

BUILDING COMPOSABLE BRIDGES BETWEEN THE CONCEPTUAL SPACE AND THE IMPLEMENTATION SPACE

Paul Gustavson
Tram Chase

SimVentions, Inc.
11905 Bowman Drive, Suite 502
Fredericksburg, VA 22408, U.S.A.

ABSTRACT

Often times the process and effort in building interoperable simulations and applications can be arduous. Invariably the difficulty is in understanding what is intended. This paper introduces the notion of composable bridges as a means to help transition abstract ideas or concepts into concrete implementations.

We examine the key elements to achieve composability, which includes the direction provided by a process, the importance of a conceptual model, the use of patterns to help characterize reusable aspects of a design, the importance of having good discovery metadata and well-defined interfaces that can be implemented, the use of components, and the practical use of libraries and tools. We suggest that of all these elements a properly documented conceptual model provides the basis for formulating a composable bridge, and that things like patterns, discovery metadata, and interfaces play a key role. We take a look at specific standard known as the Base Object Model (BOM) and examine how it provides a means to define a composable bridge. We explore how BOMs, in this capacity, can be aggregated and used (and reused) to support the creation of concrete implementations. We also explore how such composability helps to achieve various levels of interoperability.

1 INTRODUCTION

Whether we are architects, developers, analysts, educators, or managers, composability is a desire we all seem to share. There seems to be an insatiable need to assemble capabilities and develop meaningful functionality from the knowledge, tools, standards and components that we have available to us.

For some, the desire to create and compose is a trait we have had since we were young (see Figure 1). And for many, it has never left us. We have simply transferred this early desire to the context of our work as we pursue

the creation of innovative things such as models, software applications, distributed simulations, complex systems, scenarios, games, stories, virtual experiences or new found realities.



Figure 1: Composability.

Composability is defined by the DoD M&S Master Plan as “the ability to rapidly select and assemble components to construct meaningful simulation systems to satisfy specific user requirements.”

There are three aspects of composability that this definition identifies:

1. The selection and use of components
2. The construction of meaningful applications, and
3. The satisfaction of specific user requirements

We will briefly explore each of these.

1.1 The Selection and Use of Components

This first aspect of composability can be compared to the Lego® mindset as illustrated in Figure 2 in which blocks selected from the same source (i.e., Lego® bins) can be used and reused to construct various creations. The Lego® bricks serve as components.

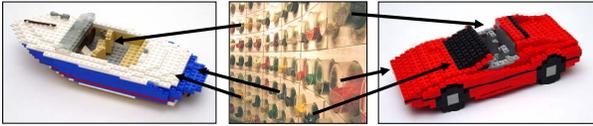


Figure 2: Composability represented using Lego® bricks.

1.2 The Construction of Meaningful Applications

Composability all starts as an “idea” in the conceptual space. For a child, such ideas start as a glimmer in the mind’s eye; a mental picture of something that they could potentially create from the bricks that lie in front of them. The bricks are only an enabler, the fuel, for bringing to life what starts out in the imagination. However, during the process of building they may continue to formulate their conceptual model mentally, until, at last, a meaningful physical creation is complete. This is where, for a child, the magic happens; when their idea has become something real and tangible. This is where the conceptual space meets the implementation space. The question though, is does it satisfy what was intended?

1.3 The Satisfaction of Specific User Requirements

Once a Lego® composition is complete, a typical child will revel in their creation. Eyeing it as if it were a prize; satisfied in what they have built but only if it meets their desired requirements.

What happens for a child is not much different than what happens in the workplace. Ideas are formulated sometimes captured on paper, as diagrams via a tool, or as drawings on a white board. And if the passion and drive are there, the ideas are churned and worked until a satisfying product is conceived, whether it be a software application, a PowerPoint, a proposal, or new system or simulation. But what we create truly isn’t satisfying unless it has met our requirements.

Thus, there is a point for any successful project where what has been implemented is compared to what was conceptualized. Consider the questions that are pondered at the conclusion of a project, especially large projects:

- How did it go?
- Did we meet all our requirements?
- Was the sponsor happy with the results?

It’s intriguing that we often wait to ask these questions until after a project is completed. This may be a telltale sign that those involved in the project are perhaps not communicating early enough regarding what is intended (i.e., the concept) and they are not subsequently correlating those intentions with what they are building or using (e.g., components) in their effort to realize an implementation. What is needed, therefore, is a means to assist in bridging

well defined concepts with what is ultimately being implemented. Considering that the process and effort in building interoperable simulations and applications can be arduous, this need for bridging the conceptual plane to the implementation plane through composability is important.

2 FORMULATING COMPOSABLE BRIDGES

Typically, a bridge is defined as “a structure spanning and providing passage over a gap or barrier.” In music it is defined as “a transitional passage connecting two subjects or movements” (Dictionary.com). And in the context of development, a bridge should be defined as “a means to span and provide a way to connect an idea (i.e., initial concept) to something implementable.” This idea is conveyed in Figure 3.

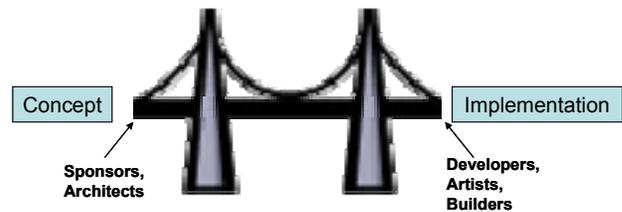


Figure 3: The development bridge.

For projects that fail, it’s easy to determine that a bridge encouraging communication among stakeholders was never properly formulated. It fell short. But for projects that succeed, a bridge is formed, which makes the journey however long or short, possible to bare. In fact, what we all want for any project is to be able to bridge quickly and easily from initial concept to implementation. The question is how can that best be done?

What if such bridges could be defined structurally as means to convey a concept that can be mapped to one or more potential implementations? What if the common desired behaviors (understood first conceptually) could be individually defined, described and cataloged providing a means to assist in communicating an idea that can be bridged to something implementable? And what if such bridges could be reused and aggregated to formulate the scaffolding needed for larger project specific bridges? Wouldn’t such use of bridges increase our likelihood for effective communication among stakeholders and for achieving successful creation of meaningful applications?

Our focus is to explore how to begin building and using composable bridges; bridges which gap ideas formulated in the conceptual space with what can be realized in the implementation space. We postulate that the conceptual model provides the basis for a composable bridge. And we consider what standards and various techniques could be applied to better achieve composability and interoperability within the M&S domain.

2.1 Why the Conceptual Model is Key

If a survey could be taken asking simulation professionals what the key elements are for composing successful simulations and interoperable applications, we might expect the following answers:

- Following a process is important
- Requirements and good design are crucial
- The use of components is what helps expedite development
- Having the right tools is key
- Complying to standards ensures success
- Effective communication is what it takes.

This is a compelling list, and it is hard to argue the merit of any of these items for developing and integrating simulations, especially interoperable simulations. However, there is one other element often missed that is perhaps central to all of these others, and that is that best practices encourage the production of a conceptual model. For example, look at the typical process prescribed for M&S development, which is illustrated in Figure 4. This process identifies the need for requirements within Step 1, but closely examine what’s identified in Step 2. Notice this step is identified as “Perform Conceptual Analysis”? This step precedes Design and Development; Steps 3 and 4.

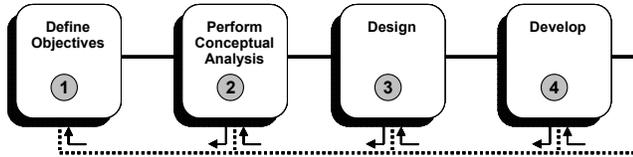


Figure 4: Common development process.

Interestingly enough the number one most common development issue is inadequate requirements and design (Gustavson 2003). In other words, most development shops are completely missing Step 2 of the process identified in Figure 4. The impact of missing Step 2 is that it often results in miscommunication and misunderstanding among stakeholders, limiting the success of a project.

Step 2’s goal is to produce conceptual models. Conceptual models identify what needs to be represented, and how things are supposed to behave. It’s this artifact that helps bridge the communication gap between multiple stakeholders, providing a common framework for collaboration and understanding. Such understanding leads to better composability, and therefore better software and simulations.

Additionally, conceptual models need to be leveraged throughout development. In other words, we need to keep coming back to it, for it ties what it is we intend to build (Objectives), with what we are designing and developing. It creates a bridge.

Consider this, if the conceptual model is not carried forward – applied, understood, visualized, and used at the various stages of development – then how will it be known that the objectives have been met and satisfied?

2.2 Discovering Patterns

The question that should then be asked is, “what should we look for when we are trying to identify and define our conceptual models?” This is where the concept of patterns comes into play. Patterns result in a solution you can reuse for supporting a common problem or need.

Patterns are nothing really new. Noted author and professor Christopher Alexander first pioneered the concept of patterns years ago when he focused on aspects for improving upon the way building projects are designed and engineered. In his landmark book titled “The Timeless Way of Building” he describes the concept as follows:

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”

Within the software engineering realm many have also embraced Alexander’s pattern concept, as evidenced by the plethora of pattern books that are available. We find that in software engineering patterns are being applied to support analysis, design and aid in refactoring. Within the M&S arena we are seeing the same type of opportunity for patterns.

Consider the overall concept of Patterns. A pattern behavior is something that occurs with consistency, which is recognized and reproduced. Fowler describes patterns as “an idea that has been useful in one practical context and will probably be useful in others.” In short, our best investments are in patterns. And patterns are key aspect of our conceptual model. Some common patterns that are employed within military M&S scenarios are depicted in Figures 5 and 6.



Figure 5: Weapon’s effect.

Typically patterns are discovered rather than invented. In this example, we unveil a common pattern that has been reused with great frequency in the DIS and HLA community. Two entities are depicted. One that fires at another. Of interest is the pattern associated to this Weapon's Effect behavior. When the firing entity propels an ordnance on the target, two reciprocal actions will typically occur. The Firing Entity, within a simulation, will then update the position of the projectile and then indicate when the munition has detonated. And then, upon detonation, the target is then responsible for sharing its damage state so that the firing entity is aware of the target's condition. This particular pattern illustrated in Figure 5 is also decorated with the various states associated to each type. It can be seen how an action can transition a state change upon each entity. This aspect of States of an entity, which is known as a State Machine, is also a key aspect of a conceptual model.

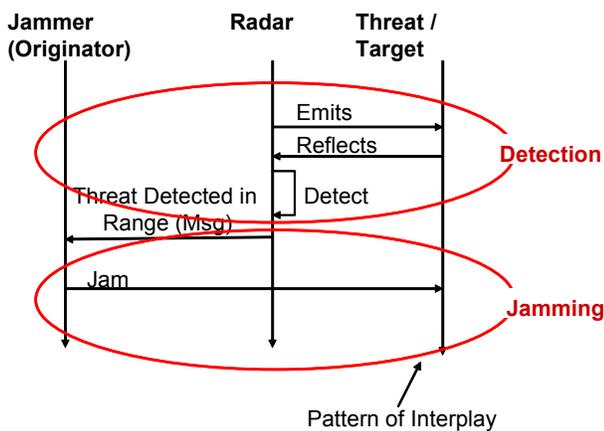


Figure 6: Jamming / detection patterns.

In the example shown in Figure 6, two patterns are revealed. We could conceivably use the “Detection” pattern for other purposes besides just “Jamming” such as “Vectoring Interceptors.” What we learned from this example is that the best way to discover a pattern is to perform a conceptual analysis on the problem space. Otherwise, rather than two patterns being revealed, we would have walked away with a single pattern which was fairly bulky, specialized, with limited reuse.

2.3 Identifying Interfaces

In achieving composability though, it's not enough to discover and document patterns. Step 3 of the process identifies that Design is an important facet to the development effort. A big part of design is to focus on the “interface” of what will be provided and what should be supported by an implementation whether that resulting implementation may be a piece of hardware, software, or a service.

Within the software and simulation engineering field an interface is often described in terms of class structures

that collectively define the inherent capabilities of an application, component or service.

Bjarnes Stroustrup, who was responsible for the creation of the C++ language, shares the following insight regarding interfaces:

“...it is essential for the software industry's health that key interfaces be well-specified and publicly available.” - Bjarne Stroustrup

Interfaces provide a contract of what is available and accessible, and provides a framework to resulting implementations (i.e., software components, simulations) that support what's described by the metadata and defined by the interface.

2.4 Applying Components

Once a desired interface is known, the logical progression is to look for available components that support the conceptual model. If candidate components are not found, then the framework for developing a new component is already at hand.

The DoD M&S composability definition, which was described previously, referred to this concept of components. Components in the M&S world, of course, are functionally different than a Lego® brick, but the goal is the same. Consider the definition for an M&S component.

“Reusable building blocks which have a known set of inputs and provide expected output behavior, but the implementation details may be hidden. Such components are useful for constructing simulations and/or providing functionality for simulation systems.” – COI M&S Metadata Focus Group

The unique thing with a Lego® brick is that it is clear how to snap it into other bricks. The inputs and expected outputs are known. We don't really care about the specific implementation aspects of the brick itself; whether it's plastic, hollow, or solid. But we do care about function and form of each brick. Therefore we look for a brick that satisfies a part of our pattern, and can adhere to our interfaces. For example we look for one that has the number of nubs that we desire to complete some portion of what we intend to create. When the brick we desire is found, there should be enough information inherent in the brick for us to know how it connects with other bricks.

We recognize that Lego® bricks are a fairly simplified example of composability. In other words, it is easy to pick up a brick and know how it can be used. Therefore, we dare not trivialize the effort associated to M&S composability as being as simple as Lego® construction. M&S components don't reflect that intuitiveness that Lego®

bricks inherently have. But what Lego® bricks and M&S components do share in common is that the inputs and output behavior of a component should be known; that is its interface should be exposed. This allows us to understand the functionality a component provides in potentially fulfilling a concept or objective. In this way a component provides a means to satisfy a composable bridge.

2.5 Leveraging Metadata

Another key concept to help optimize composability and reuse is to ensure the discovery of useful conceptual models, patterns, or supporting components. If the components we are thumbing through aren't described in a manner the reveals its purpose then there is reason to be concerned. Completing the bridge from concept to implementation will be an arduous task.

This is where the concept of metadata comes into play. Metadata is data about data. It labels and describe what something is. Metadata is formally defined as follows:

Metadata is “structured, encoded data that describe characteristics of information-bearing entities to aid in the identification, discovery, assessment, and management of the described entities” (Gustavson 2003).

We want and need to use metadata to catalog patterns, interfaces and components.

3 PUTTING IT ALL TOGETHER

Figure 7 provides a graphical summarization of the key concepts we have identified.

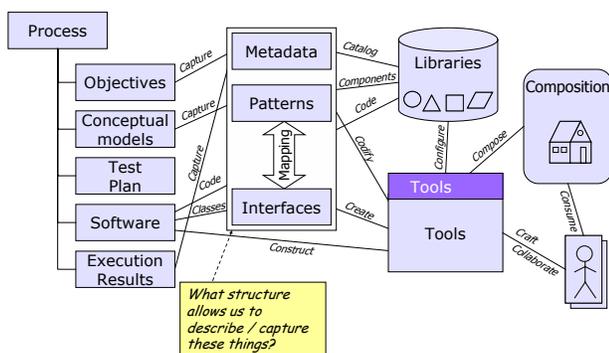


Figure 7: Summary of key concepts.

The presentation form of this graphic, which in a formal setting includes animation, helps in the story telling. It starts off depicting what is common today. Software produced from our process is supported by tools and is often maintained by libraries. Users of such tools build software

assets and access and manage software assets via libraries. These tools also help us leverage the various software assets to compose new applications / capabilities.

What is often lacking is the metadata and conceptual models, which provide a means to catalog and describe the anticipated behavior which is behind such software or simulation assets. Interfaces are also needed to properly reuse and integrate such software and simulation assets (i.e., components). The ability to map between our conceptual models and the various interfaces provides a means to carry forward our conceptual model in our software thereby increasing the likelihood of it being reused to support composability.

The combination of the metadata, patterns, interfaces, and how the interfaces and patterns elements map helps to fulfill the core desire we asked earlier:

What if [reusable] bridges could be defined structurally as means to convey a concept that can be mapped to one or more potential implementations?

We now have identified a framework to support this idea. The question now is simply the following:

What common structure allows us to represent well understood, reusable assets?

In order to answer this question, it is important to understand what the characteristics are of this desired common structure so we know what we are looking for. Visibly we can see in Figure 6 that we need the following:

- discovery metadata,
- patterns,
- mappings of entity and events used for a pattern to
- interfaces that describe the specific class structures of what will be modeled, and shared.

But collectively what does this all entail?

Well, Christopher Alexander, who fathered the concept of patterns even before software and simulations were even an item of interest, expressed the following ideas pertaining to desired characteristics. He shares, and we paraphrase, that a pattern should support the following characteristics:

- Identify and name the common problems in a field of interest.
- Describe the key characteristics of effective solutions for meeting some stated goal.
- Help the designer move from problem to problem in a logical way.

- Allow for many different paths through the design process.

These characteristics need to be considered when identifying a common structure to represent well understood and reusable assets; assets which are intended to be used as means to formulate reusable and composable bridges, which expedite the development process.

4 CHOOSING A COMMON STRUCTURE – THE BOM

One standard that matches well with Alexander’s desired characteristics of a pattern is the Base Object Model (BOM) standard. The BOM is a recent Simulation Interoperability Standards Organization (SISO) Standard developed in the open community for the purpose of supporting composable and interoperable object modeling. It is defined as “a piece part of a conceptual model, simulation object model, or federation object model, which can be used as a building block in the development and/or extension of a simulation or federation” (SISO 2006).

The idea behind BOMs actually can be traced back to the mid 90s when HLA was first being cultivated. It was then that this notion of a piece part concept was considered which could serve as building blocks in respect to the development process and the creation of interoperable object models (DMSO 1996).

The conceptual model aspect is one of the discriminators of the BOM; one of the things that sets itself apart. Prior to the BOM standard, the M&S community did not have a formal and easy way to describe and share conceptual model elements, and did not have an easy way to carry that conceptual model forward through the development process.

Figure 8 peers under the hood of what the BOM standard provides. The subsections that follow dive further into the BOM structure elements.

4.1 Model Identification

The first and foremost piece identified in Figure 8 is the Model Identification, which represents the essential Discovery Metadata. Metadata is important so that BOMs can be described, discovered, and properly reused.

The important thing to share about BOM metadata that it offers not only a way to tag and “label” models, and identify one or more POCs, but a way to collect and share feedback usage through a Use History component. Consider how one views books on Amazon before a book is purchased, and the ability for that prospective buyer to read other reviews – to garner the feedback of other readers / users. That’s just one capability offered through this metadata piece. The Discovery Metadata provided by the BOM

is based on other standards, such as the DDMS, Dublin Core, VV&A Recommended Practice guide (RPG), and HLA, resulting in a well structured and clean means to catalog BOMs.

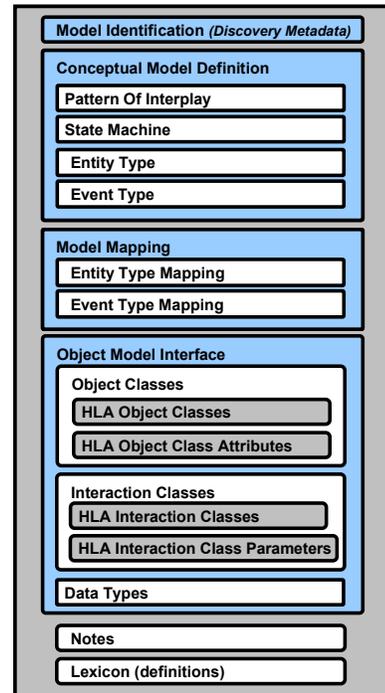


Figure 8: The BOM structure.

4.2 Conceptual Model Definition

The BOM also offers a formal way to capture and share the Conceptual Model. A Conceptual model provides a description of “what is to be represented, the assumptions limiting those representations, and other capabilities needed to satisfy the user’s requirements” (IEEE). In regards to the conceptual model what can be reflected is the Pattern of Interplay, the States of an entity, the entity types and event types.

This idea of pattern discovery is very relevant. A Pattern is “an idea that has been useful in one practical context and will probably be useful in others” (Martin Fowler). The Weapon’s Effect pattern shown in Figure 5 is an example of something is done with some frequency in combat simulations; it is a pattern. This is again is the differentiator from other object modeling frameworks. And this aspect is important, because if intent can be understood as well as the anticipated behavior, then it is easier to know how to reuse something. The conceptual model forms the basis of defining a reusable bridge component.

4.3 Object Model Interface

There is also the aspect of model mapping, which will be touched on in a moment. But first it's important to examine the Object Model Interface. In Figure 8, the first thing that may be seen in regards to the Object Model Interface is an HLA label tethered to Object Classes, Interaction Classes and Data Types. Rightly or wrongly there is often a negative or positive reaction to the HLA label. But it's important to not be fooled by the HLA label. BOMs are not restricted to HLA. There is a perfectly good explanation of why this is here.

It's important to first explain what aspects are not HLA about the Object Model Interface of the BOM. Notice what is not identified are HLA Dimensions, HLA Time, HLA Tags, HLA Synchronizations, HLA Transportations, HLA Switches – they are not in there because they were not seen as essential to document a BASE object model.

All that is really needed at the object modeling level is a way to describe data structures – specifically data types, object classes and the types of interactions that stakeholders need represented. HLA simply provided the most accepted and understood class structure mechanism for describing data types, object classes and interaction classes and that's why it is reflected by the BOM. The development group behind this standard didn't want to re-invent something that was already sufficient for M&S developers.

4.4 Model Mapping

It is important go back to the Model Mapping aspect. This is one area where some of the magic happens. The focus here is that the ABSTRACT things described in a Conceptual Model (entities and events) can be mapped to the actual types of things to be modeled and represented by a system implementation. These models are described in the Object Model Definition. Thus, if a firing entity is identified at the conceptual level (in the conceptual model), a Model Mapping indicates what object classes (or interaction classes) will fulfill the entities and events associated to it.

Incidentally it needs to be clearly understood that a BOM does not require within itself both Conceptual Model Definitions and Object Model Interfaces. Object Model interfaces can live /reside in other BOMs (or FOMs). In other words a Mapping can be made across one or more BOMs, FOMs or other architectures models (such as TENA) defining classes. This loose coupling capability is very important for bringing to bare composable bridges.

5 BOM USE CASE EXAMPLES

To date BOMs have been formulated and used to document and communicate the conceptual space for the

Army, Navy, Air Force, Missile Defense Agency, and general simulation community. For example, JHU/APL used BOMs to represent a synergistic conceptual model of the Airborne Electronic Attack (AEA) communications architecture for the Air Force. Such BOMs were developed from the collection of DoDAF views that were originally formulated by the JHU/APL architecture team. The BOMs have helped to solidify mission objectives and capabilities. Additionally, a mapping of the AEA conceptual space provided by such BOMs is being made using to the software constructs representing JHU/APL's simulation environment. This allows for effective communication and traceability in the composition of AEA models.

BOMs have also recently been used by the surface Navy to rapidly prototype and explore potential Mid-Range Ballistic Attack Munitions (MR-BAM) concepts. These BOMs provided the framework for a resulting prototype software model and simulation that was developed and demonstrated within a very short period of time.

A set of BOMs, known as the Real-time Platform Reference (RPR) BOMs, have been also been developed for the general simulation community. These BOMs define building block components of what had been historically a monolithic model set called the Real-time Platform Reference (RPR) FOM. By breaking the RPR FOM into a set of manageable RPR BOMs, it is now much easier to customize and extend specific capability in respect to both the simulations and the FOMs that such simulations use with requiring significant rework and testing. This facet is explored further in Section 6.3.

6 THE PURSUIT OF INTEROPERABILITY

According to the DoD M&S Master Plan, composability is necessary to enable effective *integration*, *interoperability* and *reuse*. We have already talked about *integration* provided through mapping and *reuse* supported through metadata, but it's time to complete the thought and discuss *interoperability*. Figure 9, illustrates two aspects of composability: model composability and system composability. Thus far we have focused our attention on Model Composability. Taking an idea from the Conceptual Space and reaching a successful implementation. However, within a world in which simulations must interoperate, there is another facet of composability identified as System Composability which correlates with the idea of Interoperability.

It's simply not enough to claim victory once the implementation is complete, we must also explore how such an implementation can integrate with other implementations

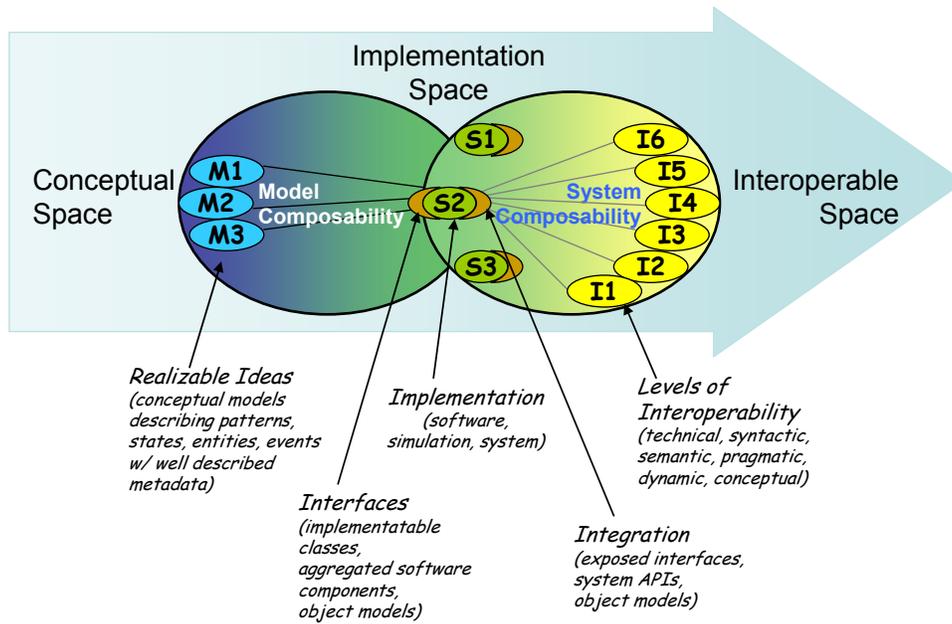


Figure 9: The BOM structure

6.1 Levels of Interoperability

According to Tolk, there are six levels of interoperability (Tolk and Muguira 2003) that need to be explored and pursued to achieve the System Composability capability desired. These levels of interoperability are identified in Figure 10.

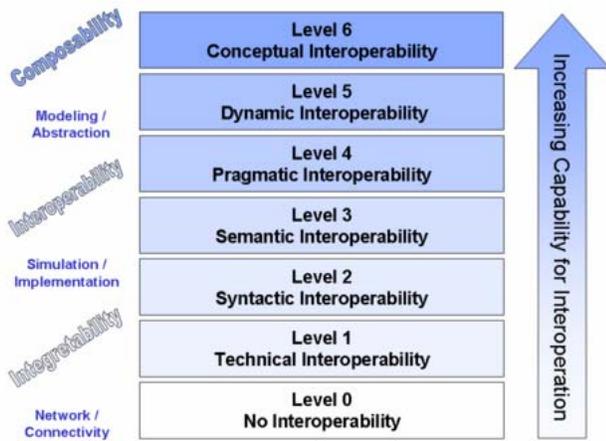


Figure 10: Tolk's levels of conceptual interoperability model (LCIM).

It's important to understand what each of these levels of interoperability entail:

- **Level 1: Technical Interoperability** requires an agreed upon communication technology infrastructure and protocols such as UDP or TCP/IP to

support the handshaking among networked systems.

- **Level 2: Syntactic Interoperability** is achieved using technology such as XML, which offers a means to define and use a common data structure among the systems established in a network.
- **Level 3: Semantic Interoperability** is achieved when a common reference model (i.e., definition set) is used to perpetuate the understanding of the level 2 data being shared.
- **Level 4: Pragmatic Interoperability** is achieved when the systems, simulations or applications involved in the exchange of data are aware of the specific methods and/or procedures that a calling system is requesting.
- **Level 5: Dynamic Interoperability** is achieved when systems are able to come "on-line" and begin to exchange and reflect data with other systems. Such systems are "able to comprehend the state changes that occur in the assumptions and constraints that each is making over time, and they are able to take advantage of those changes" (NATO 2002).
- **Level 6: Conceptual Interoperability** is achieved when the anticipated capability that is to be provided by the models and simulations to be used are fully understood and agreed upon by all the stakeholders. At this level of interoperability there is no ambiguity in what is expected to be shared.

We could spend significant time further discussing each of these levels of interoperability, and the standards

the are available to support each level, but the ability to achieve Level 6 Conceptual Interoperability is what ensures the likelihood of success at any of the other lower levels of interoperability. And, according to Davis what is required for Level 6 interoperability is a “fully specified, but implementation independent model” (Davis and Anderson 2003). This is where the recent BOM standard can be applied.

6.2 The Role of Conceptual Models and BOMs

BOMs can be used to represent “piece parts of a conceptual model that can be used as a building block in the development and/or extension of a simulation or federation” (SISO 2006). It provides a candidate standard that can help achieve the interoperability desired from Level 6 down to Level 2 by helping focus on:

- what needs to be shared conceptually within an M&S environment,
- how the intended models are to perform pragmatically,
- how qualifying interfaces, which map with the conceptual space, are semantically defined, and
- how such models are syntactically structured (i.e., it provides a template).

That said, it should be noted that BOMs are not intended to be a replacement of interoperability standards like HLA. On the contrary, they are instead intended to complement and facilitate the use of such interoperability standards in an independent way.

6.3 Common Use of Object Model Interfaces

Interoperability standards such as HLA and TENA, while serving different domains, share some interesting characteristics. Principally the use of Object Models is shared by the HLA and TENA communities. Object models offer semantic interoperability, and BOMs, however, provide a common object modeling mechanism that can be used across different architectures such as HLA, TENA, and DIS (Cutts, Gustavson, and Ashe 2006).

The piece part and building block concept provided by the BOM standard offers the modularity capability that is sought for interoperability standards such as HLA and TENA. Additionally the BOM standard can be applied to support object modeling of other architectures.

A key word to be emphasized is the word “Base” in Base Object Model. It’s important to understand what is meant by “Base”. A BOM serves as a base in several different ways:

1. It serves as an interface for “Base-level” components that can be constituted with other base-level components. BOMs offer foundational pieces that can be leveraged as a basis for object modeling. Like selecting components off a palette, BOMs can be selected to construct object models of simulations and federations. Thus the idea of a building block. In this way it offers a flexible component approach.
2. It also offers the “basic” elements needed for object modeling. While there are some roots and semantics borrowed from the HLA, what has been stripped away are things that would have restricted BOMs to just HLA implementations. This is very important from the perspective of Syntactic Interoperability, and this will be explored later.
3. Close examination of Figure 5 reveals a weapons effect pattern that can be captured as a BOM. In this pattern example one entity fires a munition on a target. The munition detonates, and an update regarding the damage state of the target is reflected. This is a commonly anticipated behavior for most theater warfare exercises. We expect to shoot at things – and this is how we typically do it. Therefore “base” in this context refers to “fundamental patterns of interplay.” Such patterns provide the basis for fulfilling the overall objectives. The aggregate of these objectives, is what is seen on the right hand side of Figure 11. Each BOM provides a “basis” of understanding at the conceptual model level, describing the fundamental behaviors and models that we can compose into providing a much richer model set.

6.4 Supporting Different Interoperability Architectures

As BOMs are stitched together it results in something called a BOM Assembly. The combination of BOMs spanning both conceptual model and the structural elements offered by object model can be selected, connected, and coupled together to formulate a BOM Assembly. Through the use of some transformations that assembly can be used to represent an HLA Object Model or a TENA LROM as illustrated in Figure 11.

The benefit of this type of mapping using BOMs was shared by Cutts and Gustavson at the I/ITSEC 2006 conference in Orlando, Florida:

“the abstract things described in a Conceptual Model (entities and events) can be mapped to the actual types of things we are modeling, which are described in the Object Model Definition of a BOM. So, if I identify that there is a firing entity at the conceptual level (in the conceptual model), my

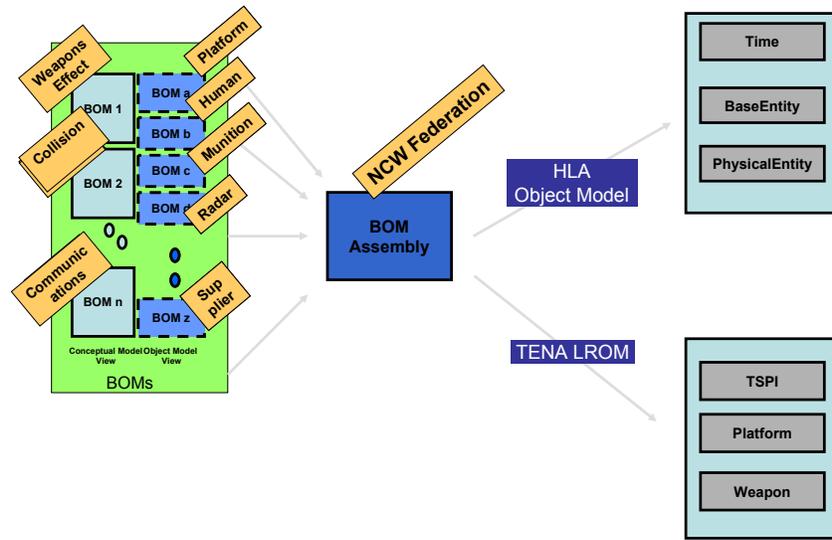


Figure 11: BOM assembly applied to different interoperability architectures.

mapping tells me what system architecture classes [HLA, TENA, Navy OA or otherwise] can fulfill the entities and events associated to it” (Cutts, Gustavson, and Ashe 2006).

In this way, the mapping aspect of a BOM provides a powerful construct for building composable bridges, by spanning the conceptual space with the implementation space.

7 GUIDANCE

So how does one begin to build and use highly reusable assets that help bridge the conceptual space with the implementation space? Again it all starts with the conceptual model, which needs to be carried forward into the other products that are built, such as software. And conceptual models and software need to be properly described with metadata, and mapped so that appropriate building blocks and supporting software implementations (e.g., components) can be identified and used. This is best accomplished with iterative / incremental approach. Also known as a spiral model. Build a little. Test a little. Learn a lot. Go back and add. Share experiences via the metadata. That is what is meant by this approach.

And as patterns are being discovered and described “Consider what should be variable in your design” and “encapsulate the concept that varies” within the pattern (Gamma et al. 1995, p. 29).

Many software developers learn the power of class inheritance, and some begin to over use and abuse this extensible methodology supported by object oriented languages. However, in regards to reuse, inheritance can be a highly limited aspect. We recommend instead to “favor object composition over class inheritance” especially in respect to conceptual modeling (Gamma et al. 1995, p. 20). It is far

more effective in regards to reuse to define a class that “has a” an attribute of another class than to define a class that “is a” an extension of another class. The “is a” relationship provides a hard dependency and binding on another class which can limit the class in being affectively used by others. Whereas, the “has a” relationship allows a class to couple it self with other classes in a very loose and flexible way. The attributes of that class which associate to another class, can adapt to other classes being used with out affecting the class for which the attribute is associated to.

Within a BOM such classes are defined at the conceptual model as entities. And attributes are defined as characteristics. Furthermore, a BOM does permit inheritance at the Conceptual Model Definition layer. It does, however allow for inheritance of classes that are being defined within the Object Model interface layer, which may yield opportunities for appropriate use of inheritance. But at the conceptual model definition layer, it is neither recommended nor feasible.

Another very important aspect is that a BOM (or conceptual model for that matter) should always be designed to an interface rather an implementation (Gamma et al. 1995, p. 18). It’s important to ensure separation of interface from the implementation. This mirrors the Model Driven Architecture (MDA) concept of Platform Independent Models (PIM), which are provided by the BOM, and with Platform Specific Models (PSM), which, within BOM speak, are identified as BOM Component Implementations (BCIs) and defined for a particular platform or language. Having the ability to have BOMs that characterize capability without regard to platform and language, and an available set of BCIs (i.e., components) that fulfill the capability for my platform and language of choice is desirable. It provides the fuel needed to bring conceptual ideas to life, and in a composable way.

8 SUMMARY

In this paper we defined a bridge as “a means to span and provide a way to connect an idea (i.e., concept) to something implementable.” We have identified that such a bridge can be and should be represented and supported by a well-defined conceptual model. That such a conceptual model should act as a bridge, providing an effective way to communicate among stake holders.

We suggested that such bridges could be defined structurally as a means to convey a concept describing common patterns, which can then be mapped to one or more potential implementations. We suggested that such bridges could be built for reuse. We then explored the aspects of building composable bridges linking the conceptual space and the implementation space. The goal of such bridges is to help bring to life satisfying interoperable systems, simulations and applications quickly and easily.

As an analogy we explored the art of composing Lego® creations in how it relates with our desires within the M&S domain. We have stated that the difference between building a Lego® creation and an M&S creation is the complexity of what is intended, and have recognized that the clarity provided by a conceptual model is what helps bring a concept to implementation to a potential state of interoperability. We concluded that a conceptual model provides an effective bridge that could be easily reused to support multiple projects and interoperability efforts.

As an enabling technology, we explored how the BOM, which is a recent SISO standard, offers a means to define and share composable bridges. That it offers a component-based standard for reflecting conceptual models and linking such conceptual models to implementable interfaces. Interfaces that can be supported by a variety of architectures including various software languages (C++, Java) and interoperability standards (HLA, TENA, DIS).

In conclusion we recommend that a standard such as BOMs be used and applied as a common framework for defining and sharing reusable bridges that can be composed with other bridges thereby serving as a building block, which helps facilitate communication among stakeholders and help realize implementation needs.

REFERENCES

- Cutts, D., P. Gustavson, and J. Ashe. 2006. LVC interoperability via application of the base object model (BOM). I/ITSEC.
- Davis, P. K., and R. H. Anderson. 2003. Improving the composability of Department of Defense models and simulations. RAND Corporation. Available via <http://www.rand.org/publications/MG/MG101/> [accessed July 1, 2006].
- Dictionary.com, *bridge*. *The American Heritage® Dictionary of the English Language*, 4th ed. Houghton Mifflin Company, 2004. Available via <http://dictionary.reference.com/browse/bridge> [accessed June 19, 2007].
- DMSO. 1996. The high level architecture (HLA) object model template specification, Version 1.1.
- Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. Design patterns: elements of reusable object-oriented software. Addison Wesley.
- Gustavson, P. 2003. Capturing intent-of-use for the conceptual model – a key to component reuse. 03F-SIW-080, Fall SIW.
- IEEE. The HLA federation development and execution process (FEDEP), 1516.4.
- NATO Research and Technology Organization. 2002. The NATO code of best practice for command and control assessment, revision 2002. Available via the Command and Control Research Program, NATO.
- SISO. 2006. BOM template specification. SISO-STD-003-2006.
- Tolk, A., and J. A. Muguira. 2003. The levels of conceptual interoperability model (LCIM). Simulation Interoperability Workshop (SIW), SISO.

AUTHOR BIOGRAPHIES

PAUL GUSTAVSON is a co-founder and Chief Technology Officer of SimVentions, Inc. www.simventions.com and is focused on the development and integration of technology for creating innovative and engaging experiences and solutions. Paul is a graduate of Old Dominion University, with a B.S. in Computer Engineering (1989), and has supported a wide variety of modeling and simulation, system engineering, web technology, and mobile computing efforts within the DoD and software development communities. He is a principal author of “C++ Builder 6 Developer’s Guide”; and contributor to other books and articles; and, has presented at numerous conferences. He is also a long-time advocate and pioneer of the Base Object Model (BOM) concept for enabling simulation composability, interoperability, and reuse.

TRAM CHASE is a software engineer at SimVentions, Inc. www.simventions.com and is focused on the development and integration of technology for creating innovative and engaging experiences and solutions. In support of BOMs, Tram has been the lead developer of BOMworks™, a tool used to build, edit and compose BOMs. Tram is a graduate of Virginia Tech, with a B.S. in Mathematics (1994), and has supported a wide variety of modeling and simulation and system engineering efforts within the DoD.