Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2014, Article ID 492737, 18 pages
http://dx.doi.org/10.1155/2014/492737

Review Article

Software Frameworks for Model Composition

Mikel D. Petty,[1] Jungyoon Kim,[2] Salvador E. Barbosa,[1] and Jai-Jeong Pyun[3]

[1] *University of Alabama in Huntsville, Huntsville, USA*
[2] *REALTIMEVISUAL Inc., Seoul, Republic of Korea*
[3] *Agency for Defense Development, Daejeon, Republic of Korea*

Correspondence should be addressed to Mikel D. Petty; pettym@uah.edu

Received 10 July 2013; Accepted 17 December 2013; Published 18 February 2014

Academic Editor: Abdelali El Aroudi

A software framework is an architecture or infrastructure intended to enable the integration and interoperation of software components. Specialized types of software frameworks are those specifically intended to support the composition of models or other components within a simulation system. Such frameworks are intended to simplify the process of assembling a complex model or simulation system from simpler component models as well as to promote the reuse of the component models. Several different types of software frameworks for model composition have been designed and implemented; those types include common library, product line architecture, interoperability protocol, object model, formal, and integrative environment. The various framework types have different components, processes for composing models, and intended applications. In this survey the fundamental terms and concepts of software frameworks for model composition are presented, the different types of such frameworks are explained and compared, and important examples of each type are described.

1. Introduction

A software framework is an architecture or infrastructure intended to support and enable the integration and interoperation of software components. It may consist of concepts, technologies, tools, protocols, standards, control mechanisms, interfaces, and processes intended to enable the rapid, efficient, and flexible assembly of systems from components in a practical setting. Here we focus our attention on a subclass of software frameworks, specifically those software frameworks where the components are either implementations of models, for example, a model of terrain effect on wheeled vehicle movement, or nonmodel support components, for example, a map display for a user interface, and the framework is intended to support the composition of those models and support components into more complex models and simulation systems. For brevity, such software frameworks specifically designed to support the composition and integration of models and simulation support components will be referred to as *simulation frameworks* or when the meaning is clear simply as a *framework*.

This paper's scope is simulation frameworks for model composition. Its purpose is to review such frameworks, including both their theoretical characteristics and capabilities and their implementation and use in practice. To that end, this paper is both a tutorial and a literature survey. As a tutorial, it explains basic context and defines important terminology and provides a categorization scheme for simulation framework types. As a literature survey, it cites and summarizes publications in the research and professional literature. In an important part of the paper's approach to the subject, several existing software frameworks that have been developed to support model composition are described in some detail.

This paper is structured as follows. Section 2 introduces background material, including context and key definitions. Section 3 defines categories or types of simulation frameworks. Section 4 describes important examples of the different simulation framework types. Section 5 provides a summary of paper's findings and recommendations.

2. Background

This section introduces essential background information for understanding the software frameworks to be surveyed. It

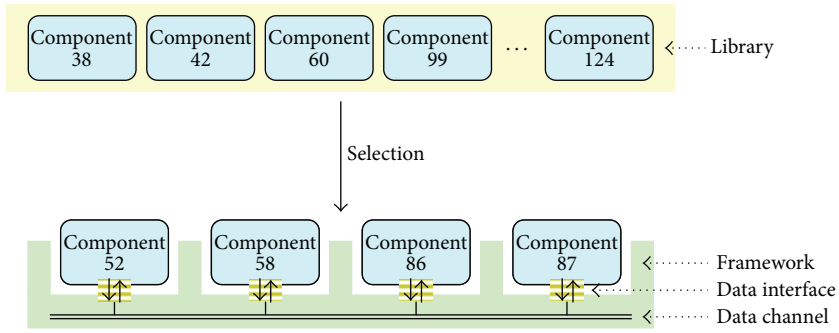


Figure 1: Framework, components, and library.

presents a series of important definitions of terms relating to simulation frameworks and components and uses those definitions to place the idea of a software framework for model composition in the context of related ideas and technologies. The nature of software components within such a framework is also discussed.

2.1. Definitions and Context. To begin, several important terms relating to simulation frameworks and components are defined and explained. The definitions are not given in alphabetical order, as is often conventional for lexicons, but in an order intended to enable the reader to comprehend the definitions as each builds and depends upon the preceding definitions.

Software Reuse. Software reuse refers to using a previously developed unit, package, or module of software more than once, either for the purpose for which it was originally developed or for a new purpose or in a new context. Software reuse may save time, effort, or cost for development or testing. If the software being reused is a model, reuse may add credibility to the new application if the software underwent verification, validation, and accreditation for its previous use [1, 2].

Component. The concept of components is fundamental in the context of general software reuse. Multiple definitions of component are available from the software reuse literature; selected examples include the following. (1) A unit of executable or source code that is available for reuse [3]. (2) A reusable software package or module that encapsulates a set of related functionality and communicates with other components via an interface [4]. (3) An encapsulated unit of software with a known set of inputs and expected output behavior where the implementation details may be hidden or unknown; an interchangeable element of a system that conforms to a specification [5]. Our focus in this paper is on software frameworks designed specifically for simulation software, that is, simulation frameworks. In simulation frameworks, a component is a software component and has all the properties of one, but it may also have additional simulation-specific properties. A component may be a model capable of simulating all or part of some real-world system of interest, such as a physics-based model of aircraft flight dynamics, or it may have functionality specific to a simulation implementation, such as a future event list in a discrete event simulation. Hereinafter, the term *component* will refer to components in general, whereas *model component* will refer to components that implement all or part of a model and *simulation component* to components that are not a model but implement some simulation-specific support functionality.

Framework. A software framework is an architecture or infrastructure intended to support and enable the integration and interoperation of software components. The essential idea is that components developed to be compliant and consistent with a framework may be combined, connected, and composed within that framework, and that such compositions may be assembled more readily and with more likelihood of correct operation than would be possible without the framework. Definitions of software frameworks in the literature range from quite generic, for example, "a set of interacting objects that, together, realize a set of functions" [3] to quite specific, for example, "a subsystem that contains abstract and concrete types and classes designed for reuse" [6]. In most definitions the key defining concepts are components as the units of integration, a mechanism to support their integration and interoperation and the fact that different components may be combined at different times. Some definitions of framework emphasize the components, whereas others emphasize the mechanisms to connect the components; both are crucial in an effective framework. Figure 1 abstractly illustrates the idea of a framework; it shows a set of notional components, selected from a larger set of available components stored in a component library that are connected and interoperating via the structure and capabilities provided by the framework.

Composability and Composition. Composability is the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements [7]. The defining characteristic of composability is that different simulation systems can be composed in a variety of ways, each suited to some distinct purpose, and the different possible compositions will be usefully valid. Early theoretical study of composability addressed the question of when a model can be composed

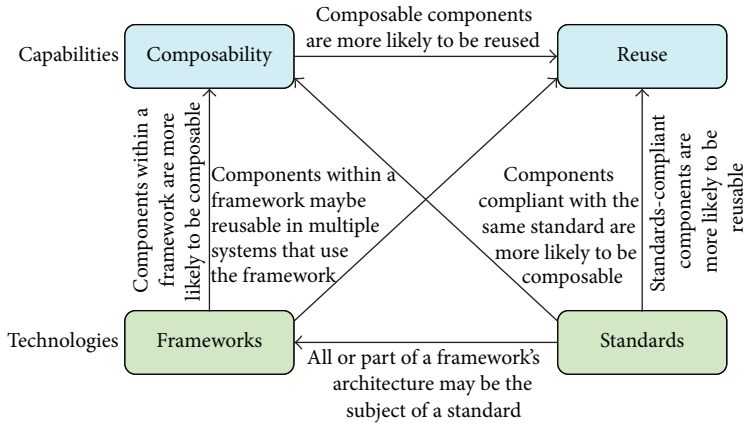


Figure 2: Survey context.

with other models, and if it is so composed, whether the resulting composition is valid [8, 9]. Similarly, the structural and numerical inconsistencies that may arise when composing components that implement physics-based models based on differential equations have been examined [10]. Of more direct relevance to simulation frameworks, efforts aimed at the practical implementation of composability have been identified [11] (Portions of this paper use concepts and terms introduced in [11]).

Given this definition of composability, composition (as a verb) is the process or capability of selecting and assembling components for execution. Exactly how the components are selected and assembled will depend on the framework within which they are composed. The details of selecting components depend more on the framework's library, and the details of assembling them depend more on the framework's implementation mechanisms. Both will be discussed in this survey. Composition (as a noun) refers to a set of components that have been composed to produce an integrated or interoperable whole (in some situations, phrases such as "set of components" or "composition of components" will be used instead of simply "composition" as a noun. The terms are meant be synonymous).

Standards. A standard is an agreed-upon convention or requirement that defines or prescribes some aspect or aspects of a technical system. Such standards are usually embodied as a formal and configuration-controlled document that specifies consistent engineering or technical criteria, methods, processes or practices. Standards may be considered *de jure*, which have the formal definition just described, *de facto*, which exist when a particular product, format, or representation becomes so ubiquitous and dominant that its nonuse would cause significant problems or proprietary, which are standards-like entities that are owned and controlled by a commercial organization [12]. Simulation standards are those standards that apply to simulation, whether to software, development practices, data definitions, or any other aspect of simulation. Of special interest in this paper are the distributed simulation interoperability protocols, including distributed interactive simulation (DIS) [13], high level architecture (HLA) [14], and test and training enabling architecture (TENA) [15]), which can be understood as both standards and frameworks.

Resolution. When describing a model, resolution is defined as "the degree of detail which with the real world is simulated" [16]. When describing a component, resolution denotes the size of the component not in bytes or lines of code, but rather in terms of the scope or extent of the functionality provided by the component. A small component has a narrow and limited functionality and is likely to be useful only when combined with other components, for example, a random number generator, whereas a large component may have a broad and extensive functionality and is likely to be capable of independent operation even without other components, for example, semiautomated forces system [17] (Hereinafter, we will use the terms "small" and "large" to describe components with meaning defined here, i.e., scope of the component's functionality, not its size in bytes.). Successful components, that is, those that are reused, may be of any size. The composition of different size components defines different levels of composability [7].

Library. In the context of frameworks, a library is a collection of components that are available for reuse. A library is realized as a system that accepts, stores, and provides access to components that may be reused. Issues of configuration management (controlling and tracking changes to the components) and discovery (enabling potential users of the components to locate and assess components for reuse) are important in software libraries, but are outside the scope of this survey. A library may store other artifacts in addition to components, such as metadata, data, or documentation. A range of closely related terms are used, for example, *catalog* (which specifically stores discovery metadata), *registry* (which specifically stores metadata schemas), *repository* (which may contain components developed independently and with no prior intent for interoperation), and *storehouse* (which is generic for storage systems) [1], but the distinctions between these terms will not be needed in this survey.

Equipped with these definitions, the subject of this survey can be placed in context, as illustrated in Figure 2. In the figure "composability" and "reuse" are desired capabilities,

and "frameworks" and "standards" are technologies that contribute to the achievement of those capabilities. The mechanisms through which technologies and goals support and enable each other are indicated by the labeled arrows; the directionality of the arrows indicates that the "from" technology or capabilities supports or enables the "to" technology or capability.

2.2. Characteristics of Components. In simulation frameworks, model components are typically larger than a single model of a domain-specific object but are still small enough to be confined to a single area of subject matter expertise. Of course, not all components are model components; some may provide supporting infrastructure to simulation systems, such as a network interface component or an event queue component. Determining the size or functionality of a component is not amenable to rules applicable in all situations, but the "satisfies user requirements" clause within the composability definition provides a guideline; a component should have capabilities that are useful to potential users as a unit, neither too small nor too large. The criteria for making that determination will necessarily depend on the application.

The initial development of a composable component is often more difficult than a noncomposable one, but subsequent development is made easier by the use of previously developed components. Contributing to the additional initial effort is the need to document the assumptions and validity limits of the models more carefully in a composable component. Modeling practices already considered positive in general, such as using parameters instead of constants, documenting well, checking input for validity limits, and striving for clarity in implementation, are beneficial in particular to the development of composable components that embody those models. Not as obvious is that the needs of composability support good modeling practices, for example, the requirement imposed by composability to document a model's assumptions and limits of validity would help the modeler to consider his/her models more carefully at the outset.

Components can be integrated and used with other components only through well-defined interfaces. A component interface should define "... a set of properties, methods, and events through which external entities can connect to and communicate with the component" [18]. In some cases components would have customizable aspects that could be modified at run-time through the interface.

Structured descriptions or specifications of the components (sometimes called metadata or metamodels) can be used to guide the processes of selecting components for a specific purpose and determining if a set of components can be composed [5]. A component's specification (metadata) should include details of the component's interface(s) and model(s).

In practice, it is often assumed that any model component that had been placed in a repository is valid, at least within bounds stated in its component specification. A component's validity constraints are the limits or bounds within with the component's model is deemed valid and may be defined at a low level in terms of physical parameter values, time step sizes, and so on or at a high level in terms of valid applications for the component. Even so, it has been shown that the validity of individual components does not imply that a composition of them can be assumed to be valid [8]. Validating a composition can use traditional validation methods, such as comparing composition output data to baseline data [19] and can also exploit the composition's component structure, such as automatically comparing the domains of validity for each component with the data they are receiving from other components in the composition.

3. Framework Types

Several distinct types of software frameworks for model composition have been developed. The mechanisms used to implement a framework may vary; they include rules, protocols, standards, programming language structures (e.g., class and type hierarchies), interface definitions and implementations (e.g., application programming interfaces), data translators and converters, and data transport utilities. These different mechanisms may be used together in a wide range of combinations to produce a framework. Moreover, different framework types may also vary in the forms and resolutions of the components the framework is designed to connect. Some frameworks are designed for small components, whereas others are designed for large components. (Here "small" and "large" are used as defined earlier with respect to resolution.) As a result of this variability in mechanism and design intent, a number of different types of frameworks have been developed and used within the community of modeling and simulation practitioners. Each offers a mechanism for linking components and transporting data and control between them, and each is intended and designed to support components at a certain range of levels or resolutions but they differ in regarding how to do so.

This section defines six categories or types of software framework for model composition and briefly identifies examples of each type. (An earlier version of the categorization scheme for framework types used here was introduced in [11], where the framework types were described as "approaches to composability engineering".) Extended descriptions of important or interesting example frameworks are given later.

3.1. Common Library. A common library simulation framework is based on a collection or set of software modules or components that are composable or reusable through conformance to a standard interface or set of interfaces that allows the modules to interoperate with the other modules in the library or a subset of them. Typically the components in a common library framework are not models that can be executed in stand-alone mode, although that is not a firm characteristic. The framework may also include tools, services, and standards. Common libraries are developed using a common set of assumptions underlying the models within them and a common protocol for data transfer between them. The components are intentionally designed from the outset to work together with each other in various

combinations. The library may include components with varying degrees of composability, that is, some of the modules may be composable with all or most of the components in the library, whereas other components may work with only a small subset of the other components.

Implemented examples of common library simulation frameworks include the Joint Modeling and Simulation System (JMASS) [20–23], the Common Simulation Framework (CSF) [24], the Pervasive System for Indoor-GIS (PSI) [25], and *J-Sim* [26]. It is often the case that a common library simulation framework is intended from the outset or evolves to become focused on a particular category of simulation applications. JMASS is focused on modeling aircraft avionics and electronics systems, CSF on modeling tactical missile systems, PSI on ubiquitous computing applications, and *J-Sim* on computer and communications networks. CSF and CSFA will be described in more detail later.

3.2. Product Line Architecture. A product line architecture simulation framework is based on the carefully planned development of multiple related simulation products that, to the extent possible, share common software components. The architecture is realized using two software development processes. The first is developing a set of components that will be integrated in various compositions into the products. The second is integrating subsets of those components into specific products. The product line architecture includes both the library of components and an automated (or semiautomated) process or tool for integrating them into products [27]. The range of possible products is known in advance and documented in a detailed specification. The components are designed from the outset to work together in those specific products. As with a common library framework, typically none of the components is a stand-alone simulation product.

Data and control interfaces between the components within an anticipated product are documented in advance in the specification for each component. The architecture may provide a data transfer protocol used within the products.

Examples of simulation products implemented within a product line architecture simulation framework include a flight instrumentation trainer [27], a synthetic environment model [28], a physics-based model of weapons effects on buildings and structures [29], two families of live training instrumentation systems [30, 31], and one semiautomated Forces, a real-time constructive combat model [32]. OneSAF will be described in more detail later.

3.3. Interoperability Protocol. An interoperability protocol simulation framework is based on the run-time exchange of simulation data or services, typically using a distributed simulation interoperability protocol such as ALSP, DIS, HLA, or TENA [33]. In this framework type, simulation systems consist of models or support utilities, each of which is an independently executing process or program distributed across multiple computation platforms that exchange data during the execution of a simulation via a network. That data and the transport mechanisms to distribute and deliver it are defined according to the protocol being used. In the terminology of HLA, which is an architecture standard and interoperability protocol for such systems, the simulations are *federates* and the distributed simulation systems are *federations* [14]. The federates can run independently, but normally interact during execution by sending and receiving data via network messages. In simulation framework terms, the federates are the components and the distributed simulation interoperability protocol is the mechanism for connecting them. Compliance with such a protocol does not, however, guarantee semantic composability and achieving it may require considerable additional effort beyond the initial technical integration of a system; for example, see [34].

It is useful to distinguish between special-purpose federations, which are developed and used for a single or limited number of executions (such as exercises, analyses, or experiments) and persistent federations, which are used repeatedly and possibly modified or enhanced over longer periods of time. There are numerous examples of interoperability protocol simulation frameworks, that is, persistent federations developed using a distributed simulation protocol. Examples include the Joint Training Confederation [35, 36], the Combat Trauma Patient Simulation (CTPS) [37–39], Combined Arms Tactical Trainer (CATT) [40], the Close Combat Tactical Trainer (CCTT) [40], the Modeling Architecture for Technology, Research, and Experimentation (MATREX) [41, 42], and the Framework for Incident Management (FIM) [43]. MATREX will be described in more detail later.

3.4. Object Model. An object model simulation framework is based on a standard for component specifications; note that the standard is for specification, not implementation, of the component. Components that comply with the specification standard are intended to be composable with each other and reusable in a variety of applications. Typically the components are not themselves stand-alone simulation systems, but rather are meant to be composed with each other in the context of an overall simulation system. The defining standard supports interaction of the models with tools and services via the standardized interfaces and is designed to work with interoperability protocols such as HLA.

The primary implemented example of an object model simulation framework is the Base Object Model (BOM) standard [44, 45]. The BOM standard will be described in more detail later.

3.5. Formal. A formal simulation framework depends upon a formal mathematical notation to define the components (usually models), compositions of models, and the interfaces between them. The component connection mechanism is provided by an implementation environment that actually provides the connections specified by the formal definitions. The formal framework type is motivated by a desire to mathematically prove that the components can be composed and to derive their combined behavior once composed.

Examples of formal simulation frameworks include Discrete Event System Specification (DEVS) [46] and Model-Based Systems Engineering (MBSE) [47]. The DEVS formalism supports composability through the use “coupled”

(i.e., composite) models that pass data using "ports" (i.e., interfaces). Model-Based Systems Engineering (MBSE) [47] is another formalism with applicability to simulation. DEVS and MBSE are syntactically quite different but semantically nearly identical, and they have very similar mathematical properties and limits [48, 49]. The DEVS literature is large and many theoretical studies and example applications are available; of particular relevance to simulation frameworks that use components are [50, 51]. Arguably, the defining characteristic of the formal approach, the use of a mathematical formalism to specify components, can increase the difficulty of using them for large practical applications.

3.6. Integrative Environment. An integrative environment simulation framework is a software development and execution environment is used to connect components which may have been written with no prior intent to interoperate. The components may be executable models or data files, and the model components may be written in different languages or tools, such as Excel and C++. Specialized software wrappers and scripts that are part of the simulation framework are used to connect the components. The integrative environment provides communications channels between them. The integrative environment executes the components and relays the intercomponent communications along those channels.

Implemented examples of integrative environment simulation frameworks include VisualComposer [52] and *ModelCenter* [53]. VisualComposer is focused on modeling electrical systems, whereas *ModelCenter* is a general purpose framework. *ModelCenter* is described in more detail later.

4. Example Frameworks

This section describes six important or interesting examples of simulation frameworks. A framework's historical significance, the novelty and effectiveness of its technical features, and its utility in illustrating the comparative strengths and weaknesses of each of the framework types were all considered in choosing the examples. For each example framework, the design intent, capabilities, technical features, and selected example applications are explained.

4.1. Common Library Example: Common Simulation Framework. The United States Army Aviation and Missile Research, Development, and Engineering Center (AMRDEC) commissioned the development of the Common Simulation Framework (CSF) in 1999. The framework was conceived as a standardized structure for dynamic simulations [24]. Although originally designed to be domain neutral, that is, able to technically support many domains of dynamic simulations, it rapidly evolved into a toolkit focused on the modeling and simulation of tactical missile systems [24].

4.1.1. Design Intent. CSF was intended to provide a common environment for simulation and to foster model reuse [24]. CSF was envisioned as a framework for the conduct of dynamic simulations to support systems acquisition and testing by providing a standard for the management and organization of object-oriented models that could in turn be composed into customized simulation systems. The framework was intended to provide a homogenous simulation environment that would enhance reuse of component models, enable rapid developments, and reduce the learning curve of engineers transitioning between programs. As an object-oriented simulation environment, the primary goal for CSF was the ability to assemble custom simulation systems by composing existing object-oriented models compliant with its specifications.

4.1.2. Capabilities. The allowable resolution of the component models is variable; the framework only requires that the component interfaces be compliant with CSF's specifications. The level of detail of any model component's decomposition into subcomponents is not constrained by the standard, so long as communications and invocation of services adhered to the framework's established interfaces. However, the framework's design encourages model-submodel structures that follow the structure of the actual hardware components being modeled [24]. The system provides a graphical user interface for assembling composite models. The individual models are treated as plug-ins into the overall simulation system [54] (A "plug-in" is a software component that adds a feature or capability to an existing software application, to which it connects by a predefined interface. Plug-ins rely on the application they connect to and cannot execute independently.).

4.1.3. Technical Features. Figure 3(a) depicts the layered CSF architecture. As is conventional in layered architectures, software functions in one CSF layer that may invoke functions in the layer below it. CSF was designed to support various underlying hardware platforms and operating systems, such as Linux, Windows, and IRIX. It supports both discrete event simulation and differential equations for continuous system simulation [55]. CSF is implemented in C++ using a client-server approach (user interface and models respectively). The framework itself is supported by the operating system, programming languages, and a set of general utilities. Compliant models are then constructed to execute based on the framework. In order to maximize reuse and portability, models are coded using the standard C++ library for server-side frameworks [55]. Once the models are compiled and the framework are linked into a single executable. The models exchange data during execution via method calls.

The framework layer provides several features to support simulation, including model execution scheduling, file input and output, numerical integration of physics models, and simulation dynamics [24]. Both real-time and non-real-time simulation timing is supported.

4.1.4. Example Applications. The Missile Component Library (MCLib) is a typical example of a CSF application. MCLib was developed to promote uniformity and consistency in tactical missile simulations by providing a common library to users of such simulations. As a collection of object classes it provides

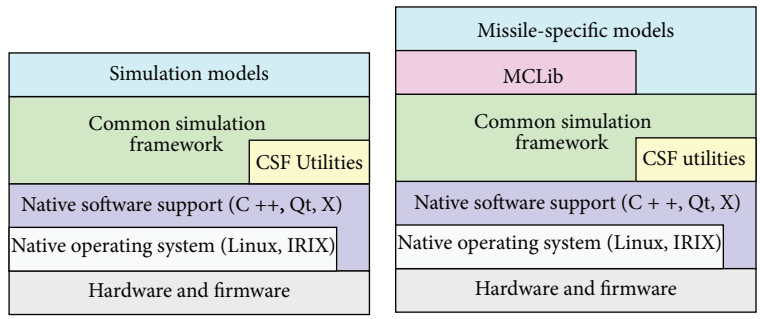


Figure 3: CSF layered architecture (a) and MCLib relationship to CSF and missile models (b); adapted from [56, 57].

general simulation support in two areas: a 6-degrees-of-freedom Propulsion Aerodynamics Controls and Kinematics module (6-PACK) and a Modular Object-Oriented Sensor Evaluation Suite (MOSES). The 6-PACK module provides a set of missile kinematic object classes that model the flight and trajectory of simulated missiles. In object-oriented fashion, these classes may be extended by end users to provide additional functionality or to modify the default behavior of the underlying models. The MOSES module is a collection of CSF-native models that capture the behavior of sensors for target detection, acquisition, and tracking. As with 6-PACK, MOSES may be tailored or extended in an object-oriented fashion. Architecturally, MCLib is a layer above the CSF framework that provides a library of object classes that may be invoked for creating individual missile models, without removing the flexibility of direct access to the framework's features [57]. Figure 3(b) shows the relationship between CSF, MCLib, and custom missile-specific models.

A second CSF application extended the framework to support hardware-in-the-loop (HWIL) testing [54]. Real-time support features within the framework were extended to provide constructs necessary to support HWIL operation. To provide a real-time monitoring capability for external hardware, the framework was enhanced by adding a real-time monitor class. The monitor class runs as a separate thread to avoid performance degradation in the framework. HWIL support was demonstrated by driving a motion simulator rate table with a simulated 6DOF missile trajectory in real-time.

4.1.5. Discussion. CSF's object-oriented design and simple use interface has led to wide use in modeling tactical missile systems [54], for example, the Non-Line-of-Sight Launch System [56].

4.2. Common Library Example: Composable Simulation Framework Architecture. Within the Korean military the use of simulation is increasing in parallel with advances in its defense technologies and weapon systems. In Korea, conventional simulation software development projects have often started from scratch; thus, many simulation systems have been designed with monolithic non-component-based structures. The investments in such systems are considered by some to be inefficient because the resulting systems frequently have redundant or overlapping capabilities across military simulation applications areas such as training, operations research, and combat analysis. In response, the Korean Agency for Defense Development (ADD) initiated research towards composable simulations which are assembled from reusable software components. The Composable Simulation Framework Architecture (CSFA) is a simulation framework pilot project started in 2010 as a feasibility study. CSFA has a hybrid architecture that could be categorized as either a common library or an object model framework.

4.2.1. Design Intent. For component composability, semantic consistency between composed components is a crucial factor [7]. Therefore, an important design goal of CSFA is to facilitate the reuse of composable components by adopting an overarching reference model that is a structured representation of the relevant domain knowledge and guides the composition of components. Additionally, the reference model supports checking the semantic consistency of composed components. Components are developed according to the domain knowledge of the reference model and stored in a repository or common library. Users from various domain areas can share common components which are to be assembled into custom simulation systems for each user's application. CSFA has a layered architecture, which enables users to separate domain-specific model components from domain-generic simulation engines and supporting tools. Domain-specific model components are layered above the simulation engine and tool layers, so as to facilitate reusability of domain models.

4.2.2. Capabilities. The CSFA framework has two main sections, the reference model and the software. The CSFA reference model, which documents combat domain knowledge and defines the system's scope, has two parts, an ontology and a conceptual model. The reference model's ontology contains representations of the combat elements (i.e., entities, units, and systems) corresponding to the likely participants in the battlefield engagements of interest to ADD. The ontology specifies the capabilities of the individual elements. A portion of the ontology is stored in a database, allowing it to be used to check the consistency of components selected for composition with the ontology. The reference model's

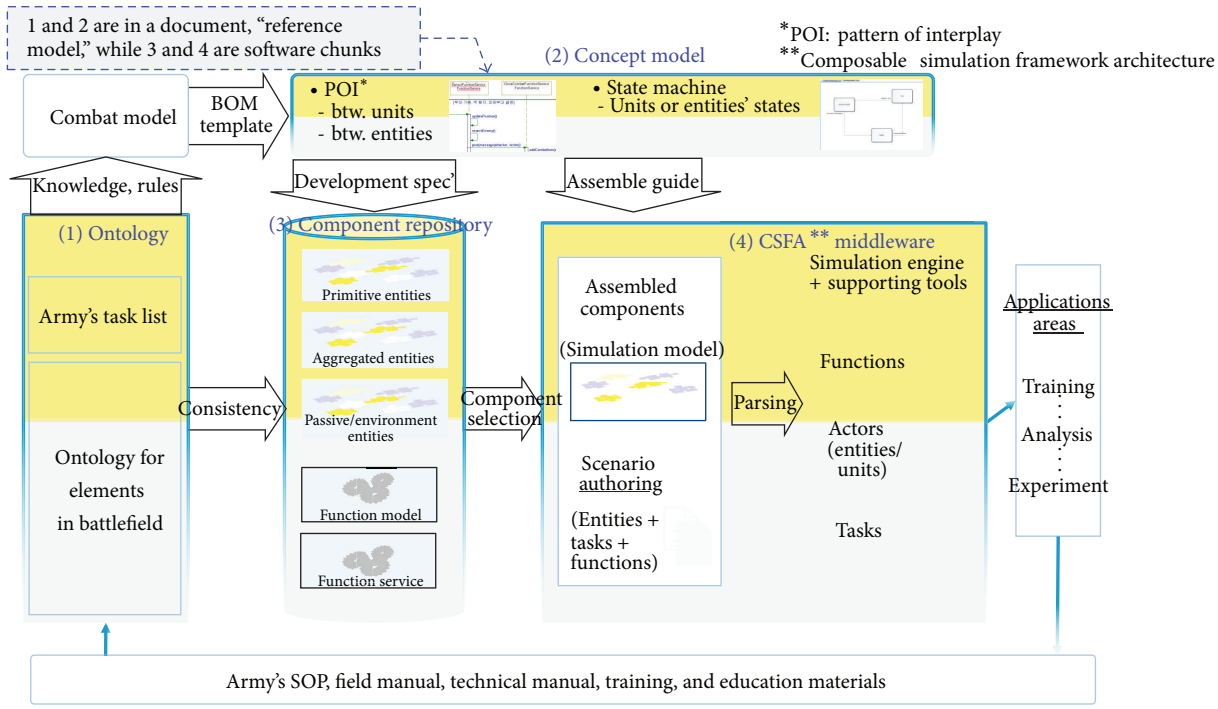


Figure 4: CSFA architecture and context.

conceptual model specifies the possible interactions between the combat elements defined in the ontology. Importantly, as a matter of policy the CSFA reference model can not be easily modified. Proposed changes to the reference model must be reviewed and approved by an administrative organization serving as a configuration control board.

The CSFA software likewise has two parts, a component repository that stores components that have been developed according to the reference model (i.e., the elements in the ontology and the interactions in the conceptual model) and middleware that implements the underlying simulation engine as well as supporting tools for functions such as scenario authoring. Figure 4 shows the four parts of CSFA and their context. Brief explanations of each part were as follows.

(1) Ontology. The ontology expresses military domain knowledge, including battlefield elements, is organized hierarchically. It provides semantic content and is CSFA's most distinctive aspect as compared to other conventional simulation systems. In the ontology, combat elements are categorized as physical or abstract; physical elements represent the battlefield entities such as combatants, automobiles, weapons, rocks, and buildings, whereas abstract elements represent behavioral aspect of elements such as motions, actions, tasks, or missions. The ontology is used to check the consistency between assembled components.

(2) Conceptual Model. The conceptual model specifies the behavioral aspects of the battlefield elements defined in the ontology, especially their interactions, and so reflects the pattern of interplay between battlefield entities. The specification template used in the conceptual model is heavily influenced by the Base Object Model (BOM) SISO standard [44], which will be described in some detail later. Each conceptual model specification includes both a pattern of interplay and a behavior state machine. The pattern of interplay and the state machine are expressed as standard UML representations, a sequence diagram and a state machine diagram, respectively. The conceptual model is also used to check the consistency of selected components, and it provides specifications for component development.

(3) Component Repository. The component repository, which can be seen as a common library, allows simulation users to select and compose components. The components implement aspects of the reference model (the ontology and the conceptual model). Developers produce components according to the specifications in the reference model.

(4) Middleware. The middleware is CSFA's execution software; it includes the run-time simulation engine and supporting tools. The simulation engine parses the scenario file, loads components and assembles a composite model to be executed according to the scenario, and executes the assembled model. It supports distributed simulation interoperability using the high level architecture protocol. The middleware's supporting tools support the various phases of simulation preparation, execution, and analysis; including scenario editing, repository management, component composition, simulation execution control, 2- and 3-dimensional visualization, and after action playback and review.

4.2.3. Technical Features. The CSFA software is written in the C++ language using the Microsoft Visual Studio integrated development environment and is implemented as a plug-in.

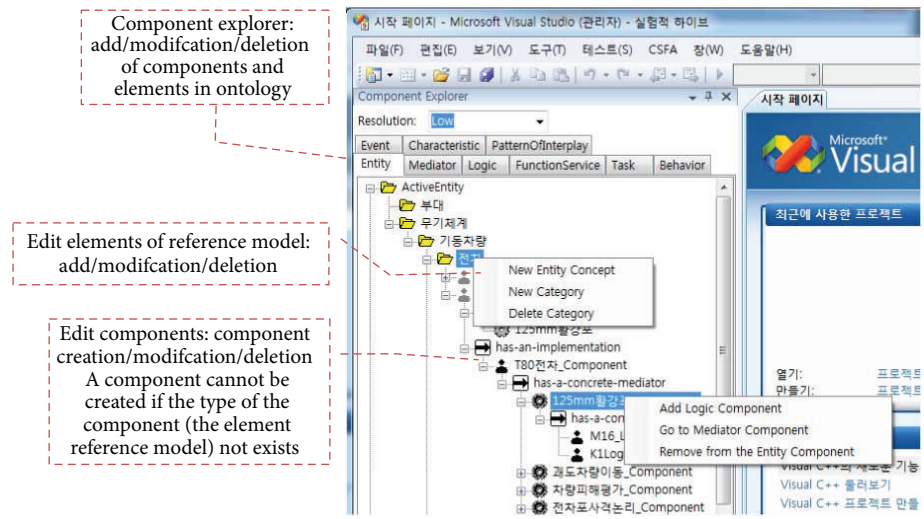
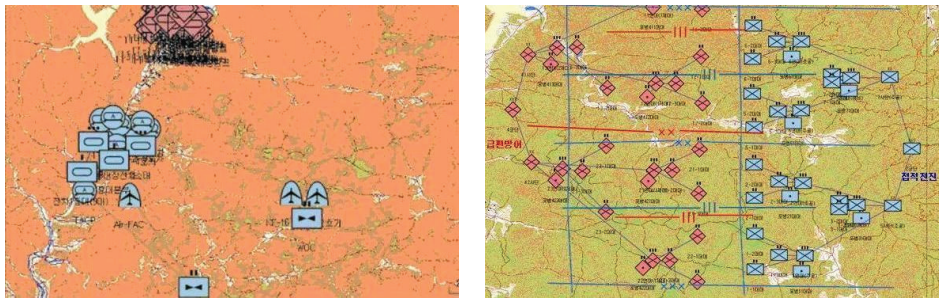


Figure 5: Example screen image from CSFA's Component Explorer.



(a) (b)

Figure 6: CSFA screen images, showing entity-level (a) and unit-level (b) combat.

This environment is consistent with ADD's emphasis on the use of commercial products. Figure 5 illustrates CSFA's Component Explorer, which is part of the component composition supporting tool. In the left panel of the figure, the yellow folders correspond to elements or categories of elements in the ontology and the black icons are implemented components. A component cannot be added to the repository unless there is a corresponding element in the ontology. Because of the policy of limiting ontology modifications mentioned earlier, additions and modifications to the yellow folders are strictly controlled, thereby preventing the development of components that are potentially inconsistent with the reference model.

4.2.4. Example Applications. As this is written, CSFA is an ongoing project; thus, there are no examples of it in practical application. However, two simulation models have been built using CSFA as feasibility demonstrations: direct-fire engagements under different close air support weapons types and a large-scale attack of blue infantry battalions on red mechanized battalions. The former models combat at the entity level and the latter at the unit level, demonstrating CSFA's ability to simulate combat at multiple resolution levels. Figure 6 shows screen images from both the entity-level model (left) and the unit-level model (right). The entity-level model includes algorithms which reflect entity level engagements such as firing weapons' kill probabilities and targets' damage assessment. In the unit-level model, Lanchester attrition functions [58] model combat between military units, considering factors that include number of weapons, type of weapons, and unit vulnerability coefficients.

4.2.5. Discussion. The Korean military has already adopted OneSAF (described later), but it is limited in actual use in Korea because of indigenous Korean tactics, weapons, and environments. CSFA is a intended as a means of overcoming those limitations. To date, CSFA development has achieved only preliminary component reuse goals, reusing only the components that reflect the elements in the reference model ontology.

Further development is planned to extend the scope of reuse to nonmodel components that are parts of software products. An important and advanced feature of CSFA is the close connection between the system's "documentation," that is, the reference model embodied in its ontology and

conceptual model, and its "software," that is, the component repository and middleware. This connection provides semantic context and consistency to components developed for the framework, thereby increasing their composability. The usefulness of conceptual models in facilitating model interoperability and composability has been observed by others as well [59].

4.3. Project Line Architecture Example: One Semiautomated Force. OneSAF is a US Army constructive entity-level combat model, designed to simulate brigade and below combat and noncombat operations at the entity level [32]. Development of OneSAF began in 1996; it has been extended and enhanced continuously since then. As a semiautomated force system, OneSAF generates doctrinally correct tactical behaviors up to company-sized forces [17, 30].

4.3.1. Design Intent. OneSAF is intended to be a general-purpose entity-level model that will reduce duplication in the US Army's modeling and simulation development efforts, provide improved interoperability and reuse, and meet future simulation needs [60]. Expected uses of OneSAF include the development of advanced concepts for doctrine and tactics, training of unit commanders and staffs, development of new weapon systems, support to test and evaluation, and production of data as input to other simulations.

OneSAF is built on a simulation framework known as the OneSAF Product Line Architecture Framework intended to "organize, categorize, and define a layered software structure in order to incrementally meet the OneSAF requirements" [61]. The framework is designed to support various user domains with multiple applications working from a common architecture and set of components [62]. Within the framework, components, tools, and services can be composed into products and system compositions specific to user applications. OneSAF's product line architecture framework has been widely described [62–66].

4.3.2. Capabilities. The OneSAF Product Line Architecture Framework, illustrated in Figure 7, is organized into layers of products and components and employs a hierarchical composition process. Beginning with the component layer in the middle of the figure and moving up, components are defined within the framework. Components may be one of four types: model, tool, infrastructure, and repository [62]. The framework allows independent development of components; they must have complete service and interface definitions and formal documentation. A single component may be used in multiple products and system compositions; conversely, multiple implementations of a single component are possible for situations when a specific product requires a particular variation of a component. One or more components are composed into products, shown in the figure in the product layer. Each product is a distinct unit of simulation or simulation-support functionality. Finally, products are composed into system configurations, shown in the figure in the system compositions layer, which are complete executable systems that provide configured end-user functionality for operational use within mission areas, such as analysis or training. Examples of typical system compositions a single platform standalone executable to be used for force structure analysis or a multiplatform federated executable to be used for distributed staff training [61].

The layers below the component layer figure contain components that provide capabilities that will be used by most or all products or that enable the composition process. The component support layer provides common simulation and composition services that are used by components or used to compose components. The repository component layer is a set of repositories for storing compositions and simulation data. The common services layer provides shared and non-domain-specific services, such as database management, time synchronization, and interoperability protocol services. Finally, the platform layer abstracts the hardware and software environment in which the components, products, and system compositions operate.

4.3.3. Technical Features. The components can be assembled within the framework into twelve specific products that collectively make up the OneSAF product line [30, 62]:

(1) *System composer* composes components into products and products into system compositions.

(2) *Knowledge engineering environment* stores and organizes combat domain knowledge.

(3) *Event planner* plans activities and tasks in preparation for a simulation event or exercise.

(4) *Model composer* composes primitive models, such as physical or behavioral models, into composite models.

(5) *Simulation generator* selects terrain and scenario information for an execution.

(6) *Technical manager* supports execution configuration, performance prediction, and benchmarking.

(7) *Simulation core* serves as the runtime engine for a simulation execution.

(8) *Simulation controller* provides mechanisms, such as maps, overlays, and monitors for run-time control of a simulation execution.

(9) *C4I adapter* connects OneSAF system compositions and actual command, control, communications, computers, and intelligence (C4I) devices.

(10) *Analysis and review* provides recording, playback, and analysis of simulation execution data.

(11) *Repository manager* enables creation and use of local and remote components and data.

(12) *Maintenance environment* provides an integrated software development environment.

4.3.4. Example Applications. OneSAF has been used for an extensive range of analysis, acquisition, and training applications. In addition to everyday applications, some unusual examples include the creation of a tool providing course of

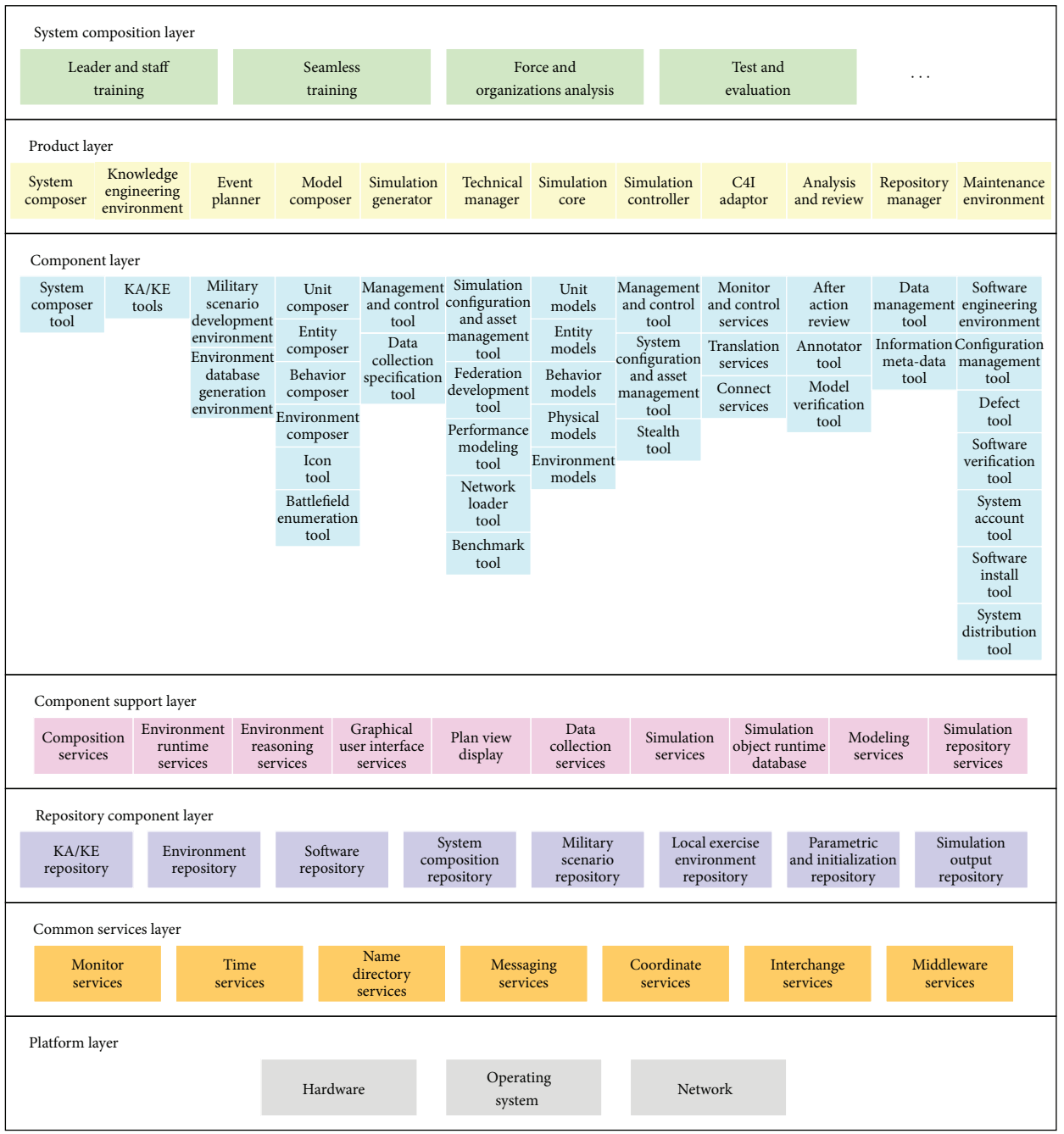


Figure 7: The OneSAF product line architecture framework; adapted from [30].

action analysis and mission rehearsal capabilities [67], testing algorithms for real C4I systems [68], modeling World War II tank combat [69], creation of a "cyber range" for cyber warfare analysis and training [70], and virtual training on operating construction equipment [71].

4.3.5. Discussion. The product line architecture approach uses layers of components and products to develop specific simulation systems. Advantages of this type of simulation framework are that only the components needed for a specific user application are composed, reducing computational requirements; components can be replaced, easing maintenance and upgrades; and development of new models and tools is encouraged by the ability to reuse existing components as context [61]. However, in the framework the products and system compositions are normally defined in advance, and OneSAF has no inherent mechanism to enforce the assumptions and dependencies of a component if it is reused in a different context [72]. Ultimately, the developer is responsible for ensuring that a composed product is valid.

4.4. Interoperability Protocol Example: Modeling Architecture for Technology, Research, and Experimentation. Development

began of the US Army's Modeling Architecture for Technology, Research, and Experimentation (MATREX) in 2006 [41]. MATREX consists of a large set of distinct combat-related models and simulation tools interoperable via HLA or TENA [42] and is an example of how an interoperability protocol may serve as a simulation framework. (The terminology of HLA and TENA differ slightly; in this description of MATREX HLA terms are used.)

4.4.1. Design Intent. MATREX is intended to support the integration of live, virtual, and constructive models operating at either the entity or engineering level [41]. From an HLA perspective, MATREX is a persistent federation, that is, an HLA federation that after initial development is reused repeatedly, perhaps with modifications and enhancements, over a long period of time. Like many persistent federations, MATREX is not specific to any particular application; rather, it is intended to be adaptable to a range of applications as is or through extension.

4.4.2. Capabilities. The MATREX federation includes approximately 30 different simulation applications or federates. One important MATREX federate is OneSAF (described earlier), which provides broad capabilities to simulate a range of combat entity types and interactions. Also more specialized federates that model specific types of entities, interactions, and phenomena in more detail to provide for higher-fidelity or higher-resolution simulation as needed are present. Among them are the Aviation Mobility Server, the Countermine Server, the Missile Server, the Weather Server, the Vehicle Dynamics Mobility Server, the Chemical Biological Simulation Suite [73], the Comprehensive Munitions and Sensor Simulation, the Logistics Server, and the Vehicle Level Human Performance Model. Five MATREX federates, collectively known as the C3 Grid, provide services related to modeling command, control, and communications actions and effects. In addition to federates that provide modeling capabilities, MATREX includes a set of nonmodel tools that support testing and debugging, execution control, and results logging and analysis.

4.4.3. Technical Features. The MATREX architecture is broadly organized into three layers [41]. The first layer includes the federates themselves, which may include high-fidelity physics-based engineering models of battlefield vehicles, sensors, weapons, and so on, as well as tools to support system integration, testing, and analysis. The second layer is the core architecture, which includes the HLA interface, the object model used by the MATREX federations, and a software middleware layer, which stands between the interoperability protocols and the federates' model-specific code. The third layer is the distributed execution infrastructure, with includes a secure network that links MATREX federates at distributed sites. The MATREX middleware is common across the federates; they communicate through it at runtime. It includes an application programming interface and code-generated software that abstracts the details of the interoperability protocol and the object model [74].

The MATREX Federation Object Model (FOM) was initially based on the Real-time Platform Reference FOM, an HLA FOM that specifies as closely as possible the same entities, attributes, interactions, and parameters defined in the DIS protocol [75]. The MATREX FOM has been extended with objects and interactions of special interest to the MATREX user community.

4.4.4. Example Applications. MATREX applications have included integration of human behavior models into the MATREX architecture [76], support of distributed test events for network centric warfare systems [77], and networked effects command and control [41].

4.4.5. Discussion. An important feature of the MATREX framework is its middleware, which provides an interface between the federates and the interoperability protocol being used to connect them. The middleware is intended to reduce or eliminate the need to modify the federates in the event of changes to the interoperability protocol and its support software, for example, different versions of the RTI, as well as differences between object models. The MATREX version of the HLA RTI has a variety of parameters that allow users to configure HLA services, such as Data Distribution Management, as best suits their application [78]; proper configuration of the MATREX RTI for Data Distribution Management requires some care [79].

4.5. Object Model Example: Base Object Models. The Base Object Model (BOM) specification (the description of Base Object Models given here is adapted from portions of [45]), which was standardized by Simulation Interoperability Standards Organization (SISO) in 2006, is intended to enable model developers and simulation engineers to create modular and composable conceptual models and object models which can be used in the design or specification of executable simulations or simulation environments [44].

4.5.1. Design Intent. The BOM standard is meant to provide a standardized means to represent important aspects of a conceptual model. Conceptual models in general are not executable; rather they are normally developed as a precursor to the subsequent development of an executable model [19, 80]. Conceptual models are used to document and communicate "what is to be represented (in the executable model), the assumptions limiting those representations, and other capabilities needed to satisfy user's requirements" [81].

A BOM model (The term Base Object Model and the acronym BOM are used in the literature to refer to both the standard and to a particular instance of a conceptual model or object model developed according to the standard. To reduce the potential for confusion, in this paper we will used the phrase "BOM model" for the latter meaning, even though that phrase is technically redundant in that the "M" in BOM stands for "model".) is intended to be a "reusable package of

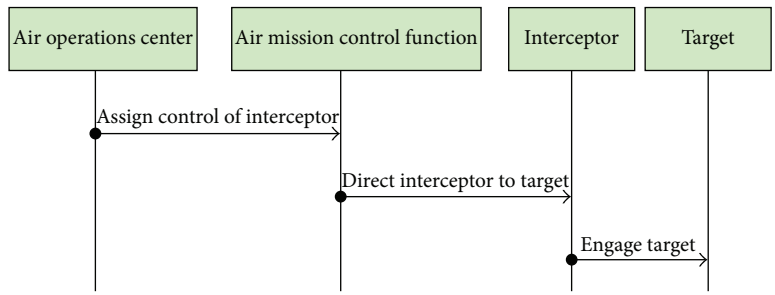


Figure 8: BOM pattern of interplay example [45].

information representing an independent pattern of simulation interplay" that will improve "interoperability, reuse, and composability, by providing "patterns" and "components" of simulation interplay that can be used as building blocks in the assembly of simulations and enterprises of simulations" [44]. The BOM standard is intended to be used for purposes such as specifying system functionality, representing scenario sequences of events, and defining reusable class structures. The primary utility of the standard is as a viable mechanism for representing conceptual models in a manner independent of a specific distributed simulation interoperability protocol or software architecture.

4.5.2. Capabilities. An example of how a BOM model can be used to document a scenario sequence of events is illustrated in Figure 8, which is a Unified Modeling Language sequence diagram [82]. The figure shows a simplified version of the directed engagement portion of a combat close air support mission. Across the top of sequence diagram several generic conceptual entities have been identified: AirOperationsCenter, AirMissionControlFunction, Interceptor, and Target. In the BOM each of these conceptual entities is mapped to specific object model classes within an executable model that can support the conceptual entity, if the latter are known. For example, conceptual entity AirOperationsCenter could be mapped to a specific Combined Air Operations Center object class, AirMissionControlFunction to an E-3 Sentry Airborne Warning and Control System aircraft object class, Interceptor to an F-22 Raptor object class, and Target to a MiG-29 Fulcrum object class. The mapping from conceptual entities to specific object model classes can be changed, enhancing the reusability of the BOM model in that the same BOM model could be reused in different scenarios with mappings from the conceptual entities to different scenario-specific object model classes.

Figure 8 also shows the archetypical sequence of interactions (the "pattern of interplay") that occur among the conceptual entities, for example, AssignControlOfInterceptor. Within a BOM model, actions can be specified as either a conceptual event, which defines a message or trigger that occurs within the simulation or as another conceptual model pattern providing a more detailed representation of the action. Conceptual events within a BOM model can map to specific object model class attribute updates or to interaction classes. The mappings from conceptual entities and conceptual events to supporting object model classes provide a useful mechanism when choosing the underlying simulations and systems that are needed to support an exercise.

4.5.3. Technical Features. Two types of BOM models have been defined [44]. Interface BOM models have messages and triggers related to one or more classes of objects and provide a reusable pattern of interplay. Encapsulated BOM models include additional information such as behaviors for modeling. Multiple BOM models may be composed to form a complete conceptual model. Composite BOM models can be converted to an HLA Federation Object Model to support interoperability through the HLA interoperability protocol [83]. However, the conceptual modeling and model mapping features of the BOM standard are architecture-independent characteristics not provided by an HLA Federation Object Model.

The mappings from the BOM model's conceptual entities may be either the class definitions of other BOM models or more specific object models, such as HLA Federation Object Models or TENA Logical Range Object Models. The latter specific object models include protocol-specific implementation details that are not intended to be part of a BOM model.

4.5.4. Example Applications. In the United States and Europe, BOM models have been developed for several military applications. A selected sample of those applications is as follows.

(1) Joint Composable Object Models: BOM models were used as conceptual models in a project developing object models usable in multiple simulation architectures [84].

(2) Midrange Ballistic Attack Munition: BOM models were used as the conceptual model of the console operator of a hybrid German-Israeli weapon system.

(3) Template Drive Code Generator: BOM models were used as conceptual models to describe component behavior in a Dutch simulation component framework [85].

(4) Surface vessel navigation: BOM models were used as conceptual models for a maritime surface vessel navigation simulation in Turkey [86].

(5) Torpedo engineering: BOM models were used as conceptual models for reusable components implementing underwater acoustic models [87].

(6) Airborne Electronic Attack: BOM models developed from US Department of Defense Architecture Framework (DoDAF) views were used as conceptual models for the airborne electronic attack architecture for the US Air Force [88].

(7) Human behavior modeling: BOM models have been proposed as a standard mechanism for encoding and documenting multiresolution human behavior models [89].

4.5.5. Discussion. It is not clear that the BOM standard fully qualifies as a simulation framework as defined earlier. The BOM standard is for component specifications (component models and object models), not components, and it does not provide an explicit integral mechanism for linking components developed according to the standard. Instead, the BOM standard assumes that some other framework, usually one of the distributed simulation interoperability protocols such as HLA or TENA, will serve to link the components that are developed to the BOM standard.

BOM models are intended to supplement the semantic information present in an interoperability protocol object model, but mappings from BOM models to ontologies have been proposed to provide additional semantic content beyond that in the BOM models themselves [90].

4.6. Integrative Environment Example: ModelCenter. ModelCenter (*ModelCenter* is a commercial product of Phoenix Integration.) is a graphical toolkit and software framework for engineering design integration and optimization. It allows exploration of the design space to identify promising approaches to the problem under analysis.

4.6.1. Design Intent. ModelCenter is intended to facilitate the integration of multiple distinct models used in the process of engineering design. It supports the creation of linked applications by enabling the automatic runtime exchange of data between the different models. *ModelCenter* was designed to provide ease of use, in the form of integration with existing analysis and modeling packages allowing direct reuse of existing models, improved engineering design, via support for early identification of design problems and analysis of the trade space, and error reduction, by automating the process of creating data exchange channels between models [53].

4.6.2. Capabilities. ModelCenter's capabilities fall into three primary areas [53, 91].

(1) Model Wrapping. To link the multiple models that make up the composite model, automated data exchange channels are created by "wrapping" the component models. Three different types of wrapper support various forms of data. *File wrapping* wraps existing data files through the creation of input and output files for the various components and stages of the model by identifying which items in a file serve as input to each model, and what each model must output for use by subsequent stages. *Script wrapping* is accomplished via the *ModelCenter* application programming interfaces to handle formats such as the Microsoft COM protocol. *Custom wrapping* is available through tailored applications written in high-level languages such as C++ or Java. This method is employed when third party models provide APIs that allow access to internal functions.

(2) Visual Model/Process Integration. The graphical interface allows the construction of designs, through the linking of applications and models, and the execution of simulations. Linking is accomplished via drag-and-drop interfaces and support not only direct connections but also conditional and looping associations between models. Execution of and dataflow between linked components are supported directly for models such as Excel, MATLAB, or common computer-aided design and computer-aided engineering tools, as well additional custom model types. Other components may require scripting or user intervention for execution.

(3) Analysis and Optimization. Once integrated, the composite model may be executed repeatedly for trade space exploration or design optimization. *ModelCenter* is able to process both discrete and continuous variables with the goal of minimizing, maximizing, or solving for given attributes, while satisfying specified constraints.

4.6.3. Technical Features. Model wrapping is available via both a separate tool and a native component of *ModelCenter*. The latter provides the ability to wrap ASCII files via a point-and-click interface that identifies variables within the files. For files that have a defined structure, such as FORTRAN name lists and name-value pairs, matching patterns may be created to automate the wrapping process. Additionally, it will process files from commercial computer-aided engineering products. Scripting capability and functions are also available.

Along with the previously mentioned drag-and-drop capability and simulation control functions, the graphical interface provides a robust set of logical connectors, including if-then, switches, parallel branches, and looping, which can be used to link component models. Characterization of the type of data exchange across the links (values, arrays, files, or objects) is also possible.

ModelCenter can generate and analyze response surfaces. The data composing the surface may come from previous simulation runs or may be approximations created within the toolkit. The process supports curve fit types such as the polynomial or Kringing methods. The resulting response surface may then be incorporated into the set of linked models. Response surfaces created by simulations may be incorporated in lieu of the simulation.

Another *ModelCenter* component performs optimization of variables in the design. The optimizer employs genetic algorithms and recommends algorithmic approaches and variable selection from questions posed to engineers about the design and its goals. Final results are provided via optimization reports. A software development kit that allows

Table 1: Framework types summary; adapted and updated from [11].

Framework type	Components	Composition mechanism
Common library	Models implemented as software modules	Component interfaces defined by components or framework; components linked into common executable; data exchanged via method calls
Product line architecture	Software modules	Component interfaces defined by framework; components linked into common executable; data exchanged via method calls
Interoperability protocol	Independent executable	Components execute independently as separate processes; data exchanged via network messages
Object model	Conceptual model	None; connection depends on mapping and implementing conceptual module within another framework
Formal	Formal model	Interpreter for formal models
Integrative environment	Model implemented as file, spreadsheet, or software module	Components "wrapped" with special interface software; components linked into common executable; data exchanged via method calls

users to implement their own optimization routines, which may be retained for future use, is also available.

4.6.4. Example Applications. *ModelCenter* has been used for a variety of applications. A multidisciplinary optimization of the design of an autonomous underwater vehicle was performed using multiobjective genetic optimization routines. Electronics, hull geometry and performance, feasibility, cost, and risk models were integrated [92]. An optimization of a conceptual design for a helicopter for exploring Saturn's moon Titan considered typical aircraft design questions in an unfamiliar environment. Genetic algorithms were applied to assess the best solution considering the helicopter's aerodynamics, performance, propulsion, power, weights, and sizes [93]. *ModelCenter* was used as the integrative environment for a life-cycle cost analysis model for the National Aeronautics and Space Administration's Constellation spacecraft [94].

4.6.5. Discussion. *ModelCenter* is generally considered to be versatile and easy to use. In the autonomous underwater vehicle design application, it was seen as "essential for the design of highly integrated systems" [92]. The Titan helicopter conceptual model study reported success in optimizing both discrete and continuous variables, using gradient-based and stochastic optimizers. On the other hand, the study encountered loss of resolution on some low influence variables, which were not optimized as well as others [93].

5. Summary and Conclusions

A software framework is an architecture or infrastructure intended to enable the integration and interoperation of software components. Simulation frameworks are software frameworks specifically intended simplify the process of assembling a complex model or simulation system from simpler component models as well as to promote the reuse of the component models. At least six distinct types of software frameworks for model composition have been developed and examples of each exist. The different simulation framework types have different components, processes for composing models, and intended applications. Table 1 summarizes the six simulation framework types. For each type the table identifies the components that can be composed within the framework and the mechanism used to compose them.

The primary conclusion of this review is that best type of simulation framework to use depends on the application. A simulation developer intending to compose conventional software components may consider a common library or a product line architecture. If independently executing models are to be linked, an interoperability protocol is advised. Improvement in semantic composability may result from the use of an object model framework. A formal framework can enable formal mathematical analysis of component composability. Finally, an integrative environment can quickly connect a diverse set of files and models to support engineering analysis.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

Funding for the preparation of this paper and the project it is based on was provided by the Republic of Korea's Agency for Defense Development and REALTIMEVISUAL Inc. Their support is gratefully acknowledged.

References

[1] M. D. Petty, K. L. Morse, W. C. Riggs, P. Gustavson, and H. Rutherford, "A reuse lexicon: terms, units, and modes in M&S asset reuse," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2010.

[2] G. Allen, J. J. Daly, M. Heaphy, W. Barge, B. Halayko, and K. Gupton, "Making modeling and simulation reuse attractive," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 2161–2171, Orlando, Fla, USA, December 2013.

[3] H. Mili, A. Mili, S. Yacoub, and E. Addy, *Reuse Based Software Engineering: Techniques, Organization, and Controls*, John Wiley & Sons, New York, NY, USA, 2002.

[4] Y. Yu, T. Li, Q. Liu, and F. Dai, "Approach to modeling components in software architecture," *Journal of Software*, vol. 6, no. 11, pp. 2196–2200, 2011.

[5] K. L. Morse, M. D. Petty, P. F. Reynolds, W. F. Waite, and P. M. Zimmerman, "Findings and recommendations from the 2003 composable mission space environments workshop," in *Proceedings of the Simulation Interoperability Workshop*, pp. 313–323, Arlington, Va, USA, April 2004.

[6] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process, and Organization for Success*, ACM Press, New York, NY, USA, 1997.

[7] M. D. Petty and E. W. Weisel, "A composability lexicon," in *Proceedings of the Simulation Interoperability Workshop*, pp. 181–187, Orlando, Fla, USA, March-April 2003.

[8] E. W. Weisel, R. R. Mielke, and M. D. Petty, "Validity of models and classes of models in semantic composability," in *Proceedings of the Simulation Interoperability Workshop*, pp. 526–536, Orlando, Fla, USA, September 2003.

[9] M. D. Petty, E. W. Weisel, and R. R. Mielke, "Composability theory overview and update," in *Proceedings of the Simulation Interoperability Workshop*, pp. 431–437, San Diego, Calif, USA, April 2005.

[10] P. Bunus and P. Fritzson, "Automated static analysis of equation-based components," *Simulation*, vol. 80, no. 7-8, pp. 321–345, 2004.

[11] E. W. Weisel, M. D. Petty, and R. R. Mielke, "A survey of engineering approaches to composability," in *Proceedings of the Simulation Interoperability Workshop*, pp. 722–731, Arlington, Va, USA, April 2004.

[12] A. E. Henninger, K. L. Morse, M. L. Loper, and R. D. Gibson, "Developing a process for M&S standards management within DoD," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, Orlando, Fla, USA, November-December 2009.

[13] R. C. Hofer and M. L. Loper, "DIS today," *Proceedings of the IEEE*, vol. 83, no. 8, pp. 1124–1137, 1995.

[14] J. S. Dahmann, F. Kuhl, and R. Weatherly, "Standards for simulation: as simple as possible but not simpler the high level architecture for simulation," *SIMULATION*, vol. 71, no. 6, pp. 378–387, 1998.

[15] E. T. Powell and J. R. Noseworthy, "The test and training enabling architecture," in *Engineering Principles of Combat Modeling and Distributed Simulation*, A. Tolk, Ed., pp. 449–477, John Wiley & Sons, Hoboken, NJ, USA, 2012.

[16] C. M. Banks, "Introduction to modeling and simulation," in *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*, J. A. Sokolowski and C. M. Banks, Eds., pp. 1–24, John Wiley & Sons, Hoboken, NJ, USA, 2010.

[17] M. D. Petty, "Behavior generation in semi-automated forces," in *The PSI Handbook of Virtual Environment Training and Education: Developments for the Military and Beyond, Volume 2: VE Components and Training Technologies*, D. Nicholson, D. Schmorrow, and J. Cohn, Eds., pp. 189–204, Praeger Security International, Westport, Conn, USA, 2009.

[18] D. Krieger and R. M. Adler, "The emergence of distributed component platforms," *Computer*, vol. 31, no. 3, pp. 43–53, 1998.

[19] M. D. Petty, "Verification, validation, and accreditation," in *Modeling and Simulation Fundamental: Theoretical Underpinnings and Practical Domains*, J. A. Sokolowski and C. M. Banks, Eds., pp. 325–372, John Wiley & Sons, Hoboken, NJ, USA, 2010.

[20] D. Russell and W. McQuay, "The joint modeling and simulation system: a common modeling architecture for the DoD," in *Proceedings of the Winter Simulation Conference (WSC '93)*, pp. 984–988, Los Angeles, Calif, USA, December 1993.

[21] V. K. Handley, P. M. Shea, and M. Morano, "An introduction to the Joint Modeling and Simulation System (JMASS)," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2000.

[22] B. McCauley, J. Hill, P. Gravitz, and D. McFarland, "JMASS98—engagement level simulation framework++," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, March 2000.

[23] R. J. Meyer, "Joint Modeling and Simulation System (JMASS): what it does, ... and what it doesn't!," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, March 2001.

[24] B. Haynes, T. Carroll, D. Tollison, W. Kendrick, and A. Salter, "Development of the Missile Component Simulation Library (MCLib) for tactical missile simulation," in *Proceedings of the Huntsville Simulation Conference*, Huntsville, Ala, USA, October 2003.

[25] K. Shibuya, "A framework of multi-agent-based modeling, simulation, and computational assistance in an ubiquitous environment," *SIMULATION*, vol. 80, no. 7-8, pp. 367–380, 2004.

[26] H.-Y. Tyan, A. Sobeih, and J. C. Hou, "Design, realization and evaluation of a component-based, compositional network simulation environment," *SIMULATION*, vol. 85, no. 3, pp. 159–181, 2009.

[27] D. C. Gross, L. D. Stuckey, and R. R. Macala, "Implications of megaprogramming for the training systems community," in *Proceedings of the 17th Interservice/Industry Training Systems and Education Conference*, pp. 88–97, Albuquerque, NM, USA, November 1995.

[28] L. D. Stuckey, G. M. Kamsickas, and W. V. Tucker, "An approach for a configurable and accessible environment model," in *Proceedings of the 18th Interservice/Industry Training Systems and Education Conference*, pp. 492–502, Orlando, Fla, USA, December 1996.

[29] J. Mann, A. York, and B. Shankle, "Integrating physics-based damage effects in urban simulations," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 653–662, Orlando, Fla, USA, December 2004.

[30] P. Dumanoir, R. Parrish, and H. A. Sotomayor, "LVC interoperability: where is the best place to start?" in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 666–675, Orlando, Fla, USA, November 2007.

[31] J. Lanman, B. Becker, and W. Samper, "Joint service partnership: extending the live training transformation product line," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 1743–1754, Orlando, Fla, USA, November-December 2009.

[32] D. Parsons, J. Surdu, and B. Jordan, "OneSAF: a next generation simulation modeling the contemporary operating environment," in *Proceedings of the European Simulation Interoperability Workshop*, Toulouse, France, June 2005.

[33] M. L. Loper and C. Turnitsa, "History of combat modeling and distributed simulation," in *Engineering Principles of Combat ModelIng and Distributed Simulation*, A. Tolk, Ed., pp. 331–355, John Wiley & Sons, Hoboken, NJ, USA, 2012.

[34] A. Ceranowicz, M. Torpey, B. Helfinstine, and J. Evans, "Reflections on building the joint experimental federation," in

Proceedings of the Interservice/Industry Training, Simulation, and Education Conference, pp. 1349–1359, Orlando, Fla, USA, December 2002.

[35] M. C. Fischer, "Joint training confederation," in *Proceedings of the 1st International Simulation Technology and Training Conference*, Melbourne, Australia, March 1996.

[36] J. A. Tufarolo and E. H. Page, "Evolving the VV&A process for the ALSP joint training confederation," in *Proceedings of the Winter Simulation Conference (WSC '96)*, pp. 952–958, December 1996.

[37] M. B. Pettitt, B. F. Goldiez, M. D. Petty, S. Rajput, and H. Tu, "The combat trauma patient simulator," in *Proceedings of the Simulation Interoperability Workshop*, pp. 936–946, Orlando, Fla, USA, March 1998.

[38] M. D. Petty and P. S. Windyga, "A high level architecture-based medical simulation system," *SIMULATION*, vol. 73, no. 5, pp. 281–287, 1999.

[39] M. B. Pettitt, M. Mayo, and J. Norfleet, "Medical simulation training simulations," in *The PSI Handbook of Virtual Environment Training and Education: Developments for the Military and Beyond, Volume 3: Integrated Systems, Training Evaluations, and Future Directions*, J. Cohn, D. Nicholson, and D. Schmorrow, Eds., pp. 99–106, Praeger Security International, Westport, Conn, USA, 2009.

[40] H. A. Marshall, "SAF in CATT training systems, update 1999," in *Proceedings of the 8th Conference on Computer Generated Forces and Behavioral Representation*, pp. 277–283, Orlando , Fla, USA, May 1999.

[41] T. Hurt, T. McKelvy, and J. McDonnell, "The modeling architecture for technology, research, and experimentation," in *Proceedings of the Winter Simulation Conference (WSC '06)*, pp. 1261–1265, Monterey, Calif, USA, December 2006.

[42] U.S. Army Research, Development, and Engineering Command, "MATREX simulation architecture," in *Proceedings of the Department of Defense Modeling and Simulation Conference*, Orlando, Fla, USA, March 2008.

[43] S. Jain and C. R. McLean, "Components of an incident management simulation and gaming framework and related developments," *SIMULATION*, vol. 84, no. 1, pp. 3–26, 2008.

[44] Simulation Interoperability Standards Organization, *Base Object Model (BOM) Template Specification*, SISO-STD-003-2006, 2006, http://www.sisostds.org/.

[45] M. D. Petty and P. Gustavson, "Combat modeling with the high level architecture and base object models," in *Engineering Principles of Combat Modeling and Distributed Simulation*, A. Tolk, Ed., pp. 413–448, John Wiley & Sons, Hoboken, NJ, USA, 2012.

[46] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, San Diego, Calif, USA, 2nd edition, 2000.

[47] A. W. Wymore, *Model-Based Systems Engineering*, CRC Press, Boca Raton, Fla, USA, 1993.

[48] E. W. Weisel, M. D. Petty, and R. R. Mielke, "A comparison of DEVS and semantic composability theory," in *Proceedings of the Simulation Interoperability Workshop*, pp. 956–964, San Diego, Calif, USA, April 2005.

[49] R. R. Mielke, M. D. Petty, and E. W. Weisel, "A comparison of model-based systems engineering and cmposability theory," in *Proceedings of the Huntsville Simulation Conference*, pp. 300–308, Huntsville, Ala, USA, October 2005.

[50] L. Yilmaz, "Verifying collaborative behavior in component-based DEVS models," *SIMULATION*, vol. 80, no. 7-8, pp. 399–415, 2004.

[51] M. N. Alpdemir, "SiMA: a discrete event system specification-based modeling and simulation framework to support model composability," *Journal of Defense Modeling and Simulation*, vol. 9, no. 2, pp. 147–160, 2012.

[52] B. Delinchant, F. Wurtz, D. Magot, and L. Gerbaud, "A component-based framework for the composition of simulation software modeling electrical systems," *SIMULATION*, vol. 80, no. 7-8, pp. 347–356, 2004.

[53] M. Bigley, C. Nelson, P. Ryan, and W. H. Mason, "Tutorials and examples of software integration techniques for aircraft design using model center," Tech. Rep. MAD 99-06-02, Virginia Polytechnic Institute and State University, Blacksburg, Va, USA, 1999.

[54] J. D. Gardiner, "Real-time hardware-in-the-loop testing with common simulation framework," in *Proceedings of the Department of Defense High Performance Computing Modernization Program Users Group Conference*, pp. 379–381, Seattle, Wash, USA, July 2008.

[55] B. Gossage, J. Lucas, D. Harbison, and B. Haynes, "Towards a parallel simulation framework using modern, multi-node linux clusters utilizing MPI," in *Proceedings of the Huntsville Simulation Conference*, Huntsville, Ala, USA, October 2004.

[56] S. Speigle, B. Haynes, T. P. Etherege et al., "High-fidelity system simulation technology for interactive networked missile systems," in *Proceedings of the Huntsville Simulation Conference*, Huntsville, Ala, USA, October 2004.

[57] S. Speigle and P. White, "Path forward for digital simulation technologies," in *Proceedings of the Huntsville Simulation Conference*, Huntsville, Ala, USA, October 2004.

[58] J. T. Taylor, *Force-on-Force Attrition Modeling*, Military Operations Research Society (MORS), Alexandria, Va, USA, 1981.

[59] O. Balci, J. D. Arthur, and W. F. Ormsby, "Achieving reusability and composability with a simulation conceptual model," *Journal of Simulation*, vol. 5, no. 3, pp. 157–165, 2011.

[60] R. L. Wittman and A. J. Courtemanche, "The OneSAF product line architecture: an overview of the products and process," in *Proceedings of the Simulation Technology and Training Conference (SimTecT '02)*, Melbourne, Australia, May 2002.

[61] L. A. Rieger and C. T. Harrison, "Use of product line architecture for multi-use simulations," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 1424–1431, Orlando, Fla, USA, December 2002.

[62] A. J. Courtemanche and R. L. Wittman, "OneSAF: a product line approach for a next-generation CGF," in *Proceedings of the 11th Conference on Computer Generated Forces and Behavioral Representation*, pp. 349–361, Orlando, Fla, USA, May 2002.

[63] A. J. Courtemanche and R. B. Burch, "Using and developing object frameworks to achieve a composable CGF architecture," in *Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation*, pp. 49–62, Orlando, Fla, USA, May 2000.

[64] R. L. Wittman and C. T. Harrison, "OneSAF: a product line approach to simulation development," in *Proceedings of the European Simulation Interoperability Workshop*, London, UK, June 2001.

[65] D. J. Franceschini, K. R. Hawkes, and S. Graffuis, "System composition in OneSAF," in *Proceedings of the Simulation Interoperability Workshop*, Kissimmee, Fla, USA, March-April 2003.

[66] B. Grainger and C. Henderson, "Battlespace composition in the OneSAF objective system," in *Proceedings of the Simulation Interoperability Workshop*, Kissimmee, Fla, USA, March-April 2003.

[67] P. Dumanoir and J. Bittel, "Mission Planning and Rehearsal System (MPARS)—part of the OneSAF tool set," in *Proceedings of the Simulation Interoperability Workshop*, Arlington, Va, USA, April 2004.

[68] J. A. Giampapa, K. Sycara, S. Owens et al., "Extending the OneSAF testbed into a C4ISR testbed," *SIMULATION*, vol. 80, no. 12, pp. 681–691, 2004.

[69] K. M. Kelly, C. Finch, D. Tartaro, and S. Jaganathan, "Creating a world war II combat simulator using OneSAF objective system," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 510–520, Orlando, Fla, USA, December 2006.

[70] L. Wihl and M. Varshney, "A virtual cyber range for cyber warfare analysis and training," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 1932–11940, Orlando, Fla, USA, December 2012.

[71] G. Dukstein, J. Watkins, K. Le, and H. Gonzalez, "Extending construction simulators through commonality and innovative research," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 2355–2365, Orlando, Fla, USA, December 2013.

[72] R. G. Bartholet, D. C. Brogan, P. F. Reynolds, and J. C. Carnahan, "In search of the philosopher's stone: simulation composability versus component-based software design," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2004.

[73] M. J. O'Connor, J. D. Fann, and D. L. Jones, "CB defense modeling & simulation (M&S) suite," in *Proceedings of the Huntsville Simulation Conference*, Huntsville, Ala, USA, October-November 2007.

[74] C. Metevier, G. Gaughan, S. Gallant, K. Truong, and G. Smith, "A path forward to protocol independent distributed M&S," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 2764–2775, Orlando, Fla, USA, November-December 2010.

[75] Simulation Interoperability Standards Organization, *SISO-STD-001.1-1999: Real-time Platform Reference Federation Object Model (RPR FOM 1.0)*, 1999.

[76] B. Kellihan and R. Washington, "C3HPM: an application of IMPRINT with the MATREX," in *Proceedings of the Conference on Behavior Representation in Modeling and Simulation*, pp. 220–226, Arlington, Va, USA, May 2004.

[77] K. G. LeSueur, K. Yetzer, M. Stokes, A. Krishnamurthy, and A. Chalker, "Distributed tests: an army perspective," in *Proceedings of the HPCMP Users Group Conference (UGC '06)*, pp. 337–344, Denver, Colo, USA, June 2006.

[78] J. Lewis and D. Vagiakos, "Combating network load in high entity count federations," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 1351–1358, Orlando, Fla, USA, November-December 2009.

[79] J. Lewis, K. Do, and D. Vagiakos, "DDM explained: lessons for data distribution management developers and strategists," in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, pp. 2911–2920, Orlando, Fla, USA, November-December 2010.

[80] F. D. McKenzie, "Systems modeling: analysis and operations research," in *Modeling and Simulation Fundamental: Theoretical Underpinnings and Practical Domains*, J. A. Sokolowski and C. M. Banks, Eds., pp. 147–180, John Wiley & Sons, Hoboken, NJ, USA, 2010.

[81] Institute for Electrical and Electronic Engineers, *IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process*, IEEE Std 1516.3-2003, 2003.

[82] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading, Mass, USA, 1999.

[83] B. Möller, P. Gustavson, B. Lutz, and B. Löfstrand, "Making your BOMs and FOM modules play together," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2007.

[84] R. Rheinsmith, J. Wallace, W. Bizub et al., "Joint composable object model and LVC methodology," in *Proceedings of the MODSIM World Conference*, Virginia Beach, Va, USA, October 2009.

[85] R. Jansen, L. Prins, and W. Huiskamp, "Template driven code generator for HLA middleware," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2007.

[86] Y. Timar, I. Taşdelen, S. Akgün, O. Dikenelli, and I. Bikmaz, "Using BOM in conceptual modelling of distributed simulation projects," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2007.

[87] J. Watkins, P. Lallement, and D. Diederich, "Applying the base object model to the torpedo enterprise advanced modeling and simulation initiative," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2007.

[88] P. Gustavson and T. Chase, "Building composable bridges between the conceptual space and the implementation space," in *Proceedings of the Winter Simulation Conference (WSC '07)*, pp. 804–814, Washington, DC, USA, December 2007.

[89] C. Turnista, P. Gustavson, and C. Blais, "Exploring multi-resolution human behavior modeling using base object models," in *Proceedings of the Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2010.

[90] H. Benali and N. B. Ben Saoud, "Towards a component-based framework for interoperability and composability in modeling and simulation," *Simulation*, vol. 87, no. 1-2, pp. 133–148, 2011.

[91] K. A. Geiselhart, L. P. Ozoroski, J. W. Fenbert, E. W. Shields, and W. Li, "Integration of multifidelity multidisciplinary computer codes for design and analysis of supersonic aircraft," in *Proceedings of the 49th AIAA Aerospace Sciences Meeting*, Orlando, Fla, USA, January 2011.

[92] M. Martz and W. L. Neu, "Multi-objective optimization of an autonomous underwater vehicle," in *Proceedings of the OCEANS Conference*, Quebec, Canada, September 2008.

[93] N. Brown, T. Percy, J. Pimentel, S. Steffes, K. Taya, and J. R. Olds, *Conceptual Design of a Titan Helicopter and Architecture*, School of Aerospace Engineering, Georgia Institute of Technology, 2003, http://www.phoenix-int.com/documents/pubDownload.php/Titan_Helicopter.pdf.

[94] A. Prince, H. Rose, and J. Wood, "Constellation program Life-Cycle Cost Analysis Model (LCAM)," in *Proceedings of the International Society of Parametric Analysis/Society of Cost Estimating and Analysis Joint Annual Conference*, pp. 1–12, Nordwijk, The Netherlands, May 2008.