

# On the Need for Contextualized Introspective Models to Improve Reuse and Composability of Defense Simulations

**Levent Yilmaz**

Auburn Modeling & Simulation Laboratory of  
the M&SNet, Department of Computer  
Science and Software Engineering  
College of Engineering  
Auburn University  
Auburn, AL 36849  
[yilmaz@eng.auburn.edu](mailto:yilmaz@eng.auburn.edu)

Model reuse is a longstanding challenge within the defense simulation modeling community. While other disciplines successfully apply component-based approach to build systems, this approach has proven difficult to apply in simulation model development. The significance of employing explicit and distinct experimental frames to support reuse is already well recognized. Yet, characterization of the original context of a model is one of the least appreciated aspects and central to reuse and composability. Methods and tools are needed to provide mechanisms for structured representation and storage of context information, effective ways for retrieving it, and the possibility to relate it to the developer's intention. To this end, the basic model-simulator-experimental frame viewpoint is revisited to assert the role of context in reuse. The significance of separating concept, content, context, and simulator is discussed under the discrete-event system specification (DEVS) framework to provide a basis and rationale for contextualized simulation models. An abstract design strategy is presented to facilitate packaging and distribution of concept and context specification objects along with simulation models to improve reuse and composability.

**Keywords:** Composability, conceptual modeling, context, component-based, reuse

## 1. Introduction

Given the considerable productivity benefits of reuse [1], model composability is widely promoted and encouraged, particularly within the DoD community [2]. The Defense Modeling and Simulation Office's (DMSO) Composable Mission Space Environment's (CMSE) initiative, for instance, aims to identify issues related to composability to improve reuse [3,4,5]. In various military simulation domains (i.e., affordable human behavior modeling), where models are extremely costly to build and maintain, largely offsetting their potential economic advantages, reuse efficiency becomes a critical concern in achieving program objectives [6].

To achieve progress in reuse and composability, a variety of significant development areas, including capturing model constraints, objectives, and assumptions are recommended [7]. Similarly, Ören and Zeigler [8] discuss the significance of high level specification formalisms for reuse and maintenance. Various criteria for simulation model decomposition are suggested to help improve reuse efficiency in the design and development of military simulations [9]. Computational complexity of composability also is explored [10].

Besides the computational issues, various substantive reuse and interoperability challenges now are well recognized in federated distributed simulation design within the HLA infrastructure [11,12]. To help mitigate syntactic disparities among models, common operating environments [13] and product line architectures such as OneSAF are being developed to improve reuse and composability through standardization. Reuse and composability also has been examined in qualitative simulation [14,15,16] and software engineering [17,18,19], where one important goal is to formalize the compositional modeling process using an underlying domain model. Yet, while the software engineering community focuses on the development of deductive specification matching methods for software retrieval [20], simulation modeling requires taking the experimentation context, simulation goals, and objectives into consideration. That is a critical aspect of the process is the selection and assembly of models into viable combinations that work together within a simulation system to satisfy simulation study objectives. To this end, a recent workshop on the composable mission space environments recommended the development of a common conceptual modeling language that enables the development of tools that use the metadata framework to identify semantically composable components [21].

In general, the term composability is the quality of being composable and means to be capable or worthy of being composed. Capabilities for the selection and assembly of models can be improved by engineering composability directly into the simulation models via principled design precepts. The importance of distinguishing between concept (conceptual model), content (simulation model), and experimental frame is clear and well documented [8,22,2,23]. However, one of the least appreciated, but most significant, aspect central to reuse is the formalization of the original context of a model. The position advocated in this paper is that unless an M&S practitioner understands the model's contextual dependencies accurately and unambiguously, model reuse will continue to be an ineffective trial-error effort. To this end, this paper presents a strategy to improve reuse and composability via (1) separation of the concept, content, context, and simulator in simulation modeling and (2) explicit characterization and distribution of context information to facilitate systematic model qualification and model understanding for composition. The premise of the proposed strategy is based on the observation that to operate within context, models and tools are needed to provide mechanisms for structured representation and storage of context information, effective ways for retrieving it, and the possibility to relate it to intention. Given this position, we explore the following:

- To facilitate the realization and maintenance of precise relations among model abstractions, implementations (simulations), simulators, and the experimental frame, what kind of general and abstract dependency conditions are of interest? How can these dependencies be instrumental in capturing contextual dependencies of models?
- What is the best way to package and distribute original context information along with simulation models to facilitate retrieval, understanding, and reasoning about the suitability of a simulation model in a new context?

The rest of the paper is organized as follows. In Section 2 the role and significance of separating concept, content, context, and simulator are discussed. In Section 3 we overview related work on the development of context-aware computational artifacts. Section 4 discusses the elements of context and their interdependencies within a simulation modeling framework. The discrete-event system specification (DEVS) formalism is used as a basis to outline elements of context and their interdependencies. The notion of abstract behavioral component relationships is used to structure the contextual dependencies of a simulation model. In Section 5 a novel model design strategy that facilitates storage and retrieval of structured context information

is suggested. We conclude (Section 6) by discussing the utility and limitations of the strategy, as well as potential avenues for further research.

## 2. The Separation of Concept, Content, Context, and Simulator

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 to generate model behavior. A conceptual model is an abstract model that incorporates a specification (i.e., DEVS I/O system specification [22]) that 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 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. Experimental frame refers to the specification of the conditions under which the model is observed or experimented with. Figure 1 depicts these components as well as their interdependencies. While modularity has been widely acknowledged as an enabling factor for composability in software intensive system development [24], little attention is devoted to the role and significance of the separation of concerns with respect to these fundamental M&S framework components.

### 2.1 The Separation of Concept from Content

The separation, explicit representation, and reference to the concept (specification) components within a simulation model not only enable qualification of a model for reuse on the basis of its formal specification, but also facilitate substitutability of alternative simulation models that implement the same concept. While none of the existing simulation modeling environments provides language facilities to relate a simulation model to its specification, availability of such a formal specification (i.e., DEVS I/O system specification) in some form that is independent of the simulation model enables substitutability analysis between the context of the new experiment and the potentially reusable model component.

Distributing specification of a simulation model as part of the simulation software using embedded comments may not be practical since it requires discovery, location, and interpretation of such comments. Hence, explicit reference to a specification component within a simulation model representation would simplify the selection of a model for assembly during composition.

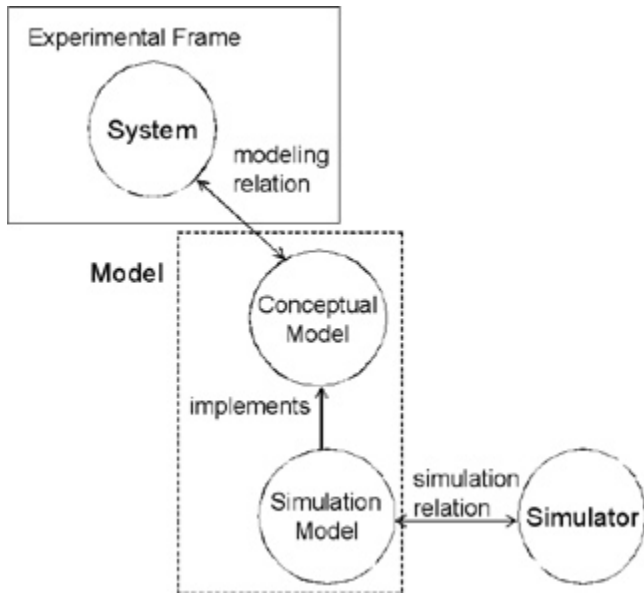


Figure 1. Model-simulator-experimental frame

### 2.2 The Separation of Simulator from Concept and Content

The significance of separating simulators from models is well recognized [22]. The simulator components in DEVS formalism [25], for instance, are separate entities that use the model components in the DEVS model hierarchy. The base model component that simulators use to derive behavior provides an abstract interface for creating families of related or dependent objects without specifying their concrete classes. Figure 2 depicts the structure of a simulation model hierarchy in relation to a simulator. Owing to the separation of the simulator from the simulation model, the simulation system can be flexibly decoupled from the knowledge of how its models are created, composed, and represented. That is, the simulator can be configured with one of multiple families of models (model family M1, M2). Given that a set of related submodels of a coupled model is designed to be used together and that we need to enforce this constraint, the associations between the coupled model and its submodels can be utilized to configure a simulator with a family of model objects (i.e., atomic models A1 and B1 are related to model family M1). The structure shown in Figure 2 isolates simulators from model objects. That is, simulators manipulate model objects through their abstract interfaces. Actual model references are isolated in the concrete coupled or atomic models; they do not appear in the simulator.

In this approach, replacing component models and thus reconfiguring a simulator with a new model assembly is simplified. By simply instantiating a different coupled model with the same abstract interface, the simulator can switch to a different assembly of model objects (i.e.,

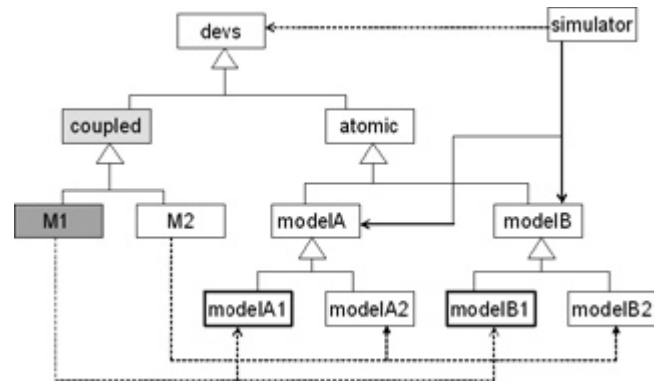


Figure 2. Decoupling simulator from simulation models

A2, B2). To assemble a selected set of components into a new composite does not require any modification within the simulator. By construction, the latter does not make assumptions about the model components, their structure, and potentially intricate details about their instantiation and process. This responsibility is rather delegated to concrete model family components (M1, M2) that are responsible for referencing and instantiating model components.

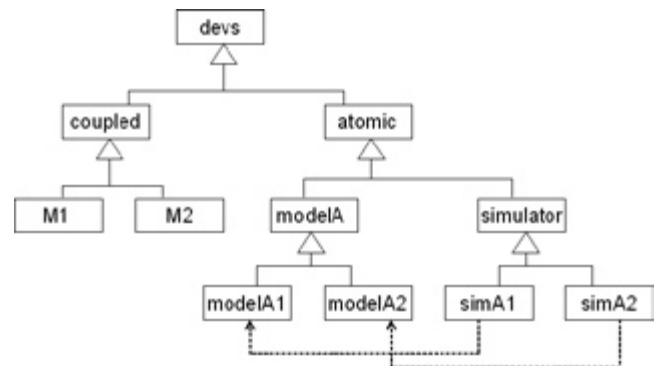


Figure 3. Flexible assembly of model components via simulator customization

Furthermore, simulators and coordinators used in object-oriented implementations of the DEVS formalism can be defined as a subclass of atomic DEVS [25]. This opens new avenues in terms of flexibility when adaptivity is needed to improve the assembly process. Simulators define an interface to create and instantiate model objects. By defining an interface for creating a model object but allowing its subclasses to decide which model object to instantiate, the configuration shown in Figure 3 can defer instantiation of model objects to specialized simulators. As such, when a simulator cannot anticipate the class of model objects to instantiate or requires specialized simulators to specify the model objects it creates and uses, the simulator specialization strategy can be used

to avoid hardwiring a set of model objects. Rather, the configuration of a set of model objects defined in the base simulator can be refined within the simulator hierarchy by flexibly instantiating different concrete model objects in place of the abstract ones to flexibly instantiate various composites that are consistent with the abstract composition style. Both of the above design strategies become useful when adaptability during composition becomes a concern in model design. As that happens, emphasis shifts from hardcoding a set of fixed behaviors toward defining a smaller set of behaviors that can be composed into a number of complex ones.

### ***2.3 The Separation of Context from Concept, Content, and Simulator***

The notion of an experimental frame and its underlying rationale suggests the significance of context in a simulation study. Often, the boundary between a simulation model and its environment is neglected even by experienced simulation practitioners. A simulation model does not run in isolation; its behavior is impacted by the behavioral dependencies of the context it is embedded, which in turn consumes the output of the simulation model. While constraints can be placed on the behavior of the simulation, developers often make implicit assumptions about a simulation's context. However, developing simulation models for reuse requires making such assumptions explicit to facilitate measuring the extent to which a new context is consistent with the original context of a model. Separation of context requires at least the following conditions: (1) the factors influencing simulation outcomes are separable, (2) useful distinction can be made between contextual factors and the others, and (3) contextual factors are capable of being recognized later on.

### **3. Related Work on Context**

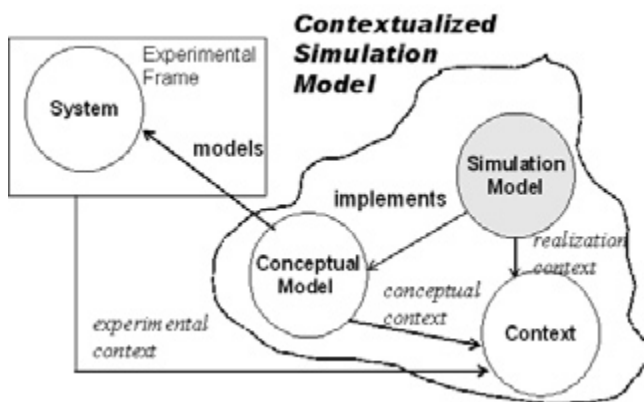
*Webster's Dictionary* defines context as (1) the parts of a discourse that surround a word or passage and can throw light on its meaning and (2) the interrelated conditions in which something exists or occurs. As such, the notion of context is a critical aspect in assessing the relevance of a model for reuse in a new experimental frame. Context and context-awareness are significant concepts that often are stressed in modern information systems development. The issue of context is recognized as a significant factor in the reuse and sharing of knowledge and information [26,27]. Early models for context representation come from Artificial Intelligence. Contexts mostly are represented in a logical notation.

Functions and predicates are given for each of the aspects (dimensions) of the environment. This results in a rule system having stand-alone context objects with concrete characteristics in their aspects. As a result, abstract and detailed requests for matching context information become feasible. The foundations of contextual reasoning are based on McCarthy's proposal [27], from which most of the ensuing logical formalizations of context derive their basis. The role of context in natural language processing [28] and common sense reasoning is substantial, as the specification of context has the effect of narrowing down the interpretive possibilities of communicated messages. Contextual information is used in a variety of domains other than reuse to improve effectiveness of information retrieval and personalization. Using web browsing context, for instance, is one of the key enablers for web site access personalization and intelligent search [29].

The significant role of context in ubiquitous and pervasive computing [30] as well as human computer interaction is widely accepted. Explicit characterization and capture of context could improve reuse and composability by representing the constraints under which a model is developed. Contextual similarity analysis can be performed to assess model qualification. Structural context has been used and analyzed in specific applications, such as bibliometrics, database schema-matching, and hypertext classification. The more general problem of finding similar objects has been studied in information retrieval and recommender systems, among other areas. Such methods can be used in model context similarity analysis. Computing similarity recursively based on structure also has been explored in the specific context of schema-matching [31]. Existing methods in reuse, however, mostly focus on composability and substantive interoperability issues based on the consistency among functional and informational profiles of components. The context and its impact on simulation reusability are not yet explored to sufficient depth. In model reuse, to enable understanding the contextual dependencies of a reusable model, metadata and tools are needed to provide mechanisms for structured storage and retrieval of context information. Availability and accessibility of context information enables comparing contexts or retrieving models with relevant contexts in qualifying a model for reuse. There exist other approaches that describe context as named, aggregated structure, where simple contexts consist of a descriptor-state-pair and complex contexts consist of aggregations of simple and complex contexts. These mainly are software application focused and difficult to generalize to a common framework.

#### 4. Exploring the Structure of Context for Reusable DEVS Models

A general framework of contextual dependencies is crucial to facilitate development of sound reasoning mechanisms for reuse and explore potential means to package such information with a simulation model for distribution. In section 4.1 we propose a multidimensional representation for context and discuss how the M&S framework presented in [22] can be augmented by a context component to improve specification of reusable simulations. Section 4.2 presents the elements of the context. The DEVS framework is used to formalize and illustrate the derivation and utility of context.



**Figure 4.** Contextualization of simulation models

We argue that by explicitly defining the context of the reusable simulation at the concept and realization levels, and then explicitly defining its dependencies, a model can better be selected and adapted for reuse within the constraints of a given experimental frame. Context can be viewed as the abstraction of those elements of the circumstances for which a model is developed, that are not used explicitly in prediction, but rather allow the recognition of new circumstances where the model can be usefully applied.

Figure 4 illustrates the plausible components of a reusable simulation model. In addition to basic concept and content aspects denoted by the conceptual and simulation model, respectively, a context component is introduced to formalize the experimental, implementation, and conceptual environment of the model. By doing so, models (conceptual and simulation) are coupled with a context component to facilitate capturing the contextual constraints that one needs to be aware of to reuse a model. The conceptual model, simulation model, and the experimental frame constitute the immediate elements of the context from which other models or model components are derived using the behavioral component relationships outlined in section 4.1.

#### 4.1 Capturing Dependencies among Context Elements in Terms of Behavioral Relationships

The role of behavioral component relationships presented below is to express dependencies between models and, in doing so, to provide information about how simulation models may or should be used in conjunction with other components in a new context. 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 the context they are currently being used. That is, they do not help address how one might qualify and reuse a model to build a new simulation. More specifically, conventional dependencies do not describe the use of a model independent of its current embedded context within a simulation program. As such, they are implementation specific. Furthermore, they are language-specific and do not reflect a conceptual view of a reusable simulation model. The component relationships defined here relate components, which may either be abstract (conceptual models) or concrete (simulation models).

##### 4.1.1 The Implements Relationship

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 behavior described by the model. In this case, the two implementations are behaviorally substitutable with respect to the conceptual model 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. While justifying the claim that a simulation model implements a conceptual model (specification) may require significant effort, especially if done formally, considerable leverage is gained from doing so. A formalization of the implements relation between DEVS conceptual and simulation models is presented elsewhere [32].

### 4.1.2 The Extends Relationship

The second 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 safe abstraction refinement, that is, component X can substitute the behavior of component Y. Hence, it implies the essential property of behavioral substitutability [33]. If abstract component (conceptual model) X extends abstract component Y, concrete component (simulation model) X1 implements X, and concrete component Y1 implements Y, then X1 is behaviorally substitutable for Y1 with respect to Y. The extends relationship, being a behavioral substitutability relationship, can be used to formalize the model qualification process. That is, if the concept designated by the model extends the abstract specification (i.e., DEVS I/O system specification) then the conceptual model and its verified simulation model can safely be used to deliver the required behavior. A formal treatment of the extends relation is beyond the scope of this article. The notion of substitutability [32,34] based on DEVS formalism presents a formal framework along with automated decision procedures to determine if a DEVS component is behaviorally substitutable for another structurally equivalent model based on their DEVS I/O system specification.

### 4.1.3 The Needs Relationship

The relationship, needs, actually applies to three different yet closely related component relationships [33]. Needs may describe a dependency between two simulation models, between two conceptual models, or between a simulation model and a conceptual model. The last one 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 model components. The

premise of this argument is that by avoiding concrete dependencies among simulation models, one can improve flexibility and reusability of a model. The needs relationship emphasizes the significance of capturing and explicitly representing import requirements of a model. That is, the assumptions and obligations of two DEVS components with respect to each other, as a result of the needs relationship among them, has to be consistent. The notion of horizontal consistency presented in [34] provides a formal strategy along with a decision procedure to determine if the interaction dynamics (DEVS I/O system specification) of two DEVS models are consistent.

## 4.2 Using Extends and Implements Relationships to Facilitate Selection and Assembly of Simulation Models

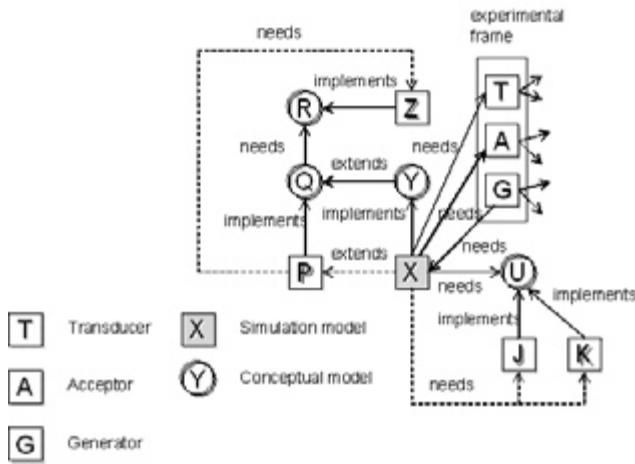
The context of a conceptual model and its simulation is critical in capturing the dependencies of a component before using it in a new simulation study. The behavioral component relationship, which are presented in section 4.1, can be used to express such dependencies to provide precise meanings useful during model composition, while reflecting a clear conceptual view of component-based simulation development process. Figure 5 illustrates these dependencies in a hypothetical scenario, where conceptual model (specification component) Y and one of its implementations, simulation model X, are considered. Identifying the conceptual, experimental, and realization context of a simulation model is critical in understanding what is needed to complete a model's definition within the new context. The design dependencies of a model are comprised of the dependencies of its abstract (conceptual) and concrete realization (simulation) components.

The conceptual, realization, and experimental context can be defined for each one of these dimensions of the context based on the behavioral component relationships presented in section 4.1. Next we discuss the derivation of contextual dimensions as well as their utility in model qualification and composition.

### 4.2.1 Conceptual Context

The conceptual context of a simulation model, X, is defined in terms of the dependencies of the conceptual model (specification component) it implements. That is, given that the specification component of simulation model X is represented by the conceptual model, Y, the conceptual context of X is defined by the conceptual dependencies of Y. Conceptual context is defined as the set of specification components (i.e., DEVS I/O behavior) to which a model is related by the extends relation to define its location within a problem domain.

The extends relationship is critical in locating a



**Figure 5.** Contextual dependencies of a simulation model

component within such a specialization/generalization hierarchy. The location of a model within taxonomy of model specifications help designers interpret model's role within the confines of the problem domain. Being able to locate a component in a schema in terms of its dependencies increases our understanding about its unique features, to which we can assign responsibilities. More specifically, one can better relate a component to objectives of the experimental frame by performing substitutability analysis for DEVS model specifications [34]. The set of all specification components reachable via the extends relationship from the conceptual model Y designate all plausible model specifications within the problem domain that safely can be substituted by a simulation model (i.e., DEVS atomic model) that implements Y (i.e., DEVS atomic model specification).

#### 4.2.2 Experimental Context

The experimental context entails the immediate concrete experimental frame components (i.e., DEVS acceptor, transducer, and generator components) [22] as well as their design dependencies in terms of the extends and needs relationships. There are at least two uses of an experimental context during reuse and model adaptation for composition: (1) A simulation model does not operate in isolation; it needs to be coupled with a driver that supplies the necessary inputs and events to facilitate a simulator to generate the behavior of the model. The experimental context defines all necessary components to experiment with the simulation model. As such, these model components may need to be transferred with the original simulation model into its new context. (2) Given a new environment for which the reusable model is being qualified, the original experimental context can be used to recognize similarities between the objectives of the simulation model and the new simulation study.

Demonstration of the mapping and satisfiability of the extends relationship among the specification components (i.e., acceptor, generator, transducer) of the original experimental context and the experimental constraints of the new simulation study enables reasoning about similarity between the contexts. More specifically, once the specification components of the models that drive the reused simulation model in the new environment are shown to be safely substitutable in place of the specification components the original experiment context, one can qualify a model for reuse. Otherwise, the new immediate environment of the simulation model needs to be customized until it is shown to be substitutable for the original experimental context.

#### 4.2.3 Realization Context

The realization context of the simulation model X entails all concrete components reachable from X via extends and needs behavioral dependencies. These components involve the context needed to complete the definition of a simulation model in its new environment. More specifically, the realization context entails the concrete components on which X depends to provide its specified behavior (i.e., model Y). Referring back to Figure 5, if X implements concept Y, which extends concept A, then there is an implicit dependency between X and P. That is, one may need to understand P to comprehend X. Furthermore, X needs to use (i.e., import) the services specified by the conceptual model or specification component U. Since U has two different implementations, Q and R, any of these models is required to complete the definition of X in the new environment; hence, they need to be included in the realization context. The specification component of X may indirectly need to use other concrete components such as C when instantiated in a new experiment. This dependency arises due to the potential behavioral relations defined over the components that the specification of X depends. For instance, model Y may depend on a conceptual model B to complete its specification due to a needs relation. Therefore, to realize the behavior specified in B, X needs to use a concrete model component C that implements conceptual model B to satisfy the behavioral specification presented by Y.

### 5. A Simulation Model Design Strategy for the Storage and Retrieval of Context Using Introspective Mechanisms

Abstract behavioral relations that depict the contextual assumptions of simulation models hold out the potential to improve ability to understand and reason about the fitness of a simulation model in a new context. Achieving these benefits, however, involves effective

communication and distribution of such dependencies. This requires delivering contextual assumptions along with the simulation model itself. The significance of this issue is apparent: a client of a simulation model can not harvest the benefits of the specification of the concept and contextual assumptions, unless the specification is delivered along with the simulation model. While conventional methods used in distributing specifications and documentation in printed form, as plain text or HTML, or in a platform specific help file can be utilized, such an approach misses the opportunity to appropriately use such contextual dependency information by leveraging them in development and integration tools.

### *5.1 A Possible Solution: Introspective Simulation Models with Reflective Capabilities*

The disadvantage of separate distribution of contextual constraints of a simulation leads to the consideration of alternative mechanisms. Distributing contextual assumptions and constraints of a simulation model as part of the simulation software in terms of embedded comments may not be practical, as discovery, location, and interpretation of such comments is counterproductive. Furthermore, distribution of simulation software as binary packages is a common practice to facilitate protection of proprietary information. In such cases, accessing contextual dependency specification information is not possible. Yet commercial software component technologies have significantly evolved during the last decade to suggest alternative mechanisms to packaging and distribution. Existing software technologies such as COM, ActiveX, CORBA, and JavaBeans provide facilities to inspect a component's external interface. More specifically, through these inspection mechanisms one can discover the interface, parameters, and services provided by a software component. Yet, such mechanisms do not yet exist in most popular simulation platforms such as HLA [35] and DEVS [22]. Though the same capability can be achieved with novel model design strategies that involve computational reflection.

The ability of a system to respond to inquiries about its structure and ideally its behavior is a well-studied concept and is the cornerstone of computational reflection [36,37,38]. Computational reflection is defined as an activity of a system to query its structure and behavior to guide its own computation by accessing and potentially updating its own state. While dynamic run-time state update facilities that allow modifying and altering behavior are beneficial for simulation modeling of complex uncertain phenomena [32], the capabilities of interest are introspective mechanisms that are limited but yet still powerful tools to query contextualized simulation models. A contextualized

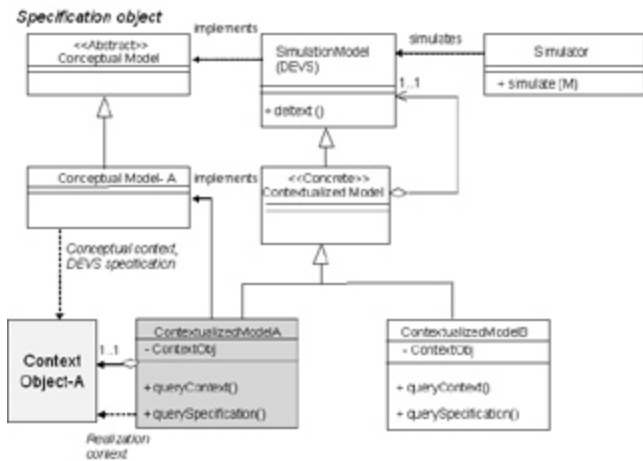
simulation model enables the packaging of and access to contextual dependency information associated with the component that embodies the software that generates the requisite model behavior. Many OO languages provide facilities for accessing interface information (i.e., Java), yet the packaging of concept specification and contextual information requires embedding coarse-grain specification objects accessible through the interface of simulation models.

### *5.2 A Design Strategy for Embedding Contextual Assumptions with Simulation Models*

We define the notion of context objects independent of the simulation model. This viewpoint is consistent with the conceptualization shown in Figure 4. Each context object stores the conceptual, realization, and experimental dependencies of the simulation model they represent. Yet, the interface of a model should reflect the introspective capabilities and services provided by the model. Seamless extension of existing models with introspective capabilities along with association with context objects constitutes the basis of the proposed design strategy. The UML [39] notation for the design configuration shown in Figure 6, lays out a novel and practical approach for wrapping a simulation model to incorporate the contextual dependency specification. The objective in developing the design strategy is to add context and concept specification objects to simulation models dynamically and transparently without effecting the simulator, which is a client of the simulation model.

The two base classes in the configuration are the abstract conceptual and simulation model components that define the interface along with possible generic operations. A simulation model is parameterized by the conceptual model to execute the protocol specified within the conceptual model by using a simulation engine (i.e., DEVS simulator). The parallel class hierarchy that constitutes the conceptual and simulation model types, including contextualized simulation models, is used in deferring instantiation of the actual conceptual model (i.e., type A) to the concrete simulation model that implements that specific concept. By inheriting and aggregating the interface of the simulation model, the contextualized model is used to attach additional responsibility to a simulation model dynamically. As such, the contextualized model provides the same interface and services required by the original simulation model. Yet, through the aggregation relation, it includes an object of type simulation model to delegate behavior generation responsibilities to the actual simulation model. As such, the requests and services required by a simulation model can be intercepted for evaluation by the contextualized model, as well. However, in addition to conventional simulation services, the contextualized

model can provide new features that enable querying the context and concept specification objects associated with concrete contextualized simulation models.



**Figure 6.** Design configuration for contextualized simulation models

For instance, the `ContextualizedModelA`, in Figure 6, includes objects of type context that can be accessed and manipulated by using the method `queryContext`. At the very primitive level, a `GetContextObject()` could be used to retrieve such objects. The conceptual model specification and the import/export requirements of the `ContextualizedModelA` are used to capture the conceptual, realization, and experimental contexts. By interposing a separate processing layer between the client (i.e., simulator) and the simulation model, it becomes possible to add or remove features that exploit the concept specification (i.e., in terms of DEVs formalism) and contextual objects embedded with the simulation models.

While the notion of wrappers and the use of reflection are not new ideas, the novelty is their use in conjunction with each other to provide access to conceptual specifications and contextual dependencies defined in terms of behavioral component relations. With the use of the above design strategy, separate features easily can be added to the seamlessly augmented interface of a contextualized simulation model to query the contents of the context object itself. Other potential uses of such wrappers in component-based composition simulation model development are discussed in the conclusion.

### 5.3 Limitations of the Strategy

While the proposed strategy enables seamless access to context objects that facilitate awareness about the objective context of the model, there are a number of drawbacks. An immediate issue is the augmentation of the simulation models with context objects and associated services that may lead to code inflation.

Yet, while the additional code can cause inefficiency if used at run-time, the expected usage of the additional features is offline (i.e., design time). That is, simulation practitioners are expected to use new services to query the context objects to make decisions about the necessary import/export requirements of the simulation models. The evaluation of the contextual dependencies takes place during the design time of the new simulation study. This strategy naturally requires development of drivers to deploy reused simulation model components and retrieve their context objects. Development of such drivers needs to be cost effective. Another concern entails the representation and communication of context objects. First and foremost, the representation must be interpretable and be manipulated through program level accesses to the contextualized simulation models. The use of semi-structured data representation mechanisms such as XML is a plausible strategy. Committing to a specific contextual object format and having a generally acceptable standard structure that defines the conceptual, realization, and experimental dependencies of a simulation model are challenging tasks.

## 6. Conclusions

In [2, p.13], the authors argue a fundamental need for building into agreed methods of model representation the requirement that the conceptual model, simulation model, and the context of reuse (experimental frame) be distinguished and specified separately. Agreeing with their finding, we suggest means to capture and formalize the conceptual behavioral dependencies among these fundamental M&S framework components. Furthermore, it is argued that the conceptual and realization context of models and their simulations is as important as the experimental frame to qualify and eventually deploy a new model within a simulation study. Composability is defined as the capability to select and assemble components in various combinations to satisfy specific user requirements meaningfully. A defining characteristic of composability is the ability to combine and recombine components into different systems for different purposes. Ability to locate and reuse existing models in new experiments and application scenarios is critical as a first step to compositional construction of complex simulations. However, within the simulation modeling community there is not yet an agreed conceptual model of what constitutes a reusable simulation model. The approach presented in this paper is a step toward conceptualizing reusable models in terms of explicit, unambiguous, and precise characterization of contextual dependencies. The significance and role of separating concept, content, context, and simulator is discussed to provide a basis and rationale for contextualized simulation models.

We argue that the notion of experimental frames is necessary for effective reuse, but insufficient to make a decision for the qualification of a model for a new experiment. While the value of an experimental frame for reuse is well documented, a significant aspect central to reuse is the characterization of the context of the model itself. We have shown how the use of formal conceptual specifications could facilitate qualification of a model for reuse. In particular, the extends relation acting as a behavioral substitutability condition could provide a useful avenue for a sound and formal reuse assessment tool.

Within the current M&S development practice, the suggested approach has certain practical limitations. Most conventional simulations are missing formal specification of their behavior. Unless we develop means to package and distribute specifications along with concrete simulation models, the value of behavioral specifications will be confined within the limits of the original construction process. Yet, such specifications of concepts, as argued in this paper, can be extremely useful for reasoning about fitness of a component in a new experimental frame. Furthermore, having a formal ontological representation of the context in terms of behavioral dependencies would enable an M&S practitioner to determine how and under what circumstances a model can be reused in conjunction with other components. Next generation simulation development environments could facilitate leveraging ideas presented in this paper by encapsulating, packaging, and distribution of formal I/O system specifications (i.e., DEVS) along with the simulation code. Furthermore, each model can be packaged with information of its contextual dependencies. The ability of a simulation component to respond to queries about its own behavior, structure, and dependencies would be an immediate emerging application to make use of such a facility. Further avenues may emerge from such services. For instance, developing built-in self verification mechanisms that use such a facility to access encapsulated specification through a reflective API could as well prove useful during the reuse process. Besides, packaging test cases with the model components or deriving test cases/oracles from the model specification embedded within the simulation component, and then applying them for verification could facilitate instilling confidence in the reused component. Such a strategy can enable M&S enabled testing of distributed systems as well [23].

## 7. References

- [1] Basili, V., Briand, L., and W. Melo. (1996). "How Reuse Influences Productivity in Object-Oriented Systems," *Communications of the ACM*, vol. 39, no. 10 pp. 104-116.
- [2] Davis K. P. and Anderson A. R. (2003). Improving the Composability of Department of Defense Models and Simulations, RAND Technical report available at <http://www.rand.org/publications/MG/MG101/> (last accessed 2004).
- [3] Defense Modeling and Simulation Office (2004). "Composable Mission Space Environments," available at <https://www.dmsomil/public/warfighter/cmse> (last accessed 2004).
- [4] Gustavson P., L. Root, P. Zimmerman, and C. Turrell, (2003). "Conceptual to Composable: Driving Towards Rapid Development of Simulation Spaces," Interservice/Industry Training, Simulation, and Education Conference (IITSEC) 2003, available at <http://www.simventions.com/whitepapers/1427.pdf> (last accessed 2004).
- [5] Tolk A. (2004). "Composable Mission Spaces and M&S Repositories - Applicability of Open Standards," Proc. Of Spring 2004 Simulation Interoperability Workshop (SIW), available at <http://www.sisostds.org>.
- [6] AHBM (2004). The Affordable Human Behavior Modeling Program in Human Systems Research, Office of Naval Research, synopsis available at [http://www.onr.navy.mil/sci\\_tech/personnel/342/training\\_afford.asp](http://www.onr.navy.mil/sci_tech/personnel/342/training_afford.asp). (last accessed 2004).
- [7] Overstreet C. M., R. E. Nance, and O. Balci. 2002. Issues in Enhancing Model Reuse. *International Conference on Grand Challenges for Modeling and Simulation*, Jan. 27-31, San Antonio, Texas, USA.
- [8] Ören T.I. and Zeigler B.P. (1979). "Concepts for Advanced Simulation Methodologies," *Simulation*, vol. 32, no. 3, pp. 69-82.
- [9] Hofmann M. A. Criteria for Decomposing Systems into Components in Modelling and Simulation – Lessons Learned with Military Simulations, dissertation, Armed Forces University, Munich, Germany 2003.
- [10] Page, E., and J. Opper. (1999). "Observation on the Complexity of Composable Simulation." In *Proceedings of the 1999 Winter Simulation Conference*, 553-560.
- [11] Ceranowicz A., M. Torpey, B. Helfinstine, J. Evans, J. Hines (2002). "Reflections on Building the Joint Experimental Federation," In *Proceedings of the IITSEC 2002 Conference*.
- [12] Nance E. R. (1999). "Distributed Simulation with Federations: Expectations, Realizations, and Limitations," In *proceedings of the 1999 Winter Simulation Conference*, pp. 1026-1031.
- [13] Carr F.H and L. Myers (2003). "Interoperability and Reuse Through a Modeling and Simulation Common Operating Environment," In *Proceedings of the Simulation Interoperability Workshop 035-SIW-016*, Spring 2003.
- [14] Falkenhainer B and K. Forbus. (1991). "Compositional Modeling: Finding the Right Model for the Right Job," *Artificial Intelligence*, Vol. 51:pp. 95-143, 1991.
- [15] Forbus K. (1984). "Qualitative Process Theory. *Artificial Intelligence*," vol. 24:pp. 178-219, 1984.
- [16] Levy Y.A., Y. Iwasaki, and R. Fikes. (1997). "Automated Model Selection for Simulation Based on Relevance Reasoning," *Artificial Intelligence*, Vol. 96:pp. 351-394, 1997.
- [17] Broy M. (1997). "Compositional Refinement of Interactive Systems," *Journal of the ACM*, Vol. 44(6):pp. 850-891, November 1997.
- [18] Jackson D. and M. Jackson. (1996). "Problem Decomposition for Reuse," *Software Engineering Journal*, vol. 11 (1), January 1996.
- [19] Jackson M. (1994). "Problems, Methods and Specialization," *IEEE Software*, vol. 11, no. 6 pp. 57-62, November 1994.
- [20] Zaremski M. A. and J. M. Wing (1995). "Specification Matching of Software Components," In *Proceedings of the 3rd ACM SIGSOFT symposium on Foundations of software engineering*, pp. 6-17.
- [21] Morse L. K., M. D. Petty, P. F. Reynolds, W. F. Waite, and P. M. Zimmerman (2004) "Findings and Recommendations from the 2003 Composable Mission Space Environments Workshop", *Proceedings of the Spring 2004 Simulation Interoperability Workshop*, Arlington VA, April 18-23 2004, pp. 313-323.
- [22] Zeigler B. P., H. Praehofer, and T. G. Kim. (2000). *Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems*. 2nd Ed. Academic Press. Davis.

- [23] Zeigler B. P., D. Fulton, J. Nutaro, P. Hammonds. (2003). "M&S Enabled Testing of Distributed Systems: Beyond Interoperability to Combat Effectiveness Assessment," 9th Annual Modeling and Simulation Workshop, Dec. 8-11, 2003, ITEA White Sands Chapter.
- [24] Szyperski C., D. Gruntz, and S. Murer (2002). *Component Software - Beyond Object-Oriented Programming - Second Edition*, Addison-Wesley / ACM Press.
- [25] Zeigler B. P. and Sarjoughian H. (2003). *Introduction to DEVS Modeling and Simulation with Java: Introduction to Component-based Simulation Models*. ACIMS Technical Report. [http://www.acims.arizona.edu/SOFTWARE/devsjava\\_licensed/DevsJavaUserGuideV1.1.zip](http://www.acims.arizona.edu/SOFTWARE/devsjava_licensed/DevsJavaUserGuideV1.1.zip) (last accessed 2004).
- [26] Chandrasekaran B. and T. Johnson (1993). "Generic Tasks and Task Structures," *Second Generation Expert Systems*. Springer, Berlin, pp. 232-272.
- [27] McCarthy J. (1991). *Notes in Formalizing Context*. Technical report. Computer Science. Stanford University.
- [28] Porzel R. and I. Gurevych (2003). "Contextual Coherence in Natural Language Processing," *Modeling and using Context: In Proceedings of the CONTEXT 2003*. pp. 272-285.
- [29] Turner M. R., E. H. Turner, T. A. Wagner, T. J. Wheeler, and N. E. Ogle (2001). "Using Explicit, A Priori Contextual Knowledge in an Intelligent Web Search Agent," *In Proceedings of the CONTEXT 2001*. pp. 343-352.
- [30] Jang S. and W. Woo (2003). "Ubi-UCAM: A Unified Context-Aware Application Model," *Modeling and Using Context: In Proceedings of CONTEXT 2003*, pp. 178-189.
- [31] Jeh G. and J. Widom. (2001). *Scaling Personalized Web Search*, Technical report, Stanford University Database Group, 2001
- [32] Yilmaz L. and T. Oren (2004). "Improving Simulation Reuse within the Model-Simulator-Context Framework," *In Proceedings of the Conceptual Modeling and Simulation Conference at the First International Mediterranean Modeling Multiconference*.
- [33] Yilmaz L. and Stephen H. Edwards. "Specifying and verifying collaborative behavior in component-based systems," *In Proceedings of the RESOLVE Workshop 2002*. Technical Report TR-02-11, Dept. of Computer Science, Virginia Tech, Blacksburg, VA, pp. 95-104.
- [34] Yilmaz L. (2004) "Verifying Collaborative Behavior in Component-Based DEVS Models," *Simulation: Transactions of the Society for Modeling and Simulation International*. Special Issue: Component-Based Modeling and Simulation, Vol. 80, No. 7-8, pp. 399-415.
- [35] Dahmann, J., Fujimoto, R., and Weatherly, R. (1998). *The DoD High Level Architecture: An Update*. *In Proceedings of the 1998 Winter Simulation Conference*, pages 797-804, Washington,
- [36] Kiczales G., J. Des Rivieres, D G. Bobrow (1992). *The Art of the Metaobject Protocol*. MIT Press.
- [37] Saeki M., T. Hiroi, T. Ugai. (1993). "Reflective Specification: Applying a Reflective Language to Formal Specification," *In Proceedings of the 7th International Workshop on Software Specification and Design*, IEEE CS Press, pp. 204-213.
- [38] Smith B. (1982). "Reflection and Semantics in a Procedural Language," *Technical Report*, MIT, Laboratory for Computer Science, Cambridge, MA.
- [39] Booch G., J. Rumbaugh, I. Jacobson (1999). *The Unified Modeling Language User Guide*. Addison-Wesley.