in a software environment.

Joint work of: Traoré, Mamadou Kaba; Zeigler, Bernie

Experiences with building blocks for discrete event simulation

Edwin Valentin (TU Delft, NL)

In the last 6 years I have been developing sets of building blocks for discrete event simulation in different domains, among them automatic transportation systems, passenger flows at airports, baggage handling systems, container terminals and supply chains. These sets have been used by experts in those domains to support problem owners that are dealing with issues of resource allocation. The main results were that the ideas of composition of simulation models had advantages, but short comings are observed with relation to usability, maintainability/extendability and trust. Improvements to the concepts have been made by using concepts from systems engineering and software engineering. These concepts have been used to provide a structural support for designers of building blocks and this structural support has been used for designing a set of building blocks for public transportation. This new set of building blocks has been used in three laboratory experiments were novices and experts in modelling had to make adjustments to a simulation model and perform a simulation study for a

fictive situation in the Netherlands. In this laboratory experiment participated 40 novices and 14 experts. The results were that modellers with building blocks had more success, both novices as experts. However, before experts really get some results, they need to be brainwashed from their first initial concepts about how a certain domain can be represented.

Building Blocks in Simulation

Alexander Verbraeck (TU Delft, NL)

The BETADE project at Delft University of Technology (Building blocks for Effective Telematics-based Development and Evaluation) focused on developing new theories for building block based development in a number of domains, among which modeling and simulation. Other application domains were document handling, GIS, and business engineering. Within BETADE, a definition of building block or component has been used that emphasizes the interfaces of building blocks, their clearly defined functionality, and relative independence of other building blocks. The definitions used is:

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

When we look more closely at the role of building blocks or components in simulation, we can identify three different types of actors or roles: the building block developer, the model builder who uses the library of building blocks, the model user or analyst who uses the resulting model to carry out experiments, and the problem owner who is interested mostly in the results of the simulation study. The interesting challenge is that in the use of a building block library, the information flows from the building block developer to the problem owner. When designing the building block library, the information has to come from the problem owner(s) to the building block developer. Creating a generic building block library is difficult because the genericity that the building block developer might want, is often not valued by the problem owner. There are many other challenges for building blocks for all the roles.

The overall challenge on all levels is the description of the functionality of the building blocks, not only at the syntactical level, but especially on the semantic and pragmatic levels. For the syntactical level, an XML dialect might be sufficient. For semantics, we could use metamodeling and ontologies, but these are hardly available in the modeling and simulation field. Pragmatics, really important for the simulation study, are even harder to describe.

For defining simulation building blocks, we can and should learn from other fields. One of these fields is software engineering, another is distributed simulation. Recently, Web services have also grown in importance, and the componentized structure and service offering look really interesting for the definition and creation of simulation components.

As a result of more component-based development of simulation, the simulation field and applications will change. Instead of building models from the ground up, we will see an assembly of components to create a simulation model. This means that our simulation methods will also have to be changed. Instead of top-down development, our methods will have to deal with a combination of top-down and bottom-up approaches.

Component-based Development of Real-Time-Systems

Gabriel A. Wainer (Carleton University - Ottawa, CDN)

M&S techniques can offer significant support when designing complex applications. The Simulation-Driven Engineering approach is of particular interest for the development of embedded applications. The Simulation-Driven Engineering (SDE) [1] approach relies on simulation-based modeling for developing components of real-time systems. We have analyzed the feasibility of this approach using CD++, a modeling and simulation toolkit that is based on DEVS. DEVS is a sound, formal modeling and simulation framework, which allows hierarchical, modular model composition and component reuse.

We show how to use SDE to build real- time models incrementally, integrating hardware components with models simulated in CD++. By using different experimental frameworks, it is possible to analyze the execution of models in a risk- free environment, allowing one to check the modelŠs behavior and timing constraints. The proposed approach allows secure, reliable testing, analysis of different levels of abstraction in the system, and model reuse.

Testing and maintenance phases are highly improved due to the use of a formal approach like DEVS form modeling the system. DEVS provides a sound methodology for developing discrete-event applications, which can be easily applied to improve the development of real- time embedded applications. These advantages include secure, reliable testing, model reuse, and the possibility of analyzing different levels of abstraction in the system. Model execution is automatically verifiable, as the execution processors are built following the formal specifications of DEVS. DEVS bibliography shows how to build execution engines that enable mimicking the modelŠs behaviour in a homomorphic formalism. Hence, the developer only needs to focus on the model under development.

The concept of experimental framework eases the testing tasks, as one can build independent testing frames for each submodel. Closure under coupling eases this task, as models can be decomposed in simpler versions, always obtaining equivalent behaviour. Finally, the semantics of models are not tied to particular interpretations, thus existing models can be reused. Likewise, modelŠs functions can be reused by just associating them with new models as needed.

Currently we are developing embedded versions of RT-CD++ to run in an embedded platform (one running on the bare hardware, and the second version on top of RT-Linux). We are also developing a verification toolkit to use the timing properties of the DEVS models under development. In this way, we will

have an environment for SDE in which the user creates models, test them in the simulated environment, uses verification tools to analyze timing properties, and downloads the resulting application to the target platform, being able to provide rapid prototyping and enhanced development capabilities.

These results are also related with the current efforts on standardization of DEVS models. The objective is to study the possibility of developing standards for a computer processable representation of DEVS that supports common understanding, sharing and interoperability of DEVS implementations. Computer processable forms include all forms of simulation and real-time execution as well as various forms of syntactic and semantic analysis. This group will perform an analysis study for the potential establish-ment of a core for a DEVS standard. The actual standards development work for the standardization of basic primitive and compound DEVS modeling constructs (syntactic and semantic) in support of higher-level extensions such as agent, cellular, and dynamic simulation models. Information about this effort can be found in: http://www.sce.carleton.ca/faculty/wainer/standard/