A Peer Reviewed Online Computational Modeling Framework

Pieter J. Mosterman, Don Bouldin and Andrzej Rucinski

Abstract— **Along with theory and experimentation, computational simulation has become the third pillar of scientific discovery. While in industry computational modeling has seen application at an enterprise-wide level in the context of Model-Based Design, in academia models are typically still limited to isolated use by specialists. Once a project is completed, the intellectual property embodied by the model is lost. To harness the effort spent, a networked repository is proposed that stores peer-reviewed models. These models are evaluated whether they adhere to a set of quality requirements so they capture intrinsic value. This would facilitate the type of multi-disciplinary collaboration that is required to engineer the systems that have emerged and that continue to gain in importance. This work puts forward an outline of such a peer-reviewed online repository.**

Keywords— **Model repository, composability, engineering education; model-based design;**

I. Introduction

Where scientific inquiry has traditionally been performed based on theory and experimentation, more recently, computational science has come to be considered the third pillar of such inquiry [1]. This has further been recognized in a bill passed in the House of Representatives [2]. In the design and analysis of engineered systems, computational modeling and simulation has long been applied to aid engineers and scientists in studying the system dynamics. Tasks such as the design of feedback control, analysis of robustness characteristics of a controller, power requirements and performance characteristics of actuators, etc., are well suited for a computational approach. Clear advantages are evaluations that can be done more expediently in simulation than in real time and in a less expensive manner, tests that can be performed in operational regions of a system that may be considered dangerous, destructive, controversial, or simply impossible, and optimizations that can be automated in a computational framework. Furthermore, simulation can provide information about the system that may be difficult to obtain, for example because of instrumentation problems.

To perform such design and analysis tasks, a single engineer or scientist, or a team of them, would model a system for the particular task at hand. The result would be a dedicated model that captures the required phenomena to solve a specific problem. Since reusability often would be an afterthought, there was no need to ensure a model could be employed in a different context. One manifestation of this lack of reusability would be certain behavioral scenarios (e.g., a failure sequence) to be inherent in the model since these would be the only scenarios of interest to a team of engineers and scientists (e.g., [3]).

More recently, Model-Based Design has found its way into many stages of the overall design of a system [4]. To reap the full benefits of Model-Based Design, there has been an increasing effort to relate, combine, and integrate models that have been developed by other design teams. For example, in embedded control systems, a controller aims to operate a physical process, the *plant*, to achieve a desired behavior [5], [6]. The plant may be modeled by a team of engineers to better understand performance and power characteristics. To synthesize a controller model, the team of control design engineers may model the plant in less detail to capture the pertinent dynamic behavior. It would be advantageous to share the modeling effort between these teams.

The enterprise-wide adoption of Model-Based Design puts forward a need for substantial infrastructure support such as version control and configuration management. In addition, models of the different parts of the overall system need to be designed following a paradigm that supports connecting, combining, and even integrating models with each other. This requires a departure from the use of models for individual studies. For example, an essential aspect is the separation of core algorithm and test bench. By encapsulating the core algorithm in a separate model component with explicit input and output, reuse is greatly improved, if only, because no model modifications are required by users other than the original model designer.

Not only do the separate parts of the models have to be combined for subsystem and system-level studies in one stage of development, the parts have to also be combined while in different stages of development. An example of this is processor-in-the-loop (PIL) simulation, where the target hardware and production software is tested with a simulated model of the plant [4].

In academia, the use of models is still mostly confined to isolated studies. To an extent, this is predicated on the merit system that favors the specialist, which, in turn, hampers multi-disciplinary schooling and research. In a recent report [7], this lack of cross-domain knowledge and activity has been identified as a key element to further the present state of science and technology.

So far, the increasing use of computational modeling in combination with the merit system has resulted in manuscripts that document models developed by solid engineering efforts submitted for peer review. The lack of scientific contribution, however, often prohibits publica-

Pieter J. Mosterman is with The MathWorks, Inc., Natick, MA. E-mail: pieter.mosterman@mathworks.com

Don Bouldin is with the University of Tennessee. E-mail: dbouldin@tennessee.edu

Andrzej Rucinski is with the University of New Hampshire. E-mail: andrzej.rucinski@unh.edu

tion. Furthermore, the paper documentation often highlights certain aspects but does not provide sufficient information to reconstruct the described model. Moreover, it typically is impossible to validate the correctness of the model.

To address these issues, a network repository is proposed where computational models can be submitted for peer review. Having a formal evaluation procedure in place allows for creating a body of high quality executable models. These models can then be made available for use in academic or industrial projects, thereby harnessing the intellectual achievements that are currently often lost. Furthermore, it will be conducive to the multi-disciplinary perspective that science and technology must assume to address the present-day challenges. It will further prepare students for an industrial environment where formal reviews of models are mandatory and the ability for colleagues to use another one's model is essential. Finally, it is seen as a step towards a merit system that has moved beyond recognizing the achievements of an individual in isolation, but towards recognizing such achievements as part of an ecosystem of modeling and simulation efforts by colleagues.

In Section II some background is provided on online repositories for data and model storage and this is contrasted with some of the needs for sharing models of dynamic systems. Section III outlines Model-Based Design and some of the technologies and methodologies that are being applied at an enterprise level. In Section IV requirements for models to be of value to a community are discussed. Section V then puts forward a set of criteria that are intended to form the foundation of a peer reviewed repository. In Section VI the conclusions of this work are presented.

II. Background

The one-off design of computational models is a common place practice in academia. Once a project such as a Ph.D. dissertation is completed the models are disbanded and the significant effort involved in their construction is lost. Some efforts to capture the investment for posterity may be by means of publishing a paper that describes the model, but such publications invariably lack information to reconstruct a model that faithfully reproduces presented results, if it can be reconstructed in an executable form at all.

In some domains, the benefit of making algorithms and data electronically available has been recognized for a long time now. For example, algorithms that have been implemented in MATLAB® [8] can be made available on the MATLAB Central web site.[1] Although MATLAB Central tailors well to algorithms, it lacks support for some of the features that are desired by a repository of dynamic models. For example, whether a dynamic model suits a purpose, it is essential to know the details of its interface (data type, sample rate, dimensions, etc.). At present, MATLAB Central does not facilitate including this as search criteria.

Similar to MATLAB Central, Google[2] allows searching for syntactic matches of text string in the file name of a model and in cases such as a text file format, for syntactic matches in the file content. Again, this lacks the deep semantic search options that are required to efficiently find models of dynamic systems. In addition, models typically consist of a number of files such as initialization scripts, parameter values, the core algorithm, libraries, test bench, etc. Simply obtaining one model file tends to be insufficient to have an executable model. Furthermore, models on MATLAB Central as well as models found with Google are typically closed entities with no exposed interface of input and output variables. This significantly impedes the use of a model for a purpose different from the original intent as the model will have to be edited; a very undesireable activity.

The Physiome Project[3] aims to establish a database of models in the computational biology domain. Models can be submitted via a Wiki interface and be commented on by other users. Each model is required to be accompanied by a peer reviewed article or a detailed description. Some modeling guidelines are provided and a classification into models that are more or less formally validated and verified is provided. Though comments may be provided, the database is not peer reviewed. Furthermore, search only facilitates a syntactic match of the XML based representation of a model.

In the domain of electronics design automation, there are several repositories of intellectual property (IP). For example, the OPENCORES project[4] aims to freely publish open source hardware in the form of IP cores of computation under a liberal license. The objectives include developing standards, development tools, and IP source and documentation.

Still other projects exist, however, none of them address the needs outlined in Section I. In particular, the compositionality, search facilities, test benches, and peer reviewed status are key ingredients that are missing.

III. Model-Based Design

To understand the requirements for an online peer-reviewed repository, it is useful to evaluate Model-Based Design at an enterprise-wide level with specific attention directed to system decomposition. This provides insight into needs for using computational models as the foundation of the efforts of many individuals and teams in a wide spectrum of tasks and domains.

A. The Essence of Partitioning

To negotiate the complexity of modern embedded systems, the partitioning principle is typically employed [9]. The use of partitioning is based on the *divide and rule*[5] adage. Assuming that the complexity of the design task increases exponentially with the size of the system under design, breaking the task up in partitions that can

[1] http://www.mathworks.com/matlabcentral/
[2] http://www.google.com
[3] http://www.physiome.org/
[4] http://www.opencores.org/projects.cgi/web/opencores/mission
[5] From the Latin *divide et impera*.

be composed at linear expense, the cumulative complexity reduces. This has allowed the design of systems that are much beyond the cognitive abilities of one or a team of engineers. In turn, the partitioning prohibits optimizations across the boundaries of the individual partitions. Therefore the partioning is a delicate process that must be performed with care.

A good partitioning minimizes the dependencies of the partitions and optimizes each of them individually, along with verifying and validating the partitions.[6]

In general, a number of reasons to partition can be listed:

- To support re-use.
- To avoid having to work with the largest assembly, which results in large overhead on development resources.
- To isolate errors and defects.
- To support multi-user development.
- To reduce complexity
- To be more efficient, as incremental technologies can be applied for:
  - Code generation
  - Code and model compilation
  - Verification & Validation (V&V)
  - Test vector design
  - Failure Mode, Effects, and Criticality Analysis (FMECA)

If context and interfaces are strict and well-defined, dependencies between partitions can be eliminated. When properties for the partitions are proven, the properties then hold under composition. Therefore, more advanced analysis and design methods of higher computational complexity can be applied to the smaller components.

Once a system is partitioned, design teams can work on the separate partitions. The partitioning then supports the version control and configuration management systems that are imperative for successful concurrent engineering.

In general, the design of a system is decomposed in three dimensions. In one dimension, the system is structurally partitioned into subsystems. These subsystems, in turn, can be structurally partitioned, which results in a hierarhical dimension. In the third dimension, a functional decomposition of the different stages of the design of a partition is employed. In this dimension, a subsystem passes through representations such as continuous time feedback control, its discretized and sampled time version, and then the scheduled control functionality to be implemented on the microcontroller [10].

To illustrate the functional decomposition, consider a fuel injection controller. The initial specification may define the interface. Based on this, control synthesis techniques may be employed by the control design engineers to design the control algorithm. All of this is often done in the continuous-time domain and aspects such as stability and response time are addressed. In the next stage, the continuous-time control law is discretized in time to move it closer to an implementation on a microprocessor. Additionally, discretization of values may be performed in case of a fixed point implementation. In this stage, aspects such as processor load and signal integrity are studied. The next stage requires integrating the controller with the rest of the system and so aspects such as mode switching, data transformations, signal filtering and data analysis are subject of study. In the next stage, software engineers who are responsible for the implementation on the target processor are concerned with executing the functionality in threads, scheduling, etc.

In order to minimize redoing earlier design effort, it is beneficial to have a system view throughout the different design stages of the separate partitions. For example, to serve as a natural test bench. Furthermore, the effort expended in an earlier design stage is best leveraged in later design stages (e.g., [11]). This requires relating, combining, and integrating models of system partitions that are in different stages of their design but also in rather different domains such as mechanical, electrical, or the electronics and software domain.

B. Communication Between Humans

Though it is important that models that are being shared are sufficiently robust and stable so that they do not have to be understood in detail by users other than the original design team, it is also important to ensure that a model is readable still, if only for purposes of review. A quick and sound understanding by humans is typically achieved by adhering to a familiar modeling style and properly documenting the model content.

B.1 Style Guidelines

Style guidelines help greatly in efficiently forming an understanding of the content of a model. In industry, style guidelines are used throughout the design of a system, ranging from formulating requirements to graphical models to programming code. For code, lint tools are popular but other style guidelines such as the Motor Industry Software Reliability Association (MISRA)[7] exist. For models, style guidelines such as the MathWorks Automotive Advisory Board (MAAB)[8] guidelines for Simulink® [13] and Stateflow® [14] models are extensively used. Tools to check graphical modeling style guidelines automatically are available [15].

An example MAAB style rule [12] that addresses the interface of basic blocks in a graphical model is given in Table I. It mandates that a basic block shall not have a bus as input where a bus may consist of a number of named variables that can be of different data types. It further states that a basic block may have a vector as input and output, where a vector consists of a number of indexed variables of the same data type.

B.2 Documentation

Documentation is one of the most reliable modalities to help develop a deep understanding of the content of a

[6]Verification is determining whether the system has been designed correctly, whereas validation is determining whether the correct system has been designed.

[7]http://www.misra.org.uk/

[8]http://www.mathworks.com/industries/auto/maab.html

TABLE I
4.5.1.2.DB_0086: BASIC BLOCK INTERFACE SIGNALS

ID: Title	**db_0086: Basic block interface signals**
Priority	mandatory
Scope	MAAB
Automation	possible
Prerequisites	
Description	A basic block ... • has no input busses. • has no output busses. • may have an input vector, if the block is vectorized. • may have an output vector, if the block is vectorized.
Benefit	Respecting the guideline ensures ... • accessible signals. • a clear system structure.
Penalty	Breaking the guideline ... • may cause problems with non-accessible signals. • may cause a lot of redesign work. • may cause problems or errors with custom tools.
Author	Daniel Buck
Last Change	07.02.2000, Daniel Buck

model. Documentation can be in the form of a narrative describing particular aspects of the model or in the form of a set of structured requirements. In either case, facilities to hyperlink descriptions or requirements with pertinent model elements help quickly develop an understanding of why and how the model has been constructed.

The documentation often contains references, for example, to the theory of physics based on which certain equations are derived. Furthermore, often models are incrementally improved from version to version of a product. Reference to previous versions and substantiations as to why certain changes are made are likewise important.

IV. Model Composability

After a system has been partitioned and the separate partitions have moved through their design stages, the eventual implementation needs to be assembled. In the ideal case, this would be a matter of model composition, i.e., the properties of the partitions in isolation still hold under composition. In practice, it is a matter of integration instead, where the properties of the partitions are affected by other partitions.

Typical properties that are affected in the integration stage are those that relate to a specific implementation. Such properties may be referred to as *parafunctional*. For example, when a software task is executed on a microprocessor that is also responsible for another task, interaction and interference because of the shared resource may occur. Another source of interference that requires an integration effort is the direct interaction between partitions and the assumptions made on the interface. Finally, the indirect interaction between partitions across mediating partitions, especially in feedback configurations (such as across the plant in feedback control configurations) may cause interference as well [16].

A. Defining Interaction

An important step in alleviating the integration effort is by defining the interactions between partitions rigorously. This interaction is made up by the interface and its behaviors.

A.1 Interface Definition

To rigorously define an interface, first, the syntax has to match. In particular, complex interfaces contain data with extensive and sophisticated hierarchical structures that have to match. In software engineering, such complex interface data is often captured as a `struct`. From a system engineering perspective, such interface data may be captured by a *bus*. Hierarchical buses can be automatically translated into `struct` elements in the generated code, where facilities are available to enforce contiguous memory to be used. This is especially important if large amounts of data have to be transferred across the interface often.

A hierarchical bus in the model of an automobile power train could be as follows:

```
33 typedef struct {
34    transmission transmission;
35    engine engine;
36    real_T throttle;
37    real_T brake;
38    real_T speed;
39 } CANB;
```

In the model, the `CANB` bus consists of the signals `throttle`, `brake`, and `speed`, as well as two other buses, `transmission` and `engine`. These buses consist of signals, again, where the generated struct for the `engine` bus is:

```
24 typedef struct {
25    real_T w;
26    real_T EGO;
27    real_T MAP;
28 } engine;
```

This struct shows the signals that comprise the `engine` bus to be `w` for angular velocity, `EGO` for estimated generated oxygen and `MAP` for manifold air-pressure.

A.2 Context Definition

In addition to a static match of interfaces, the dynamics have to be aligned. In other words, the data provided at the interface must adhere to the assumptions based on which the receiving model has been designed [17]. Such assumptions may vary from minimum and maximum values of an interface variable to a specific frequency band, sample rate, event sequences, power load, etc. As part of the Ptolemy project [18], the type of interface variables is aimed to be extended by a definition of dynamic semantics.

A dedicated language to capture this dynamic information appears to be most convenient to the modeler. Here, assertions are a prime candidate as they have proven their utility in software engineering to validate part of the context in which a program executes is according to assumptions made by the programmer. Likewise, assertions can

be included in models. To illustrate, a model of a transmission controller may take as one of its input a throttle signal. The design may be made under the assumption that the throttle signal takes on values between 0 and 100. If the controller is operated outside of the bounds of this region, an assertion that the context is violated is thrown.

B. Model Robustness

To minimize the system integration effort, it is important that the models of the partitions are thoroughly evaluated before being used as the basis for an implementation that becomes the artifact of integration. Such evaluation can utilize a set of approaches that have been developed and are part of design of engineered systems in industry.

When a model is shared, often it is highly undesirable for a user other than the one who originally constructed the model to have to make any modifications. Otherwise, the effort expended in understanding the model in order to appropriately make the changes may be as high as the effort to construct the model from the start.

Therefore, shared models need to be thoroughly tested. Measures of how well a model has been tested are often based on coverage analyses. For example, statement coverage analysis may provide a measure of how many statements out of the total set have been evaluated when the model is subjected to a set of tests.

Other types of coverage analyses exist. Notably, to certify a system under DO-178B [19] safety regulations as used by the Federal Aviation Administration (FAA), a modified decision/condition coverage (MC/DC) [20] analysis has to evidence 100% coverage. To achieve MC/DC, every point of entry and exit in the program must have been invoked at least once, every condition in a decision in the program must have taken on all possible outcomes at least once, and each condition must have been shown to affect that decision outcome independently.

V. Review Criteria

Given the evaluation of strategies, principles, and best practices to successfully employ Model-Based Design at an enterprise-wide level, a set of rules for a peer-reviewed repository is provided. These rules intend to ensure that accepted models are of high quality as judged by a number of criteria. These rules can be organized to have the following classification of objectives:

- **Composable**—ensure an interface with well-defined elements and meta characteristics (e.g., data type, sample time, etc.) as well as carefully defined assumtions (valid operational regions).
- **Reusable**—ensure that the model is robust and stable by providing a separate test bench that evidences MC/DC analysis of 100%.
- **Understandable**—adhere to proper modeling guidelines (e.g., low cyclomatic complexity of control structures, uncluttered diagrams, etc.).
- **Substantiated**—provide traceability to the documentation (e.g., by links to structured requirements), include a formal document with a description of the model content, including references to related models and literature.
- **Findable**—include meta information about interfaces, numerical solvers that are required, number of elements in the model, different formalisms used (differential equations, differenence equations, state machines, etc.), and cyclic dependendencies (algebraic loops).
- **Consistent**—provide verification results (e.g., is a divide by zero protected).
- **Correct**—show validation of the model with respect to theoretical data, experimental data, or simulated data obtained by models qualified under the peer-review process.

TABLE II
1.1: Interface definition

ID: Title	rule_1.1: Interface definition
Priority	mandatory
Description	A submitted model ... • has to have at least one input port. • has to have at least one output port. • has to define the structure of data on the input ports. • has to define the structure of data on the output ports
Benefit	Respecting the guideline ensures ... • the model can connect based on the interface information only.
Penalty	Breaking the guideline ... • may make it impossible to connect to the model. • may cause difficulty connecting to the model. • may make connecting to the model context dependent.

For each of the classes of criteria, one or more rules can be formulated that constitute the foundation of a review process. For example, Table II captures a rule to enforce proper interface definitions so a model is composable.

VI. Conclusions

Computational modeling and simulation has become the third pillar of scientific discovery. This is exemplified by the use of modeling and simulation in research and development. In case of the latter, industry has started implementing Model-Based Design at an enterprise-wide level. In order to maximize the return on the modeling investment, models have to be constructed so they can be conveniently reused by different design teams.

In academia, modeling is still very much an isolated effort of the specialist. Once the task for which a model is designed is completed, for example, because a student graduates, the model is typically disbanded.

To harness the invested effort for the community, a networked repository to store models of dynamic systems is proposed. The idiosyncrasies of such models and specific needs for a successful implementation of such a repository are discussed. Peer review of submitted models is put forward as an essential aspect. This relates to the level of scrutiny that is found in industry where models are reviewed by colleagues before allowed to be checked in.

A classification of model quality characteristics is presented based on which a set of rules can be derived to form the criteria based on which a peer-review process can be

founded.

References

[1] President's Information Technology Advisory Committee, "Computational Science: Ensuring America's Competitiveness," June 2005.

[2] J. R. Forbes, T. Drake, T. Feeney, J. A. Davis, R. Keller, K. M. Conaway, T. Davis, H. Wilson, and S. P. Ortiz, "Recognizing the contribution of modeling and simulation technology to the security and prosperity of the United States, and recognizing modeling and simulation as a National Critical Technology." Bill H. RES. 487, June 2007.

[3] D. Moormann, P. J. Mosterman, and G.-J. Looye, "Object-oriented computational model building of aircraft flight dynamics and systems," *Aerospace Science and Technology*, no. 3, pp. 115–126, 1999.

[4] P. J. Mosterman, S. Prabhu, and T. Erkkinen, "An industrial embedded control system design process," in *Proceedings of The Inaugural CDEN Design Conference (CDEN'04)*, Montreal, Canada, July 2004, cD-ROM: 02B6. [Online]. Available: http://msdl.cs.mcgill.ca/people/mosterman/papers/cden04/fp.pdf

[5] K. J. Åström and B. Wittenmark, *Computer Controlled Systems: Theory and Design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1984.

[6] R. C. Dorf, *Modern Control Systems*. Reading, MA: Addison Wesley Publishing Co., 1987.

[7] President's Council of Advisors on Science and Technology, "Leadership Under Challenge: Information Technology R&D in a Competitive World," Aug. 2007.

[8] MATLAB®, *The Language of Technical Computing*, The MathWorks, Inc., Sept. 2007.

[9] K. Wijbrans, "Twente hierarchical embedded systems implementation by simulation: a structured method for controller realization," PhD Dissertation, University of Twente, Enschede, The Netherlands, 1993, iSBN 90-9005933-4.

[10] R. A. Hyde, "Fostering innovation in design and reducing implementation costs by using graphical tools for functional specification," in *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, Monterey, CA, Aug. 2002.

[11] S. Vestal, "Software architecture workshop," Minneapolis, MN, July 1994. [Online]. Available: http://www.htc.honeywell.com/projects/dssa/ftp/papers/arch-wkshop.ps

[12] MathWorks Automotive Advisory Board, "Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow®," e-Guidelines, July 2007. [Online]. Available: http://www.mathworks.com/industries/auto/maab.html

[13] Simulink, *Using Simulink®*, The MathWorks, Inc., Natick, MA, Sept. 2007.

[14] Stateflow, *Stateflow® User's Guide*, The MathWorks, Natick, MA, Sept. 2007.

[15] M. Huhn, M. Mutz, K. Diethers, B. Florentz, and M. Daginnus, "Applications of static analysis on UML models in the automotive domain," in *Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2004)*, E. Schnieder and G. Tarnai, Eds., Braunschweig, Germany, Dec. 2004, pp. 161–172.

[16] P. J. Mosterman, J. Ghidella, and J. Friedman, "Model-based design for system integration," in *Proceedings of The Second CDEN International Conference on Design Education, Innovation, and Practice*, Kananaskis, Alberta, Canada, July 2005, pp. CD–ROM.

[17] B. Falkenhainer, A. Farquhar, D. Bobrow, R. Fikes, K. Forbus, T. Gruber, Y. Iwasaki, and B. Kuipers, "CML: A compositional modeling language," Knowledge Systems Laboratory, Stanford University, Stanford, CA, Tech. Rep. KSL-94-16, Jan. 1994.

[18] J. Davis, II, R. Galicia, M. Goel, C. Hylands, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong, "Ptolemy II – heterogeneous concurrent modeling and design in java," http://ptolemy.eecs.berkeley.edu, Department of Electrical Engineering and Computer Sciences, University of Califoria at Berkeley, 1999, version 0.1.1.

[19] RTCA SC-167, "DO178B, Software Considerations in Airborne Systems and Equipment Certification," Dec. 1992.

[20] A. Dupuy and N. Leveson, "An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software," in *Proceedings of the Digital Aviation Systems Conference*, Philadelphia, PA, Oct. 2000.