A Conceptual Model for Reusable Simulations
Within a Model-Simulator-Context Framework

Abstract

Model reuse is a longstanding challenge within the simulation modeling community. While other disciplines successfully apply component-based approach to build systems, it has proven more difficult to apply in simulation development. A model is reusable to the extent that the assumptions and constraints of the experimental frame it is situated in are satisfied. Hence, without contextual information a conceptual model and its simulation may not effectively be used in a new experiment. While the value of an experimental frame for reuse is clear and well documented, one of the least appreciated, but significant aspect central to reuse is the characterization of the context of the model itself. To this end, this paper explores the extension of the basic model-simulator-experimental frame viewpoint to assert the role of context in reuse. Furthermore, the separation of model’s concept definition and its realization is emphasized to facilitate sound and formal reasoning for reuse. Based on that perspective model qualification is defined in a formal manner to conceptualize reusability in terms of formally specified behavioral dependencies.

1 Introduction

Given the significant benefits of reuse in terms of gained productivity (Basili et al. 1996) and the potential for increased quality, model reuse is widely promoted and encouraged, particularly within the DoD community (Davis and Anderson 2003). The significance and benefits of reuse libraries is discussed in (Ören 2002). Discussions on credibility of composable simulations are also of significant interest. For instance, (Weisel et al. 2003) elaborates on the validity of compositions in terms of formal metrics for semantic composability. A more general overview of the challenges and difficulties involved in model reuse are discussed by Overstreet et al. (2002).

To achieve progress in reuse, they suggest a variety of significant development areas, including capturing specifications of model constraints, objectives, and assumptions of a model. Similarly, Ören and Zeigler (1979) argue about the importance of high level model specification formalisms and their role in reuse, as well as maintenance. Reuse and model composition has been studied in qualitative simulation (Falkenheimer and Forbus 1991, Forbus 1984) and software engineering, where one important goal is to formalize the modeling process using an underlying domain model. Nayak’s 1995 ACM Distinguished Dissertation showed that the general model selection problem for application composition is NP-hard (Nayak 1992). Others have shown that deciding whether an identified collection of submodels meet a stated set of objectives is an NP-complete problem (Page and Oppen 1999).

Given the impracticality of complete automation of the composition of models to satisfy a set of objectives, the reuse community within the modeling and simulation community focused on developing infrastructures such as HLA (Dahmann et al. 1997), and product line architectural frameworks, such as ONESAF to facilitate reuse through common standards. Developing such common standards and
protocols enables technical interoperability (Dahmann et al. 1997) to the extent that models are developed to be conformant with these standards. Yet, compliance with standards does not necessarily qualify a model for effective reuse. Hence, substantive interoperability is raised as a significant challenge.

Key issues involved in enhancing reuse of simulations is discussed in (Davis and Anderson 2003) and (Overstreet et al. 2002). The fundamental issue raised in both of them is the need for emphasis on determining and resolving incompatibilities of objectives, assumptions, and constraints. The mechanisms and formalisms for capturing and explicitly representing such constraints, features, and semantics of components are argued to be central to determine substantive interoperability of simulations. To this end, formalization of the modeling and simulation relations with a sound conceptual view of reuse is critical. Furthermore, it is widely accepted that the separation of the notion of simulation implementation and its conceptual model is imperative to facilitate reasoning about models to assess their relevance for a new simulation study.

1.1 Model-Simulator-Experimental Frame

Figure 1 illustrates the basic entities in a modeling and simulation study using the popular Model-Simulator-Experimental Frame (MSEF) (Zeigler et al. 2000) viewpoint. The frame is defined as the set of conditions on a given simulation experiment in terms of the input output values and constraints that will be imposed and monitored.

![Figure 1: Model-Simulator-Frame](image)

A model is a representation of the system, entity, phenomenon, or process. Simulation is the act of using a simulator (simulation engine, or a behavior generator) to drive simulation models for generating the model behavior. While the above MSEF perspective facilitates intuitive understanding of the role of a model, simulator, and experimental frame in studying a system or phenomena, the meaning of their dependencies are not clear. That is, the interpretation of the modeling, as well as the simulation relationships are not formally defined to facilitate sound and formal decision making about the fitness of a model within a new experimental frame. Furthermore, the distinction between a conceptual and simulation model, which is critical for sound and formal reasoning about reuse and composibility, should be explicitly delineated.

1.2 The Problem: A Conceptual Model for Reusable Simulation Models

While the significance of distinction between model, simulator, and experimental frame for reuse is clear and well documented, one of the least appreciated, but most significant aspect central to reuse is the formalization of the context of a model and its software implementations. Szyperski (2002) defines a software component as follows:

“Software component: a unit of composition with contractually specified interfaces and explicit context dependencies only. “

Yet, the contextual information of a model and its potential simulation models are not captured within this basic framework. Our position is that unless an M&S practitioner (1) describes a model formally to facilitate reasoning about its fitness within a new experimental frame and (2) understands the model’s contextual dependencies accurately and unambiguously, model reuse will continue to be an ineffective trial-error effort. Hence, given this position, we explore the following:

- In the context of the presented M&S framework, to facilitate the realization and maintenance of precise behavioral relations among model abstractions, implementations, simulators, and the experimental frame, what kind of conceptual dependencies are of interest?
- Which formal modeling perspective and semantic domain are appropriate to characterize such dependencies?
1.3 Modeling the Context

The benefits of separating the model and its simulator is well known (Zeigler et al. 2000). We argue that by explicitly defining the context of the reusable simulation at the concept and realization levels, and formally defining its dependencies, a model can better be selected and adapted for reuse. Furthermore, given the significance of explicit characterization of the conceptual and behavioral constraints of a simulation model to improve reusability, we sharply distinguish between a simulation and conceptual model. A conceptual model is an abstract model and represents some aspects of reality that is delimited with reference to the goal of the study. A simulation model, on the other hand, is the computerized version of an abstract model or a conceptual model. More specifically, a model associated with a simulation study is expressed within a computer program and this computer code embodiment of a model is called a simulation model. Note that the same abstract model can be implemented by multiple simulation models.

Figure 2: Modeling the Context

Figure 2 illustrates the components of a reusable model. In addition to basic concept and realization aspects denoted by the conceptual and simulation model, respectively, a context dimension is introduced to formalize the environment of the model. A model can be implemented by different simulation models that implement the same abstract concept. Models (conceptual and simulation) are coupled with a context component to facilitate capturing the contextual constraints that an engineer needs to be aware of to reuse a model. To this end, the context defines the components with which a simulation model and its conceptual model has behavioral dependencies. The conceptual, realization, and experimental context relations, as defined below, are critical in specifying the context component.

The conceptual context relates the model abstraction to other concepts in the domain. More specifically, the relation defines how the semantics of the model relate to the semantics of other models in the domain of the experimental study. The realization context defines how the implementation depends on other concrete components for the implementation of the model within the experimental context. It provides a way of separating implementation dependencies with the direct goal of increasing portability. The experimental context defines the data components on which the simulation operates to be consistent with the experimental conditions under which the system is studied.

2 Behavioral and Contextual Dependencies

Reusing a model within a new experimental frame requires understanding the meaning of behavioral dependencies among the conceptual model, the simulation model, the experimental frame, as well as the models’ contextual dependencies. In the original framework, shown in Figure 1, the dependency between the model and its experimental frame is called the modeling relation, whereas the dependency between the model and its simulator is defined as the simulation relation. The meaning of these behavioral dependencies need to be clear and well-defined to qualify a model and simulator for a new experiment.

While other engineering disciplines successfully develop physical systems in a compositional manner from reusable parts, the same approach has proven to be difficult in developing software-intensive simulation models. A primary reason for this outcome is that models and submodels are often tightly coupled with each other, and that such contextual dependencies are often implicit. Clearly, some dependencies must exist, but to facilitate understanding a model for reuse requires making behavioral model and
simulation dependencies explicit. Such dependencies can be denoted by formalizing the conceptual behavioral component relations (Yilmaz 2000). The role of such component relationships is to express dependencies between components and, in doing so, to provide information about how simulation model components may and should be used in conjunction with other components in a new frame. Commonly used relationships within existing simulation languages include dependencies such as calls, passes data, moves, schedules, contains relationships that define the role of models within a particular context. That is, they do not help address how one might qualify and reuse a model to build a new experiment. More specifically, conventional dependencies do not describe the use of a component independent of its current context. Furthermore, they are language-specific and do not reflect a conceptual view of a reusable model. The component relationships defined here relate components which may either be abstract (conceptual models) or concrete (simulation models).

2.1 The Implements Relation

Implements relationship describes the key dependency between a concrete simulation model, and an abstract conceptual model. The implements relationship may be defined informally as follows:

A concrete component $X$ implements abstract component $Y$ if and only if $X$ exhibits the behavior specified by $Y$.

This is an intuitive relationship between a specification and its implementation. However, a fully formal and general definition of the implements relationship is very intricate and has been the subject of much research. Implements expresses a dependency between two components in the following sense. If component $X$ implements component $Y$, then $X$ depends on $Y$ to provide an abstract description of its behavior. If two different concrete components both implement the same model, then either of them may be used in a context requiring the functional behavior described by the model. In this case, the two implementations are behaviorally substitutable with respect to the specification they both implement. The two implementations may differ in non-functional characteristics such as execution time, space requirements, cost, warranty, legal use restrictions, level of trust in correctness, and so forth.

2.2 The Needs Relation

The relation, needs, actually applies to three different yet closely related component relationships. Needs may describe a dependency between two simulation models, between two conceptual models, or between a simulation model and a conceptual model. The last of these three relationships is defined as follows:

Concrete component $X$ needs abstract conceptual model $Y$ if and only if $X$ depends on the behavior specified by $Y$.

This form of the needs relationship expresses a polymorphic dependency between simulation models. Any simulation model that implements conceptual model $Y$ may serve as the actual concrete component used by $X$. Thus, this form of needs reduces unnecessary dependencies between components.

2.3 The Extends Relation

A third component relationship is extends. The term extends applies to two different, yet closely related, component relationships. One extends expresses a dependency between two conceptual models. The other expresses a dependency between two simulation models. Both extends relationships may be defined informally as follows:

Component $X$ extends component $Y$ if and only if all of the behavior described by $Y$ is included in the behavior described by $X$.

This definition conveys the intuitive meaning of abstraction refinement, that is, component $X$ can substitute the behavior of component $Y$. Hence, it implies the essential property of behavioral substitutability. If abstract component (conceptual model) $X$ extends abstract component $Y$, concrete component (simulation model) $X_1$ implements $X$, and concrete component $Y_1$ implements $Y$, then $X_1$ is behaviorally substitutable for $Y_1$ with respect to $Y$. The extends relation, being a behavioral substitutability relation, can be used to formalize the modeling relation of the basic framework.
That is, if the concept designated by the model extends the abstract specification denoted by the experimental frame, then the conceptual model and its verified simulation model can safely be used to deliver the required behavior.

2.4 The Simulates Relation

The dependency between a simulator and a simulation model is defined in terms of the simulates relation. A simulator uses a conceptual framework such as event-scheduling, process-interaction, or activity scanning to generate the behavior in terms of the computational instructions of a simulation model. A simulation engine (or a behavior generator, or a simulator) drives a simulation model, under the conditions specified within the experimental frame to generate model behavior. An event scheduling mechanism or a numerical integration algorithm to solve differential equations are simulation engines. The simulation relation can be defined as follows:

Simulator $X$ simulates (drives) a simulation model $Y$ within an experimental frame if and only if $X$ generates the behavior defined by $Y$ within the same experimental frame.

The simulator relation entails the correct realization of a simulation protocol that derives the computational behavior (i.e., event sequences) by executing (i.e., scheduling) the instructions (i.e., events) of simulation model.

In the next section we present how these relations can be instrumental in qualifying a model for a new simulation study. In particular, specific behavioral rules for the extends relation are presented to place the model qualification approach on a sound formal basis. Due to space limitations, the use of these relations in defining the contextual dependencies of models will not be discussed here.

3 Formalization of the Modeling and Simulation Relationships

3.1 Hierarchy of Conceptual System Specifications

Zeigler et. al (2000) presented a hierarchy of model specifications that can be used to capture the conceptual requirements associated with a model at different levels of abstraction. The I/O observation frame provides a structural specification for the system in terms of input output variables of interest. Formally, $IO = (T, X, Y)$, where $T$ is a time base, $X$ is a set of inputs, and $Y$ is a set of output. At the operational level these sets provide a means for specifying the structure of the input/output interface of a system. Extending this structural definition with input/output mappings help facilitate moving from the structural domain to behavioral domain. To capture the I/O behavior, the I/O definition is extended to obtain $I.ORO = (T, X, \Omega, Y, R)$, where $\Omega$ is the set of permissible input segments. Each segment can be interpreted as sequence of discrete inputs (i.e., events) over a time period. The relation $R \subseteq \Omega \times Y$ maps from input segments to output segments. A behavior, $b$, of the system can be defined as a point in the input output space, $b \in R$. Being an abstract representation of the system, the relation $R$ is a partial specification that can map a given input to more than one output. Being a partial specification, $R$, provides flexibility by enabling multiple implementations for the same model. The next level of specification in the hierarchy is the I/O System specification that provides a process-oriented viewpoint for the system under study. That is, the I/O System specification presents a process-centric viewpoint that characterizes the state-dependent interaction constraints of a model. Formally, $S = (T, X, \Omega, Y, Q, \delta, \Lambda)$, where $\delta: Q \times X \rightarrow Q$ is the global state transition and $\Lambda: Q \times X \rightarrow Y$ is the output function. Given these basic definitions, in the next section we will elaborate on the requirements and conditions for qualifying a model for reuse with respect to conceptual system specifications at the I/O behavior level by formalizing the extends relation introduced in section 2. Due to space limitations reuse at the I/O system specification level will be discussed elsewhere.

3.2 Reuse at the I/O Behavior Layer

Given an I/O observation frame, one can demonstrate syntactic conformance of a model to a system specification in a straightforward manner. Yet, the behavioral domain of the system and the model need to be considered to determine if their behavioral specifications are consistent; that is, if the model is behaviorally substitutable and reusable to satisfy the system
specification represented by the experimental frame. To simplify the illustration, we can define the I/O behavior designated by the experimental frame and the model in terms of a mapping from specification to sets of run-time behaviors, where the behavioral domain \( R \) is the relation \( R \subseteq \Omega \times Y \) that maps from input segments to output segments, as defined in section 3.1. To demonstrate that a simulation is qualified for an experimental frame one needs to establish the implements relation between the conceptual model and its simulation models. Then the conceptual model needs to be shown to extend the specification and behavioral constraints designated by the experimental frame. To establish the implements relation between a conceptual model and one of its simulation models, we need to prove that simulation \( S \) is a plausible implementation of conceptual model \( M \), and that its behavior is consistent with the abstract partial specification. A specification should ideally be non-deterministic to facilitate alternate realizations. It can be shown that an abstract model behavior can be realized by alternative simulation models by partitioning \( R \subseteq \Omega \times Y \) into a set of functions, by which an output is determined deterministically by its input. More specifically, one can define

\[
R = \bigcup_{k} \overline{R_{k}} ,
\]

where each \( \overline{R_{k}} \) can be specified by a function, \( f_{k} \). The set of functional groupings that capture \( R \) can be defined as

\[
F = \{ f_{1}, f_{2}, \ldots, f_{i}, \ldots \} .
\]

Now using this functional notation, the semantics of the conceptual model (abstract specification) can be defined as

\[
I_{M} : M \rightarrow P(F) ,
\]

where \( P(F) \) denotes sets of functions, each member of which captures a deterministic slice of the overall behavior depicted by the specification, as shown in Figure 3. Given this new definition, one can intuitively assign the behavioral domain of a simulation model to one of the members of this set. That is, we can define semantics (behavior) of a simulation model as

\[
I_{S} : S \rightarrow F .
\]

The implements relation now can be defined as:

\[
S \text{ implements } M \equiv I_{S}(S) \in I_{M}(M) .
\]

Once this relation is established, the next step is to qualify the conceptual model with respect to the system specification defined under the constraints of the experimental frame. Given two specifications \( I_{M} : M \rightarrow \rho(F) \) and \( I_{M} : M' \rightarrow \rho(F) \), the substitutability of \( M \) in place of \( M' \) can be defined using the extends relations as

\[
I_{M}(M) \subseteq I_{M}(M') ,
\]

where \( M' \) designates the abstract specification denoted by the experimental frame.

**Figure 3: Meaning of Conceptual and Simulation Models**

The formalization indicates that the model’s conceptual behavior extends the abstract I/O specification of the experimental framework. That is, the abstract I/O specification of the model is more strict or specialized compared to the behavior denoted by the frame. Hence, any simulation model that realizes the model concept also realizes the system abstraction designated by the experimental frame.

### 4 Conclusions

In (Davis and Anderson 2003, summary section p.13), authors argue that a fundamental need for building into agreed methods of model representation, the requirement that the model, simulator, and the context of reuse (experimental frame) be distinguished and specified separately. Agreeing with their finding, we suggested a perspective for capturing and formalizing the
conceptual dependencies among these fundamental M&S framework components. We also emphasized the significance of the separation of a conceptual model from its implementation to facilitate reasoning about reuse of a simulation at the conceptual level. Furthermore, we argued that the context of the conceptual models and their concrete simulation components is as important as the experimental frame to qualify and eventually deploy a new model within a simulation study.

References


