BUILDING COMPOSABLE BRIDGES BETWEEN THE CONCEPTUAL SPACE AND THE IMPLEMENTATION SPACE

Paul Gustavson
Tram Chase
Matt Wilson

SimVentions, Inc
11905 Bowman Drive, Suite 502
Fredericksburg, VA 22408, USA

ABSTRACT

Often the process and effort in building interoperable Command and Control (C2) systems and simulations 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 include 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 a 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 for C2 systems and Simulation applications.

1 INTRODUCTION

Whether we are architects, developers, analysts, educators, or managers, composability is a common shared desire. There is a consistent need to assemble capabilities and develop meaningful functionality from the knowledge, tools, standards and components that are available to us.

Figure 1. An Illustration of Composability

Often this 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 developing and integrating models, software applications, simulations, and C2 systems.

Composability is defined by the DoD M&S Master Plan as "the ability to rapidly select and assemble components to construct meaningful 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 systems, 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

Although the construction of a boat or car using Lego® bricks may be more trivial than perhaps the composability of C2 systems and simulations, what is congruous is simply the idea and desire to select and use components in building up C2 systems and simulations.

1.2 The Construction of Meaningful Applications

Composability begins 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 concept mentally, until, at last, a meaningful physical creation is complete. This is where the chasm is crossed from the original concept to the first implementation; when an idea has finally become something real and tangible. The question that is asked though is, "Does it satisfy what was intended?"

1.3 The Satisfaction of Specific User Requirements

Once a Lego® composition is complete, a child will typically revel in their creation. Eyeing it as if it were a prize; satisfied in what they have built BUT only if it meets what they desired. In the workplace these desires are better known as user requirements. And they may start as objectives and ideas formulated initially on paper as requirements, then spun and expanded as concept diagrams drawn up on paper, within a tool, or on a white board. And if the passion and drive are there, they 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 isn't it that we often wait to ask these questions until after a project is completed? This may be a telltale sign that that those involved in the project are perhaps not communicating early and often enough regarding what is intended (i.e., the concepts) 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 C2 systems and simulations can be arduous, this need for bridging the conceptual plane to the implementation plane through composability is significant.

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."[1] 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.

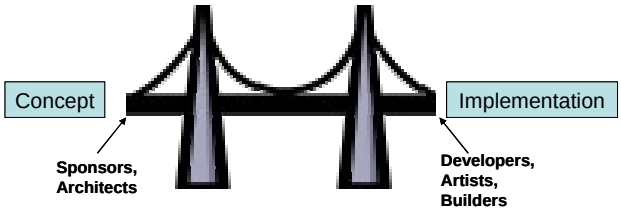

Figure 3. The Development Bridge

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 reflected structurally as means to convey a concept that can be mapped to one or more potential implementations? Like a blueprint to a house or building? 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?

2.1 Why the Conceptual Model is Key

We postulate that the conceptual model provides the basis for a composable bridge. The conceptual model is key because it is intended to describe "what the [system or simulation] will represent, the assumptions limiting those representations, and other capabilities needed to satisfy the user's requirements."[5]

Think about what is intended to be offered by a conceptual model. It is a means to understand what is to be represented. It's not revealing a finished product – but it's a way to get a quick look at what that product might be like; much like a blueprint or house plan is to a prospective buyer. Conceptual models offer a mechanism for communication!

Because a conceptual model is implementation neutral, C2 systems and simulations can share the same (or very similar) conceptual models. Capabilities can be described independent of whether those capabilities are realized as system component or simulation component.

We also recognize that C2 systems including applications and web services may embed simulations to support its objectives. Such C2 system/applications/web services will need to leverage simulations and "compose" services for specific C2/Simulation needs such as training. Thus, the need for composable bridges supported through well-defined conceptual models can be of great assistance here too.

2.2 Fitting the Conceptual Model to your Process

There are many forms that conceptual models take today across the various disciplines where they are needed. They maybe captured in white papers, software prototypes, story boards, or other related artifacts. This said, up until now, there has been little support to realize these prototypes in respect to C2 systems and simulation systems and to experiment and exercise with them at a conceptual / pre-decisional level. There are two potential reasons for this:

1. The tools and standards (including architectures) have either been non-existent or non-intuitive too encourage and support conceptual modeling for the C2 system and simulation domain, or
2. The time required to perform conceptual modeling is simply not allotted by stakeholders within the C2 system and simulation domain.

Both these reasons have merit, but perhaps the most precious resource available to any of is time. And there is a perception that, because of schedule, because of deadlines, because of time, it's often best to move quickly from requirements to development spending little time performing the analysis. For example, If you examine the typical development effort, there are a few interesting tendencies that we see take place.

(1) First Action – if it's a brand new project or even one were something is being modified – a concerted effort is often made to identify the requirements. It might start by asking, "What do we want the system(s) or simulation(s) to do?" Whether those requirements are identified as a bullet list or within a formal specification, the end-result would be some level of understanding of the requirements.

(2) Second Action – once funding is in place and the requirements identified –a team of engineers and developers typically jump in and begin to design, code and configure system(s) and simulation(s) to meet the stated requirements – or at least to an engineers understanding of the requirements.

(3) Third Action – Following the development and integration effort a series of tests might be preformed to see how the system(s) or simulation(s) measure against a set of test criteria – and invariably that test criteria is based on the requirements.

We could spend a great amount of time drilling into any of these three action tendencies, but there's one activity that is routinely missed following the First Action and prior to the Second Action. An activity that, if missed as it often is, can significantly impair the success of the other actions we've identified.

The activity that is often missed is the development of conceptual models. And yet, well defined processes, like the IEEE Federation Development and Execution Process (FEDEP) process partially illustrated in Figure 4, identify the importance of this activity. In Figure 4 it is identified as "conceptual analysis". Conceptual analysis results in the production of conceptual models. These conceptual models provide a supporting plan to fulfill the requirements that were identified in Step 1, and provide a means to guide the rest of the process effort – like a blueprint.

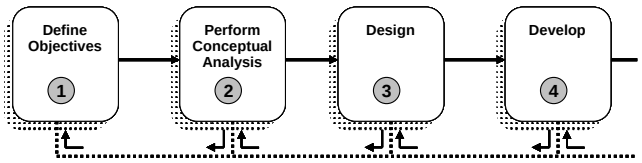

Figure 4. Common Development Process

Interestingly enough this step is the most common development issue for a development team [2]. Specifically, most development and engineering teams completely miss 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.

It's like a house being built solely based on the list of features a home owner wants. The detailed house plan, aka the blueprint, is forgone. Imagine the difficulty a team of carpenters might have in trying to frame and build the house. Imagine the future home owner feeling as if he's in the dark until the structure is revealed to him. Imagine the dialog that might take place between the home owner and the contractor's team when the home under construction does not match exactly what the home owner wanted. And what does the home owner have as a record to prove to anyone that the contractor is not implementing what he originally desired. That is why a blueprint is key.

Like the development of a blueprint, Step 2's goal is to produce conceptual models. Conceptual models identify what needs to be represented, and how things are supposed to behave. And like a blueprint, 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 proper composability, and therefore better software, systems and simulations.

Additionally, conceptual models should be leveraged throughout development. Like a blueprint, we need to keep coming back to it, because it ties what we intend to

build (i.e., our 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?

2.3 Discovering Patterns

A question to ask is, "what should we look for when trying to identify and define conceptual models?" This is where patterns come into play. Patterns offer a solution that can be reused for supporting a common problem or need. They can be a key aspect of our conceptual model. Martin Fowler, a noted Computer Scientist, describes patterns as "an idea that has been useful in one practical context and will probably be useful in others."

Christopher Alexander, a noted author and professor, 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."***

His work is a reflection of what some of the quality architects do; they look for patterns when designing new projects. Specifically they examine the requirements that have been identified and then draw from a familiar set of patterns that they're experienced with and propose a concept that satisfies the objectives. Consider that a blueprint to a future house reflects many common patterns and yet, as an aggregate, it is a very unique design.

The software engineering realm has also begun to embrace Alexander's pattern concept. Software engineering patterns are being applied to support analysis, design and aid in refactoring software. Likewise, we have the same opportunity for applying patterns within the C2 systems and simulations arena. By focusing on reusable patterns, we can achieve a higher return on investment allowing systems and simulations to be more easily developed, integrated and used.

There are two fundamental types of patterns that can be applied: structural patterns, and behavioral patterns.

- A structural pattern is something that reflects a common object or combined set of elements revealed within an environment. We might identify a structural pattern as an abstract entity like an aircraft, ground vehicle, radar or environmental feature such as a bridge and with each one, identify common attributes among these abstract entities.
- A behavioral pattern, on the other hand, is something, that reflects the anticipated actions and variations that may occur, or, perhaps common events or various states of an entity, which can be recognized and reproduced. Some common behavioral patterns that are employed within military scenarios are depicted in Figures 5 and 6.

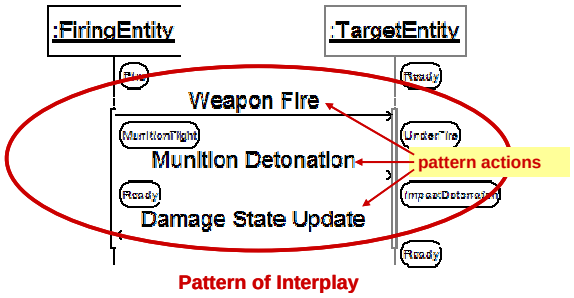

Figure 5. Weapon's Effect

Typically patterns are discovered rather than invented. In this example, we unveil a common pattern that has been reused with great frequency in the simulation 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 an executable model.

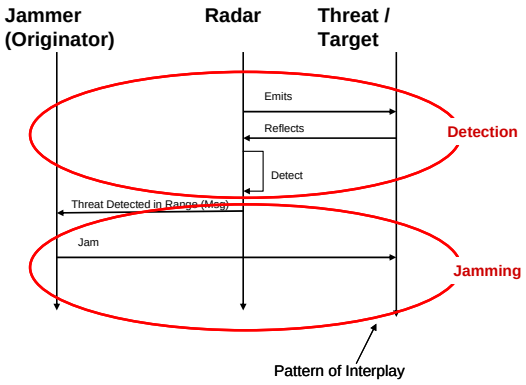

Figure 6. Jamming / Detection Patterns

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 some simple analysis on the problem space and work towards identifying a set of conceptual models. Otherwise, rather than two patterns being revealed, we would have walked away with a single pattern which was fairly bulky, specialized, with limited reuse.

2.4 Identifying Interfaces

In achieving composability, it is not enough to discover and document patterns. Step 3 of the FEDEP process , 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 be a piece of hardware, software, or a service.

Within the software engineering field, and supported within the C2 system and simulation arena, 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, C2 systems) that support what's described by the metadata and defined by the interface.

2.5 Applying Components

Once a desired interface is known, the logical progression is to look for available C2 system and simulation 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, described previously, referred to components. Components in the C2 system and simulation 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 C2 system and simulation composability as being as simple as Lego® construction. C2 system and simulation components don't reflect that intuitiveness that Lego® bricks inherently have. But what Lego® bricks and C2 system and simulation components do share in common is that the inputs and output of a component should be known; that is its interface should be exposed. This allows us to understand the building block functionality a component provides in potentially fulfilling a concept or objective. In this way a component provides a means to satisfy a composable bridge.

2.6 Leveraging Metadata

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

This is where metadata comes into play. Metadata is data that describes other 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 [2]."***

In order to leverage a toolbox of components that we can compose, it is necessary to use metadata to catalog (i.e. properly label) the conceptual model (patterns) and interfaces that reflect these available components.

3 PUTTING IT ALL TOGETHER

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

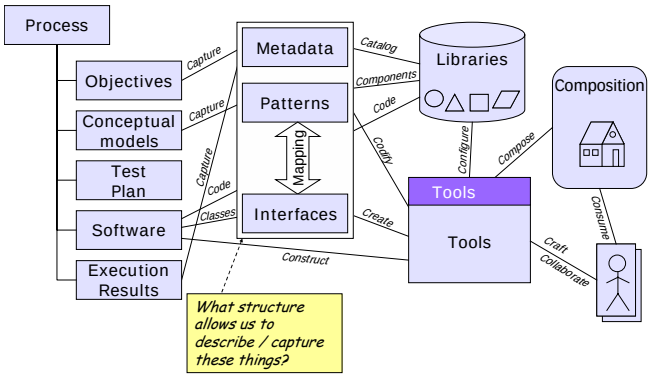

Figure 7. Summary of Key Concepts

To understand this graphic, consider the following narrative. In most simulation environments today, software produced from our process is supported by tools and is often maintained in 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 thumbnail sketch to catalog and describe the anticipated behavior which is behind such software, system or simulation assets. Interfaces are also needed to properly reuse and integrate such software, system and simulation assets (i.e., components). The ability to map between conceptual models and the various interfaces provides a means to carry forward our conceptual model in software thereby increasing the likelihood of it being reused to support composability.

This thumbnail of capability, composed of the combination of metadata, patterns, interfaces, and their mappings helps to fulfill the core objective we discussed earlier of defining and using composable bridges. Bridges that produce a blueprint that satisfies the identified requirements prior to implementation, and satisfies a plan needed for implementation. The question now is simply the following:

What common structure allows us to represent composable bridges for supporting C2 system and simulation interoperability?

Figure 7 identified a few key characteristics that we need, which can help answer this question, they include 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, system 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."[3]

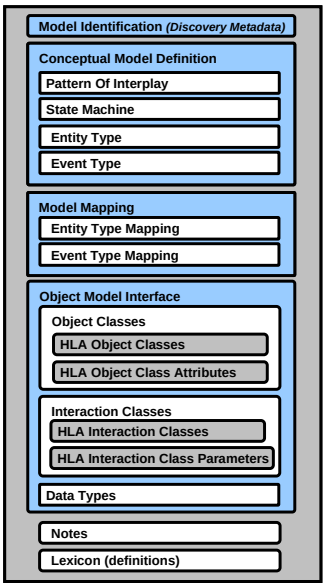

Figure 8.
The BOM Structure

The idea behind BOMs can be traced back to the mid 90s when the HLA interoperability standard 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 simulation models and system components.[4]

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. Consider items on the shelf of a grocery story. If a label wasn't on the cans of food, and boxes of cereal that described the contents, then that food resource would likely not be bought.

The important thing to share about BOM metadata that it offers not only a way to tag and "label" resources with vital data such as Points of Contact, Description, and Application Domain, but it also provides a way to collect and share feedback usage through a Use History element. Consider how one views potential books and products of interest on Amazon.com. A prospective buyer is offered the ability to read about the experiences of other readers pertaining to the book or product. That facet of metadata is also offered through the model identification metadata piece provided by the BOM. It should be noted that this metadata 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.

4.2 Conceptual Model Definition

The BOM also offers a formal way to capture and share the Conceptual Model. As stated earlier, a conceptual model provides a description of "what is to represented, the assumptions limiting those representations, and other capabilities needed to satisfy the user's requirements."[5] 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, 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 (i.e. , a C2 system or simulation component). The capabilities of these components 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 be described within other BOMs. Yes, mapping can be made across one or more BOMs, FOMs or other architectures models (such as TENA) defining

classes. This loose coupling capability is vary important for bringing to bare composable bridges for it allows relationships to be defined that can be easily replaced.

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 C2 system component or simulation was allowing it to be 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* especially as it relates to C2 systems and simulations. Figure 9, illustrates two aspects of composability: model composability and system composability. Thus far we have focused our attention on model composability, which involves taking an idea from the conceptual space and reaching a successful implementation. However, within a world in which C2 systems and simulations must interoperate, there is another facet of composability identified as system composability which correlates with the idea of Interoperability.

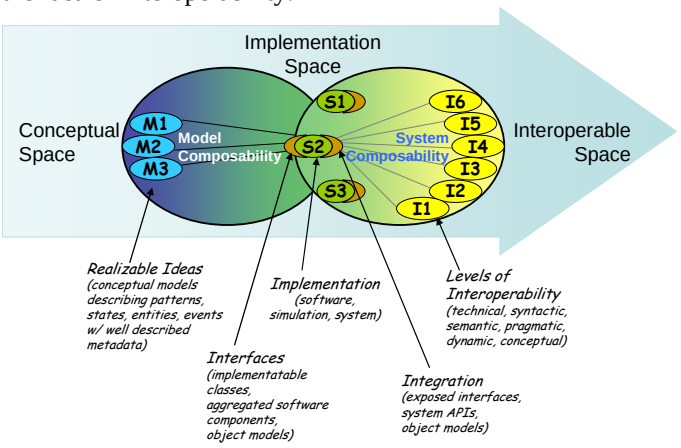

Figure 9. The BOM Structure

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.

6.1 Levels of Interoperability

According to Tolk, there are six levels of interoperability [6] that need to be explored and pursued to achieve the System Composability capability desired. These levels of interoperability are identified in Figure 10.

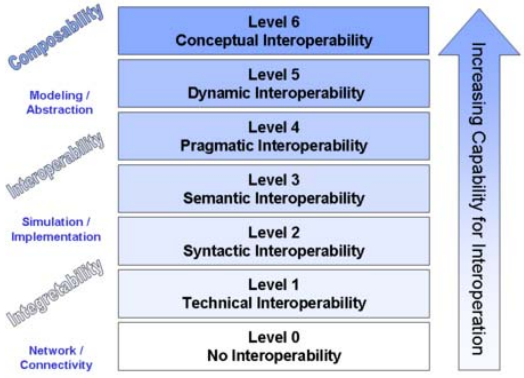

Figure 10. Tolk's Levels of Conceptual Interoperability Model (LCIM)

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."[7]
- **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. According to Davis what is required for Level 6 interoperability is a "fully specified, but implementation independent model."[8] And 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."[9] It provides a candidate standard that can help achieve the interoperability desired from Level 6 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 exchange standards like DIS, HLA or TENA. On the contrary, they are instead intended to complement and facilitate the use of such interoperability standards in an independent way – even allowing different interoperability exchange standards to co-exist – if that is what is necessary.

6.3 Common Use of Object Model Interfaces

Interoperability standards such as DIS, 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 provide a common object modeling mechanism that can be used across different interoperability architectures. [10]

The piece part / building block concept provided by the BOM standard offers the type of modularity that is needed for representing C2 system and simulation object models and facilitating interoperability of C2 system and simulations – regardless of what exchange standards are applied (e.g., DIS, HLA and TENA).

A key word to be emphasized is the word "Base" in Base Object Model. A BOM serves as a "Base" in several 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. In a very simple way object classes can be defined with their supporting attributes. Not required for these classes are the things that are implementation specific such as dimensions and routing spaces, which is used for HLA but may be meaningless for other interoperability architectures that could be employed. This aspect 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 needed of a C2 system or simulation 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 the exchange document for a specific interoperability architectures such as 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 mapping tells me what system architecture classes [HLA, TENA, Navy OA or otherwise] can fulfill the entities and events associated to it." [10]*

In this way, the mapping aspect of a BOM provides a powerful construct for representing composable bridges, because it spans the conceptual space with the implementation space.

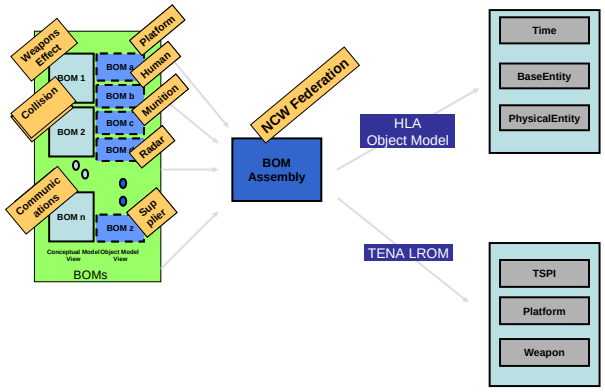

Figure 11. BOM Assembly applied to Different Interoperability Architectures

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.. And conceptual models describing C2 system and simulation components need to be properly described with metadata, and mapped so that appropriate building blocks and supporting C2 system and simulation components can be identified and used. This is best accomplished with iterative / incremental approach. Also known as a spiral model.

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. [11- p. 29]. This abstraction effort is what allows greater reuse and value.

Many developers and engineers 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. What is recommended instead are for engineers to "favor object composition over class inheritance" especially in respect to conceptual modeling [11 - 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 these composable bridges that are build and use (e.g, a BOMs) should always be designed to an interface rather an implementation. [11 - p 18]. It's important to ensure separation of interface from the implementation. Having the ability to have combosable bridges (e.g. BOMs) that characterize capability without regard to platform and language, and an available set of C2 system and simulation components that adhere to those BOMs and 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 address interoperability as it to pertains to C2 systems and simulations. We explored how such interoperability can be addressed through "composability", and suggest that composability can best be understood and carried forward through the use of "bridges" that "span and provide a way to connect an idea (i.e., concept) to something implementable."

We have identified that such bridges can be and should be represented and supported by well-defined conceptual models, providing an effective way to communicate among stakeholders.

We state that C2 systems and simulations can share the same (or very similar) conceptual models. That, because a conceptual model is implementation neutral, capabilities can be described independent of whether those capabilities are realized as system component or simulation component.

We also recognize that C2 systems including applications and web services may embed simulations to support its objectives. Such C2 system/applications/web services will need to leverage simulations and "compose" services for specific C2/Simulation needs such as training. Thus, the need for composable bridges supported through well-defined conceptual models can be of great assistance here too.

We suggested that composable bridges, like a blueprint, need to be reflected structurally as a means to communicate a concept for all stakeholders. We suggested that such bridges could be built for reuse describing common patterns, which can then be mapped to one or more potential implementations. We then explored the aspects of building composable bridges, which link the conceptual space and the implementation space. The goal of such bridges is to help bring to life satisfying interoperable C2 systems and simulations quickly and easily.

As an analogy we explored the art of composing Lego® creations in how it relates with our desires within the C2 and simulation interoperability. We have stated that the difference between building a Lego® creations and an C2 and M&S creations 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 and then 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).

The BOM provides a reflection of that conceptual model, which can be used in several ways:

(1) It helps identify a neutral way to characterize both C2 capabilities and simulation capabilities.
(2) Next, it can be used to help identify existing candidate C2 system and simulation components that support specific purposes such as training and testing.
(3) If eligible C2 system and simulation candidates…
    a. cannot be found, then the BOM helps "bridge" over to designing the capability that's needed.
    b. can be found, then selections can made with "mappings" documented, which will allow for easier reuse / integration in the future.
(4) Additionally, the use history experience can be highlighted back into the BOM (as metadata), making it even easier for sponsors, developers and engineers to select and choose viable C2 and Simulation components in the future that they want to integrate into their environment.

We recommend that a standard such as BOMs be used and applied as a common framework for defining and sharing composable bridges, which facilitates communication among stakeholders and helps realize implementation needs for C2 systems and simulations.

REFERENCES

[1] Dictionary.com, *bridge*. The American Heritage® Dictionary of the English Language, Fourth Edition. Houghton Mifflin Company, 2004. http://dictionary.reference.com/browse/bridge (accessed: June 19, 2007).

[2] Gustavson, *Capturing Intent-of-Use for the Conceptual Model - A Key to Component Reuse*, 03F-SIW-080, Fall SIW, Sept 2003*

[3] SISO, *BOM Template Specification*, SISO-STD-003-2006, SISO, 31 March 2006.

[4] DMSO, *The High Level Architecture (HLA) Object Model Template Specification*, Version 1.1, 1996.

[5] IEEE 1516.4, *The HLA Federation Development and Execution Process (FEDEP)*

[6] Tolk, A. and Muguira, J.A., *The Levels of Conceptual Interoperability Model (LCIM)*, Simulation Interoperability Workshop (SIW), SISO, Fall 2003.

[7] NATO Research and Technology Organization (RTO). (2002). *The NATO Code of Best Practice for Command and Control Assessment*; Revision 2002. Available via the Command and Control Research Program, NATO

[8] Davis, P.K. and Anderson, R.H. (2003). *Improving the Composability of Department of Defense Models and Simulations*. RAND Corporation,

http://www.rand.org/publications/MG/MG101/ [last visited July 2006]
[9] SISO, *BOM Template Specification*, SISO-STD-003-2006, 31 March 2006.
[10] Cutts, Gustavson, Ashe, *LVC Interoperability via Application of the Base Object Model (BOM)*, I/ITSEC 2006
[11] Gamma, Helm, Johnson, Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995

REVISION / PUBLICATION HISTORY

Originally titled "*Building Composable Bridges Between The Conceptual Space And The Implementation Space"* and published within the Proceedings of the 2007 Winter Simulation Conference. Authored by Paul Gustavson (SimVentions), and Tram Chase (SimVentions).

Updated May 2008 to better reflect the support this concept has for composing C2 systems. Additionally author contributes for the 2nd revision were provided by Matt Wilson (SimVentions).

AUTHOR BIOGRAPHIES

PAUL GUSTAVSON is a co-founder and Chief Technology Officer of SimVentions, Inc. (http://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. Paul lives in Virginia with his wife and two boys.

TRAM CHASE is a senior software engineer at SimVentions, Inc. (http://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. Tram lives in Virginia with his wife and two children.

MATT WILSON is a senior software developer and architect at SimVentions. He has over 16 years experience as a project manager and as a software and systems engineer. He has participated on a variety of projects and standards committees including the Object Management Group (OMG) C4I Domain Task Force since 2004. He designs and implements software system architecture for high performance, user interface intensive, distributed, web-based, n-tier, and desktop software applications. Mr. Wilson currently supports various DoD customers in the area of software development, systems engineering, and analysis. His current efforts involve Human Systems Integration (HSI), technical leadership, SBIR project development for advanced visualization, and processes for implementing Open Architecture (OA) combat systems into the US Navy fleet.