2011

Software Reuse for Modeling and Simulation

Emily Andrew

Charles D. Turnitsa
Old Dominion University

Andreas Tolk
Old Dominion University

Repository Citation

Andrew, Emily; Turnitsa, Charles D.; and Tolk, Andreas, "Software Reuse for Modeling and Simulation" (2011). *Modeling, Simulation & Visualization Engineering Faculty Publications*. 50.
https://digitalcommons.odu.edu/msve_fac_pubs/50

Original Publication Citation

Andrew, E., Turnitsa, C., & Tolk, A. (2011). *Software Reuse for Modeling and Simulation* Paper presented at the Spring Simulation Interoperability Workshop 2011, Boston, Massachusetts, April 4-8, 2011.

Software Reuse for Modeling and Simulation

Emily Andrew
Raytheon Company
Network Centric Systems
Marlborough, MA 01752
Emily_B_Andrew@raytheon.com

Charles D. Turnitsa
Virginia Modeling Analysis and Simulation Center
Old Dominion University
Suffolk, VA 23435
cturnits@odu.edu

Andreas Tolk, Ph.D.
Frank Batten College of Engineering & Technology
Old Dominion University
Norfolk, VA 23529
atolk@odu.edu

Keywords:
software engineering, software reuse, repositories, interoperability

ABSTRACT:

In Modeling and Simulation, as a distinct area of software engineering, there is much interest in being able to reuse software components. However, the practice of simulation development and maintenance is different from software engineering because of several factors. In this paper, a brief overview of the foundations of interoperability, and how they apply to the reuse of model based software is explored, as well as examination of current practices to include M&S software repositories. Some recommendations, based on research at the Virginia Modeling Analysis and Simulation Center (VMASC) and practice at the Raytheon Company Network Centric Services, are made.

1 Introduction

The topic of software reuse for Modeling and Simulation is a complex one. This paper, while diffused into several topics, does not begin to address all of the facets related to the idea nor does it fully address any one of them. In fact, each of these ideas has been the topic of much study within the series of Simulation Interoperability Standards Organization (SISO) workshops and will continue to be. However, this is an attempt to bring together some information gleaned from the theoretical and academic and, presented for the practitioner, such that the findings that do exist can be made to serve the community.

2 Foundations of Interoperability

The concept of reuse of either models or software is predicated on the idea that such models or software components are interoperable – that is, the existing element (whether a model, an algorithm, a software component, or an entire simulation system) can be made interoperable with new components that it was not originally designed to work with. Towards this end, a discussion concerning reuse properly begins with a review of what interoperability is, and how it is treated within the M&S community.

Beginning with the idea of making data objects and data bases for simulation systems interoperable, the earliest efforts at composing such is based on the historic work of federated databases. Experts dealing with very large heterogeneous and distributed data in large corporations like Coca Cola or Puma had to solve inconsistencies in databases and workflows on a big scale. The findings and recommendations led us to a multiple-layer-translation model published by Spaccapietra et al. [1] and extended by Parent and Spaccapietra [2]. Gorman’s [3] work contributed in particular to the understanding of military challenges.

Another research domain influenced the work in a similar manner, namely Reynolds et al. [4] and Davis and Bigelow [5] contributions on multi-resolution modeling challenges in distributed simulation systems. The thrust of multi-resolution modeling is to enable to reuse of simulation components that were developed with different resolutions of detail, in new system-of-system architectures with each other. The approach has not been found to have a general solution, yet.

The idea to use a layered approach to deal with the realm of interoperable and composable solutions has been used before. One of the most influential models is the Levels of Information Systems Interoperability developed by the Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance Interoperability Working Group and published in [6]. Winters et al. [7] give an overview how the various layered approaches are related regarding data management issues.

The report of the RAND Corporation on Composability Challenges [8] within the US DoD is an excellent summary of current solutions, and their open questions remain valid.

In the M&S domain, the work of Petty and Weisel lead the way for many researchers, in particular their Lexicon for Composability [9]. They were among the first to identify the need to distinguish between composability and interoperability based on the need identified by Harkrider and Lunceford [10]. Petty's and Weisel's work [9] motivated the first LCIM as presented by Tolk and Muguira [11]. Page et al. [12] refined the model by introducing integratability as the third concept. The LCIM uses a slight modification of Page's definitions. Hofmann [13] used these ideas to formulate has challenges for M&S composability. The LCIM was refined and first presented in its current form by Turnitsa in [14]

In addition to the work in the M&S domain, the re-search regarding semantic web composability was a driving force for the research described here. Welty and Smith [15] summarized the state of the art in their proceedings and identified the necessity to compose web services in a more consistent way in several papers. How this was answered is best summarized in Agarwal et al. [16] and the book of Alesso and Smith [17]. Both are using the notion of semantic web services, in which services are de-scribed in more detail allowing the user (and ultimately other software components) to identify services that can be composed meaningfully. Current work is referenced on the Semantic Web Service Initiative website. One of the most visionary papers, published by Chen et al. [18], presented the use of ontology-based knowledge management in support of composable solutions. Finally, the work de-scribed in this paper was also influenced by the mathematical foundations for Model Theory as described by Pilley [19] and the knowledge representation work of Sowa [20].

Finally, the work of agent mediated solutions in the M&S domain closed the gap between M&S composability and semantic services. To be mentioned in particular is the work of Yilmaz [21] and Yilmaz and Paspuletti [22]. They used agents to capture the behavior and information exchange requirements of M&S components and let them decide how and what to compose, using meta-structures of the semantic web to support this work.

3 M&S Software Engineering

Simply put developing for software reuse involves developing computer software code that can be usable in a variety of different programs or contexts. This is an old idea going back at least to the 1960s when the first portable languages became popular, but many enterprises report difficulty with it. In spite of the reported difficulties, it is something that happens quite frequently in an ad hoc manner.

3.1 Reuse During Development and Application

Before going forward, it is worth pointing out that people generally mean one of two very different ideas when they talk about software reuse. The first of these is reuse of software engineering components - those that will be used in the phases of developing software packages. This includes reusing sections of code, libraries, or algorithms for different projects. The second area of reuse is one that is concerned with the application of software elements that are already developed. This second use is concerned with making use of already developed components for new projects where entire software components (executables) are reused in different environments.

Both of these identified uses (reusing code snippets and reusing whole executables) are quite common in the field of Modeling and Simulation. However, because the Modeling and Simulation community differs somewhat from other software development efforts, there are also differences for the concept of reuse. One of the key ways in which Modeling & Simulation differs from other software development and application communities is that Modeling & Simulation software has to be true to the model that it is representing. Normal software considerations in terms of timing and algorithm efficiency are secondary to the requirement that the resulting software be true to the conceptual model. If we adhere to this requirement, then the model will have some chance for reuse. When we attempt to reuse software that is based on a model, then each new environment we wish to use that software in must be conducive to the environment the model was originally used in. Problems with this alignment between the model and the environment can lead to many disruptions. One way a model's suitability to an environment can be examined is to look at the possible 'touch points' where Modeling & Simulation components might interact (i.e., algorithm integration, data exchange, process initialization, etc.) and yet not violate the model that is being integrated.

3.2 Planning Modeling & Simulation Reuse

Given that we must remain true to the model when we are developing code (libraries, algorithms, classes, etc.) that will be reused, there are several things to keep in mind. First, since the software will be part of a simulation based on a model in some way, be aware of the conceptual model constraints and assumptions and follow those assumptions throughout. For example, if the original conceptual model for a vehicle movement model was that the vehicles would be moving over surfaces that report a friction coefficient, then make that part of the algorithms you use. This in turn drives some specifics about the environment where a movement function like this could be used. An example might be only in environments that report a surface friction coefficient. Second, since your code will be used to represent some part of a synthetic world, make sure that you take care of everything that you need in your section of code and do not touch anything that is not your responsibility! By compartmentalizing a complex virtual world into different software components, each of which is responsible for some portion of that world as either objects or some process, it is easier to replace and reuse new pieces for those components. Thus it's important that each component is sufficiently self -contained so that it can be replaced easily because it is not trying to handle too many things. The third requirement is document, document, and document! Everything about reusing Modeling & Simulation software components is predicated on the ability for the software developer or applications engineer to KNOW what the model is doing. The only way this is possible is for there to be sufficient documentation for both the model and assumptions about the environment the model will operate in.

Some additional points should be made here. First, there should be some understanding as to the points of interaction the component will expose when connecting with other components. This exposure of the points of interaction can be expressed using several methods: simple data Input/output; some simulation federating technology such as High Level Architecture or Distributed Interactive Simulation; or composition/orchestration architecture such as Service Oriented Architecture. In all cases, clear expression of the points of contact should be made available for those who want to reuse the component. Second, construct your software in such a way that the points of interaction can be accessed without being disruptive to your internal structure. And third, ensure the model is followed internally in a consistent way where interactions to data exchange, for example, are limited. Again, documentation is the key towards making knowledge available about the model, contact points, assumptions about the environment, the model's requirements, and so on.

3.3 Reuse is Interoperation

When we talk about reuse of Modeling & Simulation software, whether whole components at the application phase or sub-components at the development phase, the focus is on software that will be required to operate in a new context. This means it will have to interoperate with new components and with a new environment.

The new context software will have other software it needs to interoperate with. Here, care must be taken to understand there is an informal employment of the term interoperability that is based on several theoretical findings regarding the mathematical requirements and philosophical implications of a model existing in an alien environment. Simply, it means the state of simulation software working with other simulation software. As we have seen for Modeling & Simulation software, we are primarily talking about the interoperability of the models and secondarily about how the software will handle this.

To understand what is possible, we will look at the Levels of Conceptual Interoperability Model (LCIM) to determine metrics to attempt to achieve. The current version of the LCIM was first presented by Turnitsa [14], and has been the subject of many topics on composability and interoperability. In the typical diagram of the LCIM (see Figure 1- Levels of Conceptual Interoperability Model), there are indicators for the different layers of conceptual expressability between components. This includes integratability, interoperability, and composability. If it's desired that a Modeling & Simulation software component function with other components and have the ability to express information to one of these three layers, then there are clear requirements that must be followed for each layer.

As mentioned earlier, the three broad distinctions of integratability, interoperability, and composability came from observations on the distinct groupings of levels, made by Page [12]. These distinct groupings (layers) provide the organization for the following introduction to the levels of the model.

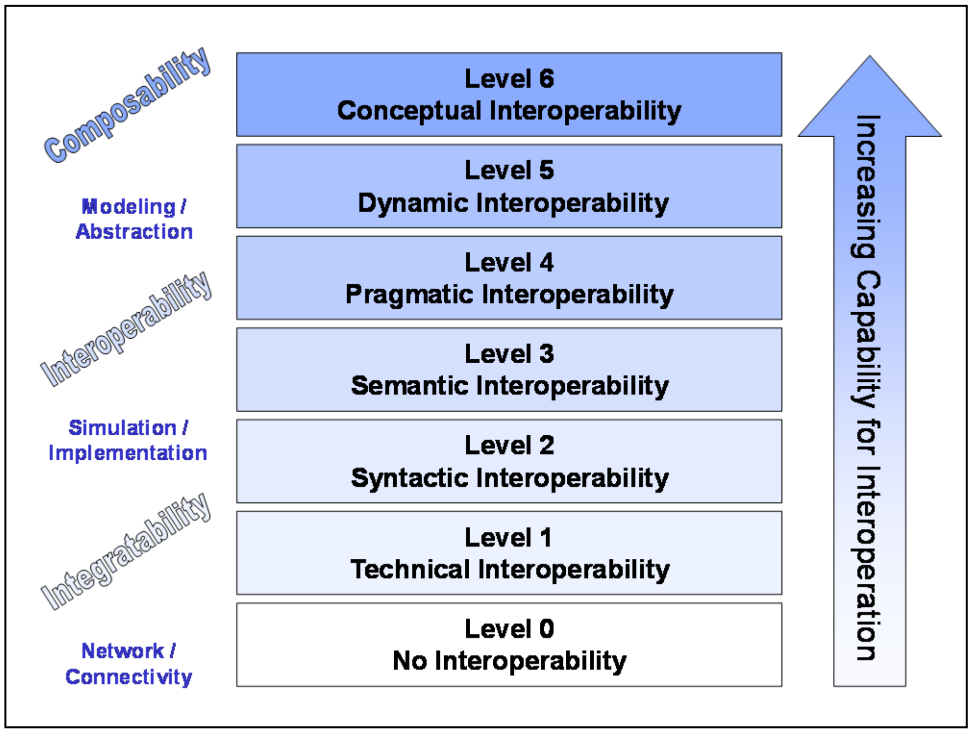
Composability
Modeling / Abstraction
Interoperability
Simulation / Implementation
Integratability
Network / Connectivity
Level 6
Conceptual Interoperability
Level 5
Dynamic Interoperability
Level 4
Pragmatic Interoperability
Level 3
Semantic Interoperability
Level 2
Syntactic Interoperability
Level 1
Technical Interoperability
Level 0
No Interoperability
Increasing Capability for Interoperation

Figure 1- Levels of Conceptual Interoperability Model

First, the lowest bar is the integratability layer. Here, for two or more components to be "integratable," they must be designed in such a way that they can work together. For this layer, the following requirements must be adhered to.

- Technical interoperability (level 1). – This is not really a consideration because, while the systems are interconnected, there is no provision for even data to be exchanged. As such, this level is merely a precursor for other levels and minimally some ability to exchange data over a network or within a shared component framework, must exist.
- Syntactic interoperability (level 2). At this level, data exchanged is within a common syntax. The data should be in a form that can be produced or read by any bit of software that will interact with it. This could be architecture objects such as pipelines, semaphores, signals, Common Object Request Broker Architecture; mark-up data such as Extensive Markup Language and binary objects; or simulation framework interactions such as Run Time Infrastructure (RTI) packets. At this level, the general syntax that the data has must be known by the developer when writing code for this level. Often this can be made modular and be replaced as use warrants.

Next, at the interoperability layer, there are two possible LCIM levels. These include Semantic and Pragmatic which are both concerned with the exchange of information. For our purpose, this is defined as "data in context." For each of these two levels to be satisfied at this layer, the following requirements apply.

- Semantic interoperability (level 3). – For this level, the techniques described for Syntactic interoperability are followed. However, the difference is there is a shared understanding of the labeling used to describe the data being exchanged. In this way, the data has semantic value and transitions from being data to becoming information. Most modern simulation interoperability frameworks, designed for software that is developed independently, support interoperability approximately to this level. At this level also, not only must the common syntax be observed but additionally the common labeling of the data must be agreed to. This could be the native labeling of one of the subcomponents involved but, in the case of components, often takes the form of a hub or interchange model.

- Pragmatic interoperability (level 4). This is a much more difficult situation. At this level, the simulation components or subcomponents have a complex relationship with each other. This is because there is an awareness of the context of the data that will be exchanged such that the components can make "pragmatic" provisions for data that is produced or how it is to be consumed. Without a shared understanding of what the foreign model is, it is unlikely that this is achievable by a software component. The rewards, however, would be a vast increase in the fidelity of the shared environment. For this, there are two possibilities. The first is the developer of a new component is aware of the context of a component the developer's component interoperates with (recall that Pragmatic interoperability requires context awareness). The second reward is there is a way to identify what is important about the context between interoperating systems. This is not as difficult as it may appear if the requirements for context related parameters are kept small such as for spatial location.

Finally at the composability layer, the last two levels of the LCIM are present. At this layer, the goal is component harmonization where assumptions, dynamic timing, and operational constraints are all considered. The requirements for these two levels of the LCIM follow.

- Dynamic Interoperability (level 5). This level is a time sensitive version of Pragmatic Interoperability. Here, the goal is for each interacting component is to not only be aware of the context of the other system but also to be aware of how that context changes over time during the execution of the simulation. This requires knowledge not only of the conceptual model of the other system, in terms of objects and relations, but also intimate knowledge of the processes involved to include timing, specific process effects, and changing states . As with the Pragmatic Level, the developed system must have the ability to become aware of the context that other systems it composes with when data exchanges. Without the system being a functional equivalent of the other, this is only possible if information about that context can be exchanged. While this may be possible in limited situations, as with Pragmatic Level interoperability, the writing of software that is adaptive to changing contexts may be quite difficult without intimate knowledge of the models involved before starting.
- Conceptual Interoperability (level 6). In addition to complete knowledge of the model used by other systems, this level of interoperability requires that conceptualizations of each model involved, to include constraints and concept interpretations, must be in alignment. The requirements of the Pragmatic Level are challenging as there must not be any differences in the conceptualizations of the composed models as constraints and assumptions of each model must be harmonized.

4 Reuse of Existing Components

The reuse of existing Modeling & Simulation components includes the reuse of the component's software in addition to the categorization and storage of the components in a repository. For this situation, the components are not originally designed with reuse in mind but rather are acquired "off the shelf" and modified so they can be reused. To find suitable existing components that might be modified for reuse also requires the design and use of a repository that can support the discovery and selection of such components. The topic of repositories to include this aspect is covered in the next section.

In developing Modeling & Simulation software for reuse, the key points previously discussed to follow include (1) be model aware and model sensitive, (2) be aware of the environment that the developed software will be reused in and what it will interoperate with, and (3) decide what level of interoperability is desired and provide the required software support. But what about existing software that is not designed with reuse in mind? In this situation, the desire is to use these simulation components or sub-components in an environment they were not designed for. What are the issues and remedies that must be considered for this case?

4.1 Evaluating Possible Modeling & Simulation Software

While evaluating possible simulations to be included in your final solution of software components, there are a couple things to keep in mind if you will be reusing existing software components not designed for reuse. This includes: (1) taking stock of the situation, (2) be aware of the model in the candidate software and (3) be aware of the inherent model(s) in the new environment the candidate software will be modified to exist in. Each of these are discussed in the following sections.

4.1.1 Take Stock of Your Situation

First, understand the specifics of your situation. What should be clear by now is that one of the most important things when working with Modeling & Simulation software is to be aware of the model that the software is implementing. Next, be aware of what other models the software is being modified to interact with in the new environment. Finally, be aware of the software itself and whether modifying it is even technically feasible. For example: Is it a binary for which you have no source code? Can it exchange data at key touch points? Will it even function in its new environment?

4.1.2 Be Aware of Your Software's Model

A conceptual model is behind every piece of simulation software. Its purpose is the conceptualization of what the software is simulating. Sometimes this is written down formally such as in Unified Modeling Language diagrams or a conceptual graph. Often it is in the requirements documents and the internal documentation of the software. If the original conceptual model is not available, then the best the developer can do is to deduce the apparent conceptual model based on the simulation software's behavior. While this sort of model deconstruction is often necessary, it is often a very incomplete and unsatisfactory solution! For this reason, when working in a software-development organizations that will practice reuse the key to survival and success is again documentation, documentation, and documentation! Although this is often stated with all software development, in the case of Modeling & Simulation software, it is especially true. Little can be done with software if the underlying model is unknown.

4.1.3 Be Aware of the Differences between Your Environment and the Candidate Software's Model

As you bring your software into a new environment, there will likely be differences between the new environment and your model. The first crucial type of difference, and perhaps deadly, is where different assumptions and constraints exist about the synthetic environment. Your software component may assume certain things about how the simulated world works, yet the new environment you want to reuse it in may make different assumptions. The second crucial type of difference has to do with how objects and processes are handled within your model and how the new environment handles them. These specifics make up what are called the 'dimensions of difference,' and there are three different types: scope, resolution, and structure

4.2 The Dimensions of Difference

While there may be many things that differ between simulation software components and the models that inspire the software's design, there are a couple of interesting observations. When a software component is compared to another component or evaluated to fit into an alien environment, there is often some commonality in the identification of what is needed and a candidate solution. For example, if one is looking for a simulated weather software component, there are likely several candidate solutions that will satisfy the need for this type of software component. However, even given the apparent "sameness" of these candidate solutions, there are differences between each. This occurs as a result of different developer perspectives under which the software was developed such as developer bias, use case requirements, and business rules. These differences, even among components that are supposedly of the same element, are called the dimensions of difference.

4.2.1 Differences in Scope

If you want to reuse simulation software, one of the key things that must be in alignment is the model's scope must be compatible with other software in the new environment. Scope is simply the limits of what the model handles based on what it is intended to represent. An example is if a software component is developed to model weather and its scope includes wind, precipitation, and air pressure, but it is then reused in an environment that expects electromagnetic effects of sunspot activity, then there is a mismatch in scope. Software with a scope that is larger but still inclusive than the scope expected in its new environment it is being reused in, however, may work but be careful.

4.2.2 Differences in Structure

When a model is first conceived of, there are some assumptions made about how the objects and processes contained in it will be structured. If that same model, and the software it is based on, is reused in a new environment and other software in that environment

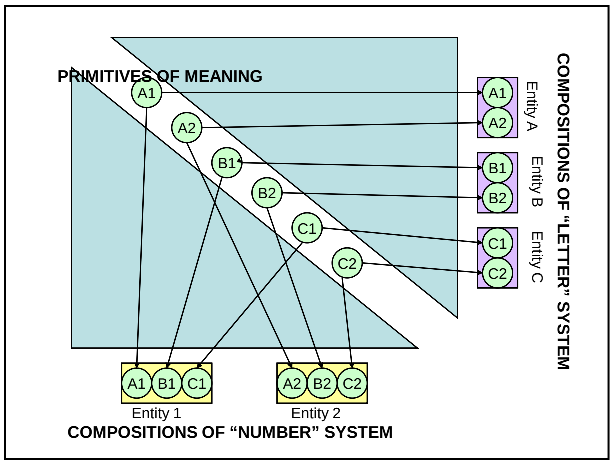

Figure 2- Primitives of Meaning

has a different structure, there may be mismatch.

A theory is in development at VMASC, to capture the techniques whereby the structural elements of objects and processes can be identified and addressed for manipulation, re-definition or isolation. Results have appeared in [29] and [30]. In (Figure 2- Primitives of Meaning) the problems of addressing objects from one structural viewpoint within the perspective of another structural viewpoint are seen, even though the comprising primitives are the same for both systems. When these differences exist between a system or environment and a simulation component that needs to be reused in that environment, unless techniques similar to those described in [29] and [30] are taken, there are likely to be not only semantic differences, but also technical differences.

4.2.3 Differences in Resolution

Resolution is a challenging subject but, as with scope and structure, if a model is built to a certain solution and then reused in an environment with a different resolution, there may be serious problems. Where resolution is involved, the resolution may apply to either objects, where the objects of a model are neither higher nor lower, or to processes, where the computed changes to objects occur at certain time intervals and thus are neither faster nor slower. An example to consider comes from modeling in the military domain. Often a combat simulation will have its resolution at an entity, a small unit such as a squad or platoon, or a large unit such as a brigade or division. In all cases, the model may be used to show the same battle, but the resolution will be either high detail, in the case of an entity, or low detail, in the case of large formations such as divisions

4.3 Methods to Overcome Differences

There are some Modeling & Simulation oriented methods that can be used to overcome these differences, at least where objects or data are concerned

One approach that can solve some or perhaps many of these problems for Modeling & Simulation software components has been developed by researchers from the Virginia Modeling Analysis Simulation Center (VMASC). This approach leveraged work from researchers at other academic institutions as well most notably the Joint Forces Command (JFCOM), North Atlantic Treaty Organization – Allied Transformation Command (NATO-ACT) [28], and Naval Postgraduate School, . This approach is known as Model Based Data Engineering or MBDE.

4.4 Model Based Data Engineering

MBDE can be used to solve certain problems in the areas of data linguistics and structure and possibly resolution. Problems of scope, however, are likely outside of the realm of help that MBDE can provide.

Very briefly, MBDE is an approach to data engineering that is based on having an understanding of the models of the various systems involved.

The problems of Modeling & Simulation software interoperability are the same problems that arise when mismatches of the types we have been discussing occur in a reuse situation

MBDE traditionally requires several steps to be followed:

- Data Modeling
- Data Administration
- Data Management
- Data Alignment
- Data Transformation

The best source for a definitive treatment of MBDE is Tolk and Diallo [23]. It has been relied on for further work in a variety of publications since 2005, most notably in [24] and [25]. The use of MBDE as the core for a web services based simulation architecture is presented in [26].

4.5 The Common Reference Model

The Common Reference Model used in the MBDE solution is a data model that can capture all of the data exchange between the software component and the other elements in the new environment it is being reused in [27].

MBDE is intended to be a "difference buffer" where the various differences mentioned (i.e., linguistic, structure, resolution, and, in some cases, scope), are all ameliorated through the use of data mappings and perhaps some algorithmic transformation of the data.

4.6 Process Differences

Of course, while data object differences is the most common part of problems with reuse and also interoperability, there are also likely to be just as many differences in the area of processes. Unfortunately at the time of this presentation, there is not a general solution that can be applied using process engineering for either reuse or interoperability. There is, however, ongoing research at the VMASC in this area on a ‘Theory of Processes’ that will highlight the differences of how individual models handle processes. This approach is currently expected to be published in a few months. The expectation is the Theory of

Processes will lead to methods for process engineering during the next few years.

5 Conclusion

In the area of software or application engineering for Modeling & Simulation projects, we see some conclusions related to the idea of software reuse. The first of these are a number of observations regarding the particular relationship between reuse and M&S. The second are a number of observations of when the effort of attempting to design (or apply, after the fact) reuse is just too much, compared to the rewards of the effort.

5.1 Particular Observations about Reuse

As pointed out in the preceding sections, there is a particular relationship between M&S and the concept of reuse. Other than reusing abstract algorithms, or even particular pieces of software, in the M&S arena, the possibility for reusing *models* introduces both new obstacles as well as new opportunities. Some specifics related to this particular relationship are:

- Reuse for Modeling & Simulation software requires all of the considerations that reuse for other types of software require.
- In addition to the usual considerations, Modeling & Simulation software requires developers to be aware of the model and, in cases of reuse, to be aware of other models it will interact with.
- Software for reuse must be interoperable, at some level, with whatever software will exist in its new environment.
- The increasing levels of the LCIM introduce more requirements on the nature of the data that is exchanged. Usually this is because there is a common basis for the involved models or, at higher levels; the models must have some awareness of each other.

5.2 When is Too Much, Too Much?

The effort to either design for reuse, or to accommodate reuse after the fact with developed simulation packages is an effort that can bring great rewards (savings in time and development cost of course, but also the introduction of additional perspectives into a multi-model solution). However, no matter what the benefits are, it is possible that the requirements to reap those benefits may prove too costly. Here are some points to consider when attempting to evaluate the relationship between cost and reward in an M&S reuse effort.

- The many differences highlighted in this presentation often come up too frequently leading to simulation mismatch even among elements that were developed to be reused to operate together.
- The fixes recommended, especially for Model Based Data Engineering, are not simple tasks. The results are real and reliable; however, the effort can be costly in terms of developer hours and other resources.
- There are times when it may be preferable to simply re-engineer the software component for the new environment. However, in this case, much of the work can be saved and shortcuts taken if documentation, as repeatedly emphasized earlier, on the original model is available
- Even if the software is re-engineered, there are likely time and cost savings as the original component can certainly serve as a prototype where methods for implementation might be used and possibly portions such as the model's algorithms reused.

Acknowledgments

This work was produced based on a series of investigations and research done by VMASC personnel in support of a contract to Raytheon's Network Centric Systems group.

6 Works Cited

[1] Spaccapietra, S., Parent, C. and Dupont, Y. (1992). *Model Independent Assertions for Integration of Heterogeneous Schemas.* Very Large Database (VLDB) Journal 1(1): 81-126

[2] Parent, C. and Spaccapietra, S. (1998). *Issues and approaches of database integration*. Communications of the ACM 41(5): 166-178

[3] Gorman M.M. (2006) *Data Interoperability Community of Interest Handbook.* Whitemarsh Information Systems Corporation

[4] Reynolds, P.F., Natrajan A. and Srinivasan, S. (1997). *Consistency Maintenance in Multiresolution Simulations.* ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 3, 368–392

[5] Davis, P.K. and Bigelow, J. H. (1998). *Experiments in Multiresolution Modeling (MRM).* RAND Corporation

[6] C4ISR Interoperability Working Group (1998). *Levels of Information Systems Interoperability (LISI).* Report for the US Department of Defense. Washington, DC

[7] Winters L.S., Gorman M.M., and Tolk A. (2006). *Next Generation Data Interoperability: It's all about Metadata.*

IEEE Fall Simulation Interoperability Workshop, IEEE CS Press

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

[9] Petty, M.D. and Weisel, E.W. (2003). A Composability Lexicon. Proceedings IEEE Spring Simulation Interoperability Workshop, IEEE CS Press

[10] Harkrider, S. M. and Lunceford, W. H. (1999). *Modeling and Simulation Composability*. Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)

[11] Tolk A. and Muguira J.A. (2003). *The Levels of Conceptual In-teroperability Model (LCIM)*. IEEE Fall Simulation Interop-erability Workshop, IEEE CS Press

[12] Page, E.H., Briggs, R. and Tufarolo, J.A. (2004). *Toward a Fam-ily of Maturity Models for the Simulation Interconnection Problem*. IEEE Spring Simulation Interoperability Work-shop, IEEE CS Press

[13] Hofmann, M. (2004). *Challenges of Model Interoperation in Military Simulations*. SIMULATION 80: 659-667

[14] Turnitsa, C.D. (2005). Extending the Levels of Conceptual Interoperability Model. Proceedings IEEE Summer Computer Simulation Conference, IEEE CS Press

[15] Welty, C. and Smith, B. (2001). *Formal Ontology in Information Systems*: Collected Papers from the Second International Conference October 17th-19th, 2001. Assn for Computing Machinery

[16] Agarwal, S., Handschuh, S. and Staab, S. (2005). *Annotation, Composition and Invocation of Semantic Web Services*. Journal on Web Semantics 2 (1): 1-24

[17] Alesso, H.P. and Smith, C.F. (2005). *Developing Semantic Web Services*. A.K. Peters, Ltd.

[18] Chen, Y., Zhou, L. and Zhang, D. (2006). *Ontology-Supported Web Service Composition: An Approach to Service-Oriented Knowledge Management in Corporate Services*. Journal of Database Management 17 (1): 67-84

[19] Pillay A. (2000). *Model Theory*. Notices of AMS 47 (11): 1373-1381

[20] Sowa, J.F. (2000). *Knowledge Representation: Logical, Philoso-phical, and Computational Foundations*. Brooks Cole Pub-lishing Co.

[21] Yilmaz, L. (2004). *On the Need for Contextualized Introspective Simulation Models to Improve Reuse and Composability of Defense Simulations*. Journal of Defense Modeling and Simulation 1 (3): 135-145

[22] Yilmaz L. and Paspuletti, S. (2005). *Toward a Meta-level Framework for Agent-supported Interoperation of Defense Simulations*. Journal of Defense Modeling and Simulation 2 (3):161-175

[23] Tolk, A. and Diallo, S.Y. (2005). *Model Based Data Engineering for Web Services*. IEEE Internet Computing 9 (4): 65-70

[24] Tolk, A., Diallo, S.Y., Turnitsa, C.D. (2007) *Model-Based Data Engineering: Preparing a Paradigm Shift towards Self-Organizing Information Exchange*. Summer Simulation Conference SCSC'07, San Diego, CA, July

[25] Tolk, A., Diallo, S.Y., King, R.D., Padilla, J.J. and Turnitsa, C.D.. 2010. Conceptual Modeling for Composition of Model-based Complex Systems. In Conceptual Modelling for Discrete-Event Simulation, ed. S. Robinson, R. Brooks, K. Kotiadis and D.-J. van der Zee, CRC Press.

[26] Tolk, A., Turnitsa, C.D., Diallo, S.Y. (2006) C*omposable M&S Web Services for Net-Centric Applications*. pp. 27-44. Journal of Defense Modeling and Simulation, Vol. 3 No.1

[27] Tolk, A., Turnitsa, C.D., Diallo, S.Y., King, R. (2009) *Engineering Management for Complex Systems*. Proceedings of the 2009 Industrial Engineering Research Conference.

[28] Tolk, A. and Boulet, J. (2007). *Lessons Learned on NATO Experiments on C2/M&S Interoperability*. IEEE Spring Simulation Interoperability Workshop, IEEE CS Press

[29] Turnitsa, C.D., Tolk, A., Kewley, R. (2009) *Exploring Primitives of Meaning in Support of Interoperability*. IEEE Fall Simulation Interoperability Workshop, IEEE CS Press

[30] Turnitsa, C.D., Tolk, A. Kewley, R. (2010) *Further Exploration in Primitives of Meaning*. In Proceedings, 2009 Spring Simulation Multi-Conference.

Authors' Biographies

EMILY ANDREW is currently the Director of Modeling & Simulation for the Raytheon Company's Network Centric Systems. Formerly, she served 29 years on active duty in the United States Air Force where she worked a number of major modeling & simulation programs until retiring in July 2007. She has an M.S. in Electrical Engineering from the University of Colorado in Denver and has completed most of the course work at ODU for a Ph.D. in engineering with a concentration in Modeling & Simulation.

CHARLES TURNITSA is a Senior Project Scientist at the Virginia Modeling Analysis and Simulation Center at Old Dominion University (ODU). In addition he is also a Ph.D. Candidate, studying under Dr. Andreas Tolk at ODU. He has an M.S. in Electrical and Computer Engineering from that institution. His Ph.D. topic deals with the specific definition of modeled process.

ANDREAS TOLK is an Associate Professor in the faculty for Modeling, Simulation, and Visualization at the Engineering Management Department of the College of Engineering and Technology at Old Dominion University (ODU) in Norfolk, Virginia. He is affiliated with the Virginia Modeling Analysis and Simulation Center (VMASC). His domain of expertise is the integration of Modelng & Simulation functionality into real world applications based on open standards. He received a Ph.D. and an M.S in Computer Science from the University of the Federal Armed Forces in Munich, Germany.