EASST

Proceedings of the
4th International Workshop on
Multi-Paradigm Modeling
(MPM 2010)

Combining SysML and Model Transformations to Support Systems
Engineering Analysis

Aleksandr A. Kerzhner and Christiaan J.J. Paredis

12 pages

Combining SysML and Model Transformations to Support Systems Engineering Analysis

Aleksandr A. Kerzhner[1] **and Christiaan J.J. Paredis**[1]

[1]G. W. Woodruff School of Mechanical Engineering,
Georgia Institute of Technology, Atlanta, GA, USA

Abstract: As modern systems become increasingly complex, there is a growing need to support the systems engineering process with a variety of formal models, such that the team of experts involved in the process can express and share knowledge precisely, succinctly and unambiguously. However, creating such formal models can be expensive and time-consuming, making a broad exploration of different system architectures cost-prohibitive. In this paper, we investigate an approach for reducing such costs and hence enabling broader architecture space exploration through the use of model transformations. Specifically, a method is presented for verifying design alternatives with respect to design requirements through automated generation of analyses from formal models of the systems engineering problem. Formal models are used to express the structure of design alternatives, the system requirements, and experiments to verify the requirements as well as the relationships between the models. These formal models are all represented in a common modeling language, the Object Management Group's Systems Modeling Language (OMG SysML[TM]). To then translate descriptive models of system alternatives into a set of corresponding analysis models, a model transformation approach is used to combine knowledge from the experiment models with knowledge from reusable model libraries. This set of analysis models is subsequently transformed into executable simulations, which are used to guide the search for suitable system alternatives. To facilitate performing this search using commercially available optimization tools, the analyses are represented using the General Algebraic Modeling System (GAMS). The approach is demonstrated on the design of a hydraulic subsystem for a log splitter.

Keywords: systems engineering, SysML, model integration, model to model transformation, requirements modeling

1 Introduction

Engineered systems are becoming increasingly complex to design because of greater consumer expectations, highly integrated products encompassing various engineering domains, and geographically distributed stakeholders. To manage this complexity, systems engineering can be applied; systems engineering is an interdisciplinary approach to creating and verifying an integrated set of system solutions to satisfy customer needs.

The systems engineering process generally consists of problem definition, analysis, and interpretation [SA00]. Formulating the problem usually involves several common tasks, including

defining the system objectives, deriving requirements for the system, and generating candidate solutions. Several iterations of these core tasks may be needed to derive a suitable final system. In an effort to precisely and unambiguously express the knowledge present in a systems engineering problem, systems engineers have begun to adopt the Model-Based Systems Engineering (MBSE) approach [Fis98]. Within MBSE, engineers represent all aspects of the problem using formal models; such models generally include system alternatives, requirements, experiments to verify the requirements and also models relating these different aspects. There are many MBSE approaches that prescribe a work flow for modeling and solving the problem [Est07].

A key challenge during any MBSE process is verifying that the prescribed objectives and requirements are met by a particular alternative. Usually, designers manually create any necessary analyses by incorporating knowledge of various domains. For complex problems, manually creating such analysis models requires significant time and effort and introduces many opportunities for error. An additional complication in many MBSE approaches is that the knowledge needed to generate these analyses is captured in diverse and incompatible tools and representation syntaxes.

This paper proposes a method for automatically generating analysis models for systems engineering problem models. The entire problem is represented in a common language, the Systems Modeling Language (SysML^{TM}) from the Object Management Group (OMG) [OMG08]. SysML is chosen because it allows the representation of the very distinct elements needed in a single language partially overcoming the challenge of using diverse tools and representations. Also, a common language allows for the expression of relationships between different facets of the problem. To automatically generate domain-specific analysis models, model transformations are utilized to combine knowledge from various models and problem-independent model libraries. Such libraries facilitate model reuse and can be used to express common structural components and their relationships to the appropriate analysis models. Model transformations are also used to transform the problem-specific analysis models into executable simulations. Finally, meta-data is associated with both models and relationships to efficiently describe their role within the systems engineering problem and facilitate reuse.

Other work has identified a need for efficient architecture and have proposed computational tools for synthesizing alternatives [AR04, BSS07, SSS05]. To analyze these alternatives, usually these works generate one type of analysis model which focuses either on geometric considerations or simple functionality. This paper supports this work by presenting an approach for generating a more varied and comprehensive set of analyses to evaluate a potential solution.

The remainder of the paper is presented as follows. In Section 2, the general approach to formally modeling meta-data and relationships within SysML is presented. This general approach is applied to the modeling of a hydraulic log splitter problem in Section 3. The problem definition is then used in conjunction with model transformations to automatically produce an executable simulation in Section 4.

2 Approach: Model Management in SysML

Since MBSE prescribes formal modeling throughout the systems engineering process, a large number of models may be needed to describe a particular problem. Even if these models are encoded in a single file, it is still important to identify which facet of the problem is being
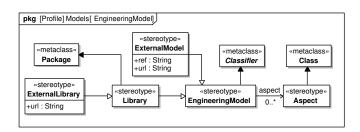
Figure 1: Profile for defining engineering models in SysML.

expressed by each individual model and how the models relate to each other. In our approach, the role of a particular model is characterized by associating related aspects. Also, to express the relationships between the models each part of the systems engineering problem is modeled in a common language, SysML. This alone is insufficient because it only facilitates expressing relationships between models; a generic, consistent, and computer interpretable approach for expressing the relationships is also needed.

SysML is chosen because it is a visual language designed to represent an entire systems engineering problem [FMS08]. The basic unit of SysML is a *Block*, which can be used to describe the system, its components, or other constructs of interest. SysML is also flexible enough to express the additional knowledge such as aspects and relationships. Because models of both the meta-data and relationships are captured within SysML, the entire representation is in a common formalism.

The approach for characterizing the models is based on Multi-Aspect Component Models (MAsCoMs) [Job08] where *aspects* are used to characterize analysis models by describing what the model represents, the representation syntax, and how it can be composed with other models. Unlike MAsCoMs, *aspects* are used here to characterize any model expressing the systems engineering problem, not just analysis models. We will refer to these models as engineering models. The concept of aspects is similar to those in aspect-oriented programming [IKL$^+$97] in that they characterize models based on their function. Also, models can have any number of associated aspects to describe their function. Unlike aspect-oriented modeling [CL02], these aspects are meant to characterize the models for composition and search purposes, not add any additional crosscutting features.

A profile is used to extend the SysML language to formally define engineering models and any other constructs unique to our approach [OMG08]. A profile is a Unified Modeling Language (UML) concept that is used as a light-weight extension mechanism for adding new constructs to UML or SysML [ISO05]. The engineering model concept is captured using a stereotype as is illustrated in Figure 1. The *EngineeringModel* stereotype has an *aspect* property which allows any *EngineeringModel* to be unambiguously classified with aspects. The *Engineering-Model* also extends from the Classifier meta-class forcing any engineering model to allow for generalization/specialization relationships. To enable reuse, the possible aspects are captured in a model library where they are organized in a hierarchical fashion and stereotyped with the *Aspect* stereotype.

Along with capturing models of interest, relationships between them are also captured within SysML. SysML provides many of the relationships necessary for modeling systems engineering
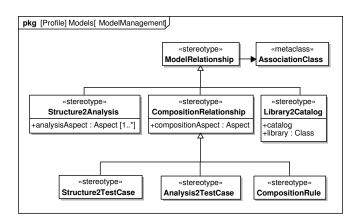
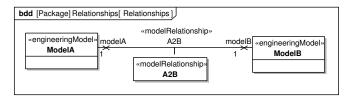Figure 2: Model Management Profile defining some possible model relationships.

Figure 3: Example Relationship between Model A and B in SysML.

problems. When existing relationships are not suitable, the UML *AssociationClass* construct is used to define a new relationship. Usages of these types can then represent a particular instance of that type of relationship. Again, a profile is used to enumerate the different relationships used in this approach, illustrated in Figure 2. For example, *CompositionRelationship*s describe how models can be composed together into more complex models.

An *AssociationClass* between two models is shown in Figure 3. This illustrates the *ModelRelationship* A2B between *EngineeringModels* A and B. This relationship expresses that usages of A and B can be related by usages of *ModelRelationship* A2B. In addition, one can express how the parameters and ports of these models are linked when such a model relationship exists.

The other relationships will be discussed in more detail in the following sections. Now that the general framework for capturing models of interest and relationships between them in SysML has been presented, the next section will cover how a systems engineering problem is expressed using this model management framework.

3 Defining the Systems Engineering Problem

The design of a hydraulic subsystem for a horizontal acting hydraulic log splitter is used as an illustrative example in the subsequent sections. This section describes how the problem of designing the hydraulic subsystem is expressed within our framework. A log splitter is a system used to divide cylindrical pieces of wood longitudinally.

This example is chosen because it involves the composition of well-defined, modular compo-
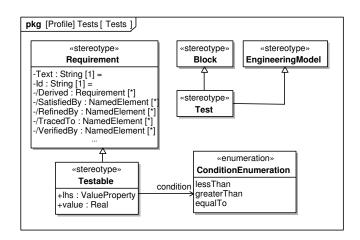
Figure 4: Profile for defining tests and testable requirements in SysML.

nents into a more complex system. Also, the design of the system must satisfy several competing requirements; the hydraulic circuit should be cost effective, light weight, and capable of actuating the wedge with both high force and high velocity. The definition of this problem consists of three major parts:

- Requirements the system should be designed to meet.

- Experiments that can be performed on the system to verify that requirements are met.

- System topologies under consideration.

This section explores how each of these is captured within SysML. We will begin with the modeling of the system requirements. One advantage of SysML is the existing constructs for modeling requirements and common relationships. These existing relationships can be used to capture how requirements are decomposed, verified, and satisfied.

The requirements begin with abstract specifications which describe how the system should behave qualitatively. These requirements are then decomposed into more concrete specifications on the system. To simplify the verification process, the requirements are further decomposed to an abstraction level where they can be quantitatively verified through experiments; that is until a particular property of the system can be bounded. Currently, this is accomplished deterministically by constraining a variable with an equality or inequality condition.

The next step is to model the experiments or test cases needed to verify that a system satisfies the specified requirements. Experiments or test cases will be referred to as tests in this paper. To formally model tests within SysML, some additional constructs are needed to formally define a quantitatively-verifiable requirement (or testable requirement), a test, and the relationship. A profile is again used to define these constructs, as shown in Figure 4. Two stereotypes are added, the *Testable* stereotype for requirements and the *Test* stereotype for experiments. The *Testable* stereotype derives from the standard SysML Requirement with additional properties for capturing precisely which system variable is being bounded by the requirement.
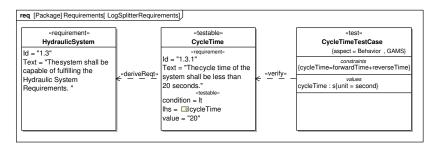
Figure 5: Requirements Breakdown for the Log Splitter's cycle time.

The use of these stereotypes along with existing SysML constructs to model requirements, derived requirements, and tests for a small portion of the log splitter's requirements is illustrated in Figure 5. In this example, a high-level requirement is decomposed into a testable requirement which can be verified by the defined test. Existing SysML relationships are used to describe that the testable requirement is derived from a high-level requirement and that the test should verify the testable requirement.

One important characteristic of the test definition is that it is defined independently of any particular design alternative. To separate the test definition from the structural definition, the test is defined using only a system boundary that captures the interface to the environment that all design alternatives should realize. The test specifies the state of the environment through inputs to the system as well as the parameters of the system that are to be measured. The state of the environment is described both by connecting the system boundary inputs to appropriate models and constraining the appropriate variables. Since the design alternative extends from the system's boundary, it has the same interfaces. Therefore, any constraints placed on the system boundary can be transferred to the test for a particular design alternative.

Once the modeling of the requirements and tests are complete, models of the possible system topologies are needed. A very simple hydraulic circuit for the log splitter will be used as the system topology under consideration. In this design, the wedge of the log splitter is pushed by a hydraulic piston which provides the force necessary to split the wood. An engine powers a constant displacement pump that, when engaged, causes fluid to flow through the system pushing the cylinder. The system topology modeled in SysML is shown in Figure 6. This model of the topology is a specialization of the system boundary. It realizes the rod interface which splits the wood and the control interface that receives input from the user of the log splitter.

Since the log splitter is comprised of common modular components, we can reduce the modeling effort by storing these components in a model library and reusing them. The library model for the cylinder is shown in Figure 7. Within this library model, common properties and ports of the cylinder are defined, such as the stroke length or bore diameter. This cylinder model can be specialized by more specific types of cylinders or, as is the case here, specific vendor-provided products which have certain values for each attribute. A combination of vendor-provided components can represent a particular system embodiment.

Now that the systems engineering problem has been defined, the next step is to solve that problem and find a system that fulfills the requirements.
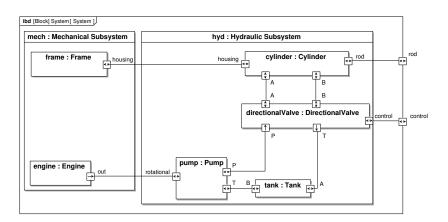
Figure 6: A SysML model describing the Log splitter architecture. An engine provides power to the hydraulic subsystem which is used to actuate a splitting wedge.
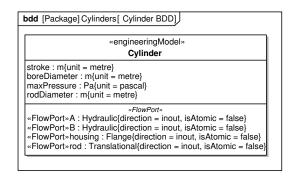

Figure 7: A Cylinder model from the model library.

4 Modeling and Composition of Analyses

Once the problem has been defined, it must be solved. To execute a particular set of tests, one may need a variety of analysis models. These analysis models can then be simulated to ensure that a design alternative satisfies the requirements. This section addresses how a number of these analyses can be created for a particular design alternative, and how they are converted into executable simulations. In this case, the definition of the analysis models appears completely within SysML. When an analysis model is located outside of the SysML environment, the *ExternalModel* stereotype defined in Figure 1 can be used to reference it.

Analyses are modeled within SysML as blocks with ports, equations, and properties in a port-based modeling approach [PDSK01]. The equations in this example are purely algebraic and modeled using the General Algebraic Modeling System (GAMS) syntax [BKM88] to allow the problem to be solved with commercial tools. Ports are used to describe the interfaces of analyses; these interfaces often abstract interfaces of the real components being modeled. The ports are connected together using SysML connectors to represent connections between the interfaces. In
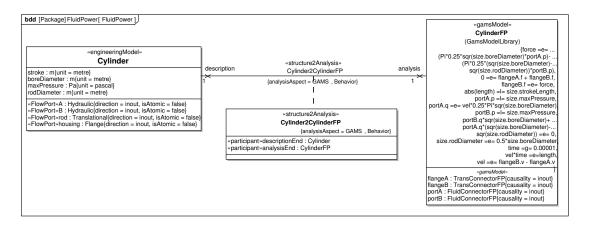
Figure 8: Relationship between Cylinder and Behavior model.

some cases, these connectors represent energy flows between the ports instead of simple equality relationships. The properties express any variables or constants that are used in the equations.

Once component-level analysis models are constructed within SysML, they can be automatically composed into system-level analyses and transformed into executable simulations. A graph-based model transformation approach is used to bridge the gap between design alternatives and analyses. The transformation to executable simulations is handled in two phases. First, the tests are transformed into a set of analysis models represented within SysML. Then, another transformation is used to create an executable simulation in a format compatible with a paradigm-specific modeling tool.

The transformation between test and analysis models is defined with the Fujaba story-diagram [FNTZ00] semantic using the MOFLON meta-case tool [AKRS06]. As is the case with many other graph-based model transformation approaches, the transformations are defined using a metamodel. The metamodel in this case is a SysML metamodel that was created from the UML metamodel and the SysML profile to simplify the use of SysML-specific constructs by reducing the use of tool-specific stereotype semantics in the transformations. Once these transformations are specified, MOFLON generates Java Metadata Interface (JMI) compliant code [Dir02]. This code can be packaged within a plug-in to execute the transformations in a specific model authoring tool, in this case the MagicDraw UML modeling tool with a SysML plug-in [NoM].

To generate a set of analysis models, the transformation starts by identifying the tests. For each test and system topology, a corresponding top-level analysis is created. For each component within the topology, a corresponding component-level analysis model is instantiated and included in the top-level analysis. The component-level analysis models are chosen based on their associated aspects and correspondence relationships to component models, such as the one shown in Figure 8. This particular correspondence relationship relates the cylinder component with a model describing the cylinder's behavior. The relationship is stereotyped with the *Structure2Analysis* stereotype and associated with the GAMS and Behavior aspects.

The relationship between the properties and ports of structural components and analysis models are also captured using SysML connectors. Properties from the cylinder, such as the stroke,
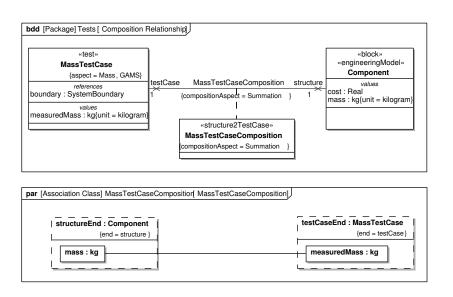
Figure 9: Composition relationship for Mass test used to add up the mass of each component into measuredMass.

are related to particular properties of the analysis. This allows the composition of the analysis model based on any values or connections present within the structural description of the design alternative. The use of similar templates for Finite Element Analysis (FEA) was shown by Bajaj *et al.* [MP07].

After the appropriate component-level analyses are instantiated, they are composed together using knowledge stored within composition relationships. This knowledge includes the attributes which must be added together into a single variable (i.e. mass or cost) or the interfaces that should be connected. Composition relationships describe how the analyses should be connected with each other and with the test. Certain attributes of the analyses are related to particular attributes of a given test. As an example, the composition relationship that component masses are added into a single system mass is illustrated in Figure 9. The relationship is associated with the summation aspect and owns a connector between the *mass* and *measuredMass* attributes which represents that the mass from each component should be added together into a single measuredMass attribute.

For analysis models where interfaces need to be connected appropriately, the correspondences between the interfaces of the structural and analysis models are used to instantiate a corresponding connection in the system-level structural model. In order to facilitate the creation of a deep-nested structure, connectors are added between corresponding component and analysis models. This also allows for traceability between the created analysis models and the original descriptive model of the system.

The result of the first phase of the transformation is a set of system-level analysis models generated from each test such as the one illustrated in Figure 10. This particular analysis includes constraints generated from a composition relationship (the component masses are added into a single attribute) and from a testable requirement (the mass should be less than 300 kg).
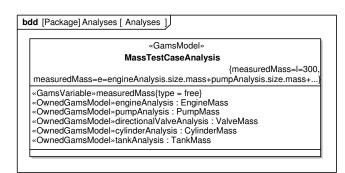
Figure 10: Analysis generated from the test for the alternatives mass.

Scenario	Component Sizing (Selection Id from Catalog)				Selected Variable Values				CPU Execution Time (s)
	Cylinder Id	Pump Id	Engine Id	Valve Id	Forward Force (N)	Total Mass (kg)	Total Cost ($)	Total Time (s)	
Maximize Force (N)	HMW-5032	SKP1NN_012	DP340E	NT-2020	139,833	94.9	993.5	20	2.82
Minimize Total Time (s)	HMW-3010	SKP1NN_012	DP390E	NT_Prince-2036	50,000	51.87	843.97	4.896	3.54
Minimize Total Cost ($)	HMW-4010	SKP1NN_012	DP240	NT-2020	53,698	51.3	657.4	9.69	2.45
Minimize Total Mass (kg)	PMC-5414	SNP2NN_4_0	DP160V	MSCDirect-01825629	52,013	32.25	708.6	9.15	78.13

Figure 11: Results from the GAMS optimization.

After the analyses are created within SysML, they need to be transformed into executable simulations. In theory, the generated simulations should test every possible design alternative using appropriate analyses to verify whether it meets the requirements. Since testing every possible combination would be computationally explosive, an optimization approach is used instead. The analyses are combined into a single non-linear mixed integer programming (MINLP) problem as demonstrated by Shah *et al.* [Sha10]. This MINLP problem is represented using GAMS and solved using the Branch-And-Reduce Optimization Navigator (BARON) [Sah96] to find several candidate alternatives that meet the requirements. A graph-based code generator specified using MOFLON is used to create the executable simulation code in GAMS syntax. Some solutions are shown in Figure 11. The solutions are composed of vendor products from the model library configured into the simple topology in Figure 6. Each solution meets the requirements and also maximizes a given objective function.

5 Discussion and Closure

As mentioned, the overall goal of MBSE is to explicitly model all aspects of the systems engineering process. This should include both the actual elements (systems structure, analyses, tests, and requirements) and the relationships between them. The overall goal is to improve traceability and reduce tedious effort by the designer during the design process. The framework presented

here addresses only a small portion of this goal by formally capturing the structure, requirements, and tests and transforming this captured knowledge into executable simulations.

One limitation of the current framework is that the test execution order is implicitly handled by a domain-specific solution tool. This approach would be insufficient if the analyses needed to be executed in different modeling tools or if they modeled the problem at different levels of fidelity. How to formally model this execution process within SysML is left for future work.

We view the work presented here as an initial step towards a more complete tool set where a problem definition is transformed and analyzed to automatically search for good design alternatives. From the problem definition, design alternatives could be automatically generated using captured knowledge [KP09], and then each alternative could be analyzed using models at varying levels of fidelity. The entire description of the problem could be maintained in a SysML model, executable simulations could be generated from this model [Sha10, JPB08]. Future work will focus on integrating each of these tasks within a common framework.

Bibliography

[AKRS06]   C. Amelunxen, A. Konigs, T. Rotschke, A. Schurr. MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. *Lecture Notes In Computer Science* 4066:361, 2006.

[AR04]   R. Alber, S. Rudolph. On a grammar-based design language that supports automated design generation and creativity. In *Fifth Workshop on Knowledge Intensive CAD*. P. 19. 2004.

[BKM88]   A. Brook, D. Kendrick, A. Meeraus. GAMS, a user's guide. *ACM SIGNUM Newsletter* 23(3-4):11, 1988.

[BSS07]   F. Bolognini, A. A. Seshia, A. K. Shea. A Computational Design Synthesis Method for MEMS Using COMSOL. In *COMSOL Users Conference*. 2007.

[CL02]   C. Chavez, C. Lucena. A metamodel for aspect-oriented modeling. In *Workshop on Aspect-Oriented Modeling with the UML at AOSD02*. 2002.

[Dir02]   R. Dirckze. Java Metadata Interface (JMI) Specification Version 1.0. *Unisys Corporation and Sun Microsystems* 2002, 2002.

[Est07]   J. A. Estefan. Survey of Model-Based Systems Engineering (MBSE) Methodologies. Technical report, California Institute of Technology, May 25 2007.

[Fis98]   J. Fisher. Model-Based Systems Engineering: A New Paradigm. *INCOSE INSIGHT* 1:3–16, 1998.

[FMS08]   S. Friedenthal, A. Moore, R. Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, 2008.

[FNTZ00]   T. Fischer, J. Niere, L. Torunski, A. Zündorf. Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In *Theory and Application of Graph Transformations, 6th International Workshop November 16-20, 1998*. Volume 1764, pp. 157–167. Springer, 2000.

[IKL⁺97]   J. Irwin, G. Kickzales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier. Aspect-oriented programming. *Proceedings of ECOOP, IEEE, Finland*, pp. 220–242, 1997.

[ISO05]   ISO/IEC. Unified Modeling Language Specification. 2005.

[Job08]   J. M. Jobe. *Multi-Aspect Component Models: Enabling the Reuse of Engineering Analysis Models in SysML*. Masters, Georgia Institute of Technology, 2008.

[JPB08]   T. Johnson, C. Paredis, R. Burkhart. Integrating Models and Simulations of Continuous Dynamics into SysML. In *6th International Modelica Conference*. Modelica Association, 2008.

[KP09]   A. A. Kerzhner, C. J. J. Paredis. Using Domain Specific Languages to Capture Design Synthesis Knowledge for Model-Based Systems Engineering. In *ASME IDETC & CIE*. 2009.

[MP07]   R. S. M. Bajaj, Peak, C. J. J. Paredis. Knowledge Composition For Efficient Analysis Problem Formulation Part 2: Approach And Analysis Meta-Model. In *ASME IDETC & CIE*. 2007.

[NoM]   NoMagic. MagicDraw. http://www.magicdraw.com.

[OMG08]   OMG. Systems Modeling Language v 1.1. 2008. http://www.omg.org/docs/formal/08-11-02.pdf

[PDSK01]   C. Paredis, A. Diaz-Calderon, R. Sinha, P. Khosla. Composable models for simulation-based design. *Engineering with Computers* 17(2):112–128, 2001.

[SA00]   A. Sage, J. Armstrong. *Introduction to systems engineering*. Wiley New York, 2000.

[Sah96]   N. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization* 8(2):201–205, 1996.

[Sha10]   A. Shah. *Combining mathematical programming and sysml for component sizing as applied to hydraulic systems*. Masters, Georgia Institute of Technology, 2010.

[SSS05]   A. C. Starling, T. Street, K. Shea. A parallel grammar for simulation-driven mechanical design synthesis. In *ASME IDETC*. Volume 2, pp. 24–28. 2005.