ABSTRACT
In this paper, a framework is defined for cost-effective reuse of engineering analysis models in the context of systems engineering. As systems engineering problems become increasingly complex and involve an increasing number of stakeholders, it is important that the knowledge required to solve these problems is formulated and communicated in a formal fashion. To allow such formal communication to occur in a cost-effective manner, a framework is introduced based on model reuse and composition. In this framework, engineering analysis models are associated with components and characterized by aspects. The resulting containers of related models are called Multi-Aspect Component Models (MAsCoMs). To allow the MAsCoMs to be incorporated seamlessly into systems engineering practice, the models are defined in the Systems Modeling Language (OMG SysML™). After introducing the basic modeling constructs, their implementation and composition is illustrated in the context of the design of a hydraulically actuated log splitter.

KEYWORDS
Model reuse; aspect; MAsCoM; SysML; systems engineering; composition; model characterization; modeling costs of reuse.

1. INTRODUCTION
Today's consumers seem to have an insatiable desire for increased integration and functionality creating a market that causes the complexity of new products and systems to increase rapidly. To manage this additional complexity effectively, systems engineers need to adapt the methods and tools they use in the systems development process. The increase in complexity affects this process by imposing the need:

- To integrate tightly across multiple disciplines: Electronics, mechanisms, controls, and software are often tightly integrated as in mechatronic systems;
- To coordinate closely among multiple stakeholders: Experts within the different disciplines and across different life-cycle phases need to combine their knowledge to achieve a competitive end-product;
- To weigh carefully the often conflicting objectives of all stakeholders: Trade-off decisions based on uncertain and incomplete information need to be made with respect to performance, cost, reliability, and other aspects.
- To manage effectively the large amount of information and knowledge involved throughout the lifecycle of the system: Cyber-infrastructure is needed to store, link, access, and maintain all this information and knowledge in an intuitive and consistent fashion.

To address these needs, the systems engineering community has started adopting a Model-Based Systems Engineering (MBSE) process [8, 10]. In MBSE, engineers formally model all aspects of a systems engineering problem, ranging from use-cases and requirements, to functional decompositions, physical
architectures and the corresponding behavioral analyses. The aspects are orthogonal directions along which a model can be characterized. This is similar to the aspects in Aspect-Oriented Software Development [35], or the different views in Computer Aided Multi-Paradigm Modeling [20].

By modeling these different system aspects formally, the different stakeholders can express their knowledge unambiguously and share that knowledge effectively and efficiently with other stakeholders. In addition, models of the different system aspects can be formally linked to each other so that the consequences of design changes can be more easily traced throughout the system in its multiple lifecycle phases, and so that analyses and decisions can be more easily revisited and updated.

However, formal modeling in MBSE also introduces additional costs. Capturing systems engineering knowledge formally in a model is nontrivial. It typically requires a higher level of expertise, additional time, and often the capture of information that would otherwise have been assumed implicitly.

It is therefore important to carefully weigh the costs of formal modeling versus its benefits. Whether this cost-benefit tradeoff favors formal modeling depends on the context. When designing a simple product or system in which the design team is small and the number and complexity of the models is small, one may not be able to justify the extra cost of capturing all this knowledge formally. However, for complex systems, the risk of not being formal is just too high — both the probability of something being overlooked and the consequences are large.

The goal of this paper is to shift the cost-benefit balance in favor of formal modeling by reducing the modeling costs. Our approach is based on knowledge reuse. By reusing the models, certain costs are incurred only once at the time the model is initially formulated and can then be amortized over multiple reuses of the model. In the remainder of this section, we argue that the potential benefit for reuse is large and that there are opportunities for promoting reuse beyond the levels applied in current practice. It is interesting to note that while model reuse can enable the cost effective generation of formal systems engineering models, model reuse itself must rely on formal modeling: One can only enable reuse by formally capturing the model, its characteristics, and the contexts in which it can be used.

The initial focus is on the reuse of analysis models. Analysis models are ubiquitous in current systems engineering practice; they are used for predicting the behavior of components and systems from different viewpoints. They are interesting from a reuse perspective because they can be reused not only from one design problem to the next, but also in multiple design iterations within a single design problem.

Consider the different tasks associated with engineering analysis. As is illustrated in Table 1, the costs and effort associated with several of the modeling and analysis activities can be reduced through reuse. For instance, model development requires deep insights into an application domain and with testing and verification can require a lot of time and effort. When reusing a model rather than developing a new one, one still needs to find and retrieve the model (e.g., from a model repository) and define the appropriate parameter values. But if sufficient context is included in the formal model definition, then these costs can be substantially smaller than when developing a completely new model.

Even more costly is model verification and validation. The process of constructing physical experiments, collecting data, and matching data to simulation results is time-consuming and expensive. Once a model has been validated in this fashion, it should be carefully protected and saved in a repository. Although it is wise to validate a model again whenever it is used in a new context, current validation and verification guidelines recommend that one verify and validate models for individual components and subsystems first before validating a system-level analyses in which these component models are used [1]. This fits perfectly with the approach introduced in this paper, where analysis models are formally organized into containers of models for reusable components or subsystems.

So far, we have argued that through formal modeling model reuse can be cost effective. However, formality by itself is not sufficient; it is also important that there be sufficient opportunity for reuse. A very specialized analysis model is unlikely to be reused because the chance that the same special design context presents itself again is small.

Therefore, the second pillar of our foundation to support model reuse is modularity. In a modular modeling approach, large models are decomposed into modular chunks that can be quickly and easily reused and configured into a large number of different system-level models. This fits well with current systems engineering practice, which relies on composition and integration to deal with complexity [4, 28]. By decomposing systems and their functions into sub-systems integrated with each other through well-defined interfaces, the systems engineering problem can be divided into smaller, less complex sub-problems, each of which can be solved by a smaller, more specialized design team.

Table 1. Costs of Modeling with Reuse.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Analysis 1 (development)</th>
<th>Analysis 2 (reuse)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formulate Modeling Task</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Develop Model</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Retrieve Model</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Define Model Parameters</td>
<td>X</td>
<td>partial</td>
</tr>
<tr>
<td>Verify Model</td>
<td>X</td>
<td>partial</td>
</tr>
<tr>
<td>Validate Model</td>
<td>X</td>
<td>partial</td>
</tr>
<tr>
<td>Simulate Model</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Since many systems require similar functionality, the subsystems satisfying these functions tend to be reused. For instance, many systems require mechanical energy and they rely on either internal combustion engines or electrical drives to provide this energy. In addition, the standardization of
components for modular design can produce greater product variety by reusing components across product variants and lines, and allows for easier validation and verification of the components [36]. We argue that because the components or subsystems are reused, the analysis models associated with these components should be reusable also. It may even be possible to compose certain types of component models into subsystem models in an automated fashion. Such automated composition is a future goal of this work, but is not presented in this paper.

In this paper, a framework is presented to support model reuse by establishing relationships between system design components, analysis models, and the many aspects of a model that pertain to analysis objectives, stakeholder perspectives, and other elements of model-based systems engineering. Within the framework, analysis models are associated with components and aspects so that their semantics of intended use are captured and represented for reuse. We define a model characterized within this framework as a “Multi-Aspect Component Model” (MAsCoM). A detailed overview of MAsCoMs is provided in Section 3.

The framework is implemented in the Systems Modeling Language (OMG SysMLTM) [33]. SysML is a formal language for describing systems for design and analysis purposes. It supports linking system design and analysis requirements with analysis models via meta-level constructs. The use of SysML to support MAsCoMs is described in more detail in Section 4, and is illustrated with an example in Section 5. Before we delve into the details, the relevant literature is first reviewed in the next section.

2. RELATED WORK

The reuse of modular design elements has been addressed by many. Baldwin and Clark [4] consider the use of a design structure matrix, task structure matrix, and modular operators to capture modularity in a design. Eppinger et al. [7] also consider that systems can be decomposed into modules, but note that some systems are integrative in nature. Integrative systems avoid the overhead of modular interfaces and can therefore achieve higher utilities [36] but are much less likely to have reusable elements. These systems are therefore not considered for the direct application of MAsCoMs. Gershenson et al. [11] take the perspective of modularity as it applies to the entire life-cycle of a product design. They claim that all components that are of the same modular form (based on function and interface) will undergo the same life-cycle processes. Using component trees to decompose structure, the level of the component being viewed and its level of abstraction have an effect on the view of the modularity of a process in the life-cycle. This also holds true for the selection of a modular equation model to predict the behavior of a piece of structure in a component tree. Although MAsCoMs are also mapped to component structures and processes (defined by aspects), such models of modules must still be stored for reuse.

The idea of reusing design knowledge by storing the knowledge in a repository has been proposed in the past. The NIST Design Repository [34] was one of the first efforts in this area. Further development of the knowledge representation underlying the NIST Repository resulted in the Core Product Model (CPM) [9]. The CPM is a high-level meta-model in which the core elements for representing products in design (i.e., form, function, and behavior) are identified and related to each other. The goal of the CPM is to provide a common foundation for product representation that can then be further refined as needed, e.g., for engineering analysis [2, 3], for manufacturing process planning, for functional decomposition [15, 32], or for assembly planning [26]. Similarly, the models developed in this paper follow the core relationships defined in the CPM, but refine them with more specific constructs for system behavior. Here, behavior is to be interpreted as any type of characteristic that can be predicted based on the form, distinguishable by many behavioral aspects, including function.

Both the CPM and this paper fit into a broader group of research efforts in which the goal is to define an ontology for design. An ontology is a formal data model for the concepts and the relationships between these concepts in a certain domain of discourse — the domain of design in this case. Most of the research in this area shares the perspective that at the foundation, one should distinguish between form, function and behavior. Examples include the work by Umeda et al. [37], Kitamura and Mizoguchi [29], and Horváth et al. [13]. However, system behavior has been the focus of investigation in only a few previous publications.

The most extensive previous research on characterizing behavior in engineering analyses was performed by Grosse and coauthors [12]. They organize the knowledge about engineering analyses models into an ontology, which includes both meta-data (e.g., author, documentation, etc. — similar to the Dublin Core [25]) and meta-knowledge, such as model idealizations and the corresponding justifications. A similar, although less extensive, meta-model for engineering analysis models has been developed by Mocko et al. [17].

In the current paper, this past work is expanded to enable reuse of engineering analyses in the context of large systems engineering efforts. In this respect, two extensions are important: First, the engineering analyses need to be related to the form (e.g., component geometry or system architecture) at a fine-grained level [23]. Second, the analysis models for components and subsystems must be formulated in a fashion that allows for composition so that a large number of different system topologies can be explored quickly [22].

Relating analysis models to form has been addressed previously in work on Design-Analysis Integration (DAI) [23]. Peak et al. relate the parameters of analysis models to parameters of design models in a declarative, reusable fashion using Constraint Objects (COBs) or, more recently, using SysML parametric diagrams [24]. In this paper, we adopt this approach but only at the level of individual components (see section on Fine-Grained Design-Analysis Relationships). By

2. RELATED WORK

The reuse of modular design elements has been addressed by many. Baldwin and Clark [4] consider the use of a design structure matrix, task structure matrix, and modular operators to capture modularity in a design. Eppinger et al. [7] also consider that systems can be decomposed into modules, but note that some systems are integrative in nature. Integrative systems avoid the overhead of modular interfaces and can therefore achieve higher utilities [36] but are much less likely to have reusable elements. These systems are therefore not considered for the direct application of MAsCoMs. Gershenson et al. [11] take the perspective of modularity as it applies to the entire life-cycle of a product design. They claim that all components that are of the same modular form (based on function and interface) will undergo the same life-cycle processes. Using component trees to decompose structure, the level of the component being viewed and its level of abstraction have an effect on the view of the modularity of a process in the life-cycle. This also holds true for the selection of a modular equation model to predict the behavior of a piece of structure in a component tree. Although MAsCoMs are also mapped to component structures and processes (defined by aspects), such models of modules must still be stored for reuse.

The idea of reusing design knowledge by storing the knowledge in a repository has been proposed in the past. The NIST Design Repository [34] was one of the first efforts in this area. Further development of the knowledge representation underlying the NIST Repository resulted in the Core Product Model (CPM) [9]. The CPM is a high-level meta-model in which the core elements for representing products in design (i.e., form, function, and behavior) are identified and related to each other. The goal of the CPM is to provide a common foundation for product representation that can then be further refined as needed, e.g., for engineering analysis [2, 3], for manufacturing process planning, for functional decomposition [15, 32], or for assembly planning [26]. Similarly, the models developed in this paper follow the core relationships defined in the CPM, but refine them with more specific constructs for system behavior. Here, behavior is to be interpreted as any type of characteristic that can be predicted based on the form, distinguishable by many behavioral aspects, including function.

Both the CPM and this paper fit into a broader group of research efforts in which the goal is to define an ontology for design. An ontology is a formal data model for the concepts and the relationships between these concepts in a certain domain of discourse — the domain of design in this case. Most of the research in this area shares the perspective that at the foundation, one should distinguish between form, function and behavior. Examples include the work by Umeda et al. [37], Kitamura and Mizoguchi [29], and Horváth et al. [13]. However, system behavior has been the focus of investigation in only a few previous publications.

The most extensive previous research on characterizing behavior in engineering analyses was performed by Grosse and coauthors [12]. They organize the knowledge about engineering analyses models into an ontology, which includes both meta-data (e.g., author, documentation, etc. — similar to the Dublin Core [25]) and meta-knowledge, such as model idealizations and the corresponding justifications. A similar, although less extensive, meta-model for engineering analysis models has been developed by Mocko et al. [17].

In the current paper, this past work is expanded to enable reuse of engineering analyses in the context of large systems engineering efforts. In this respect, two extensions are important: First, the engineering analyses need to be related to the form (e.g., component geometry or system architecture) at a fine-grained level [23]. Second, the analysis models for components and subsystems must be formulated in a fashion that allows for composition so that a large number of different system topologies can be explored quickly [22].

Relating analysis models to form has been addressed previously in work on Design-Analysis Integration (DAI) [23]. Peak et al. relate the parameters of analysis models to parameters of design models in a declarative, reusable fashion using Constraint Objects (COBs) or, more recently, using SysML parametric diagrams [24]. In this paper, we adopt this approach but only at the level of individual components (see section on Fine-Grained Design-Analysis Relationships). By
establishing the relationships between design and analysis models at the component level, the relationships are maintained even when the components are composed into larger systems, thus further promoting model reuse. To enable composition, additional knowledge is needed both about the model interfaces and about the composition process, as is further explained in Section 5. Wallace et al. [38] also consider composable models. They note that a modular, composable analysis approach allows multi-disciplinary problems to be broken down into modules that can be assigned to specialized teams—a benefit of modularity that is also exploited by MAsCoMs.

3. MULTI-ASPECT COMPONENT MODELS

As was argued in Section 1, to be cost-effective, model-based systems engineering must rely on model reuse. In this section, we develop a framework for enabling such model reuse by relying on modularity and composition.

The structure of MAsCoMs

Since current practice in systems design relies mostly on integration of modular components and subsystems, the most common units for reuse are exactly these components or subsystems. It therefore makes sense to organize the corresponding analysis models by component type also. Whenever a designer decides to use a particular component, he or she will immediately be able to identify all the analysis models that have been previously used to analyze that component or describe its behavior in a larger system. As illustrated in Figure 1, the components themselves are organized in a taxonomy so that the user can easily browse from general classes down to very specific instances of components. At each level, the component model is linked to all the relevant engineering analysis models.

However, the number of such models could be very large, so that an additional method of organization is desirable. To facilitate the task of selecting and composing analysis models further, we propose to characterize the analysis models based on one or more aspects, as is illustrated conceptually in Figure 1. In Aspect-Oriented Software Development [35] modularity is achieved by implementing cross-cutting concerns separately so that they can be woven into a variety of different software classes. In the context of modeling, rather than weaving models together, what is important is that we can identify which models are compatible with each other so that they can be composed into system-level models. To be compatible, models must characterize the components in a system from a similar perspective, in a compatible mathematical formalism and in the same executable language. By using a formal taxonomy of aspects, the semantics of the individual analysis models are defined in a computer interpretable and searchable fashion.

In the remainder of this section, the details are provided for how analysis models are organized into MAsCoMs. In addition to discussing taxonomies of components and aspects, it is explained in detail how the analysis models are tightly linked to each other through components at a very fine-grained level.

A Taxonomy of Components

In design, components or subsystem are selected and defined in an iterative fashion. First, a functional architecture is defined after which functions are assigned to components in a physical architecture [27] (or, equivalently working principles and working structures are identified [21]). The focus is initially on the selection of broad classes of components that share the same functionality. For instance, to implement the function of converting electrical to mechanical energy, the broad class of motors could be identified. In subsequent iterations, this broad class of components is gradually refined until a particular component XYZ from company ABC has been identified. At each step along the way, analysis models at different levels of abstraction are used. As the definition of the components still under consideration becomes more and more detailed, the corresponding analysis models also need to become more detailed such that the selection can continue to be narrowed down further.

To support such successive refinement of classes of components down to very specific individual components, it is meaningful to organize the components in a taxonomy; One branch of the total taxonomy—the branch of hydraulic components—is illustrated in Figure 2. This taxonomy is based on the E-class classification hierarchy [6]. Organizing components into a taxonomy has the additional benefit that one
can take advantage of the inheritance mechanism to efficiently associate analysis models with components. In the taxonomy, analysis models associated with parents apply also to children. For instance, since an axial piston pump is a type of displacement pump, the models for the general class of displacement pumps (the parent) also apply to axial piston pumps (the child). However, often, more detailed models are available for the children because more detailed knowledge is available about their structure, size, or other design properties.

Each of the nodes in the taxonomic tree corresponds to a model that defines the key characteristics of the component or class of components, as is illustrated in Figure 3; we call this a structure model. The structure models are parametric—they contain properties identifying key characteristics of the component: sizing properties, key performance parameters, as well as the intended interface of the component (i.e., the locations or ports at which the component is intended to interact with other components in a system [16]). For instance, a pump may be characterized by sizing parameters that include displacement, mass, or maximum pressure rating; by key performance parameters such as cost, efficiency or reliability; and by an intended interface consisting of two fluid ports (suction and discharge) and two mechanical ports (input shaft and housing).

The structure models are central to MAsCoMs—they serve as the central entry-points for accessing all the engineering analysis models associated with the components. The analysis models in turn define how the performance parameters in the structure model relate to the sizing properties. To facilitate maintaining consistency among all these parameters, the analysis models are tied to the structure model at a very fine-grained level as is explained further later in this section.

A Taxonomy of Aspects

When reusing a model, one needs to recognize which model is needed from among the many models that may be associated with a particular component. To help the designer do this, models are characterized using aspects. Since there are a large number of potential aspects, it is helpful to organize them also in a taxonomy, as is illustrated in Figure 4. The taxonomy also emphasizes that the aspects represent independent directions along which a model can be characterized. As a result, a model is typically characterized by multiple aspects simultaneously. For example, a pump model could be characterized simultaneously by the hydraulic and mechanical engineering disciplines, by the continuous time discretization aspect, by the DAE mathematical formalism, and by the Modelica representation syntax.

These aspects formally characterize a model and thus succinctly provide the designer or analyst with the basic information needed to select an appropriate model. Additional information about the model can be defined as meta-data that is less structured, such as model documentation, development history, or prior usage scenarios. Based on the aspects, a designer can be efficiently search or browse through a model.
repository to identify the model that is most appropriate for a particular design context. In addition, when composing multiple component models into a system-level model, the aspects provide necessary information to determine compatibility between models. For instance, to be composed, models need to be expressed in compatible mathematical formalisms and levels of discretization—it is not meaningful to combine a discrete event simulation model with a partial differential equation model. Models that are composed also need to share compatible engineering disciplines. One set of models may describe the hydraulic behavior of a system while another may describe its mechanical structure. Having formal representations of these different aspects available is particularly important when (partially) automating the composition process—one of the future goals of this research.

Fine-grained Design-Analysis Relationships

While the characterization of analysis models using the component and aspect taxonomies reduces the cost of identifying appropriate models for reuse, it does not affect the cost of instantiating these model in a specific design context. One of the goals of MAsCoMs is to facilitate (and maybe automate) this instantiation of analysis models into a system-level analysis model.

In a variety of engineering disciplines, it is common to describe systems as compositions of components in a schematic diagram (e.g., see Figures Figure 7 and Figure 9). One can interpret such diagrams as compositions of structure models (as defined previously in this section) connected to each other at their ports (intended interface locations). Assume that a system schematic is available in which specific structure models for individual components have been configured into a system by connecting their ports. Is it then possible to instantiate the corresponding analysis models and configure them into a system-level simulation? The additional knowledge necessary to support this context-specific instantiation can be incorporated in MAsCoMs with two additional diagrams: parameter maps and junction maps.

Parameter maps bind the parameter values in analysis models to the related parameters in the corresponding component’s structure model. In the context of systems engineering, the values for the parameters need to be related to the properties of the system alternative that is currently being analyzed. Since we have associated the analysis models with components in the component taxonomy, it becomes possible to establish these relationships also in a reusable fashion. How this is accomplished using SysML parametric diagrams is explained in Section 4.

In addition to parameter maps, MAsCoMs also include interface maps. Interface maps support the configuration of analysis models for individual components into system-level analysis models. Similar to the composition of structure models into a system schematic, analysis models can be configured into networks through well-defined port-based interfaces [22], as is implemented in tools such as Simulink™ [31], and in languages such as Modelica [18]. Recently, the ability to compose analysis models has even become feasible for finite element models [3, 30]. In order to configure the analysis models, one needs to define how the ports of the analysis models relate to the ports in the structure models. This is accomplished through interface maps as is further explained in the next section.
4. IMPLEMENTING THE FRAMEWORK IN SYSML

To make the MAsCoMs outlined in Section 3 useful in the context of systems engineering, all the concepts and relationships have been defined in the Systems Modeling Language (OMG SysML™) [33]. Since SysML has been defined specifically to support systems engineering, it includes modeling constructs that directly support the definition of physical architectures and engineering analyses—the main focus of MAsCoMs.

The primary modeling construct in SysML is the block. A block can represent anything, whether tangible or intangible, that describes a system. For instance, a block could model a system, process, function, or context. In this paper, the use of blocks includes the modeling of component structure, aspects, engineering analysis models, and interface junctions. Blocks are declared in Block Definition Diagrams (BDD). A BDD is used to define block features and the relationships between blocks or other SysML constructs and is thus the equivalent of a class diagram in UML [5]. For instance, in Figure 3, the properties of specific component structure models are defined, and the structure models are related to each other through specialization relationships—the white arrows. A large variety of other relationships that reach beyond just the definition of blocks are included in Internal Block Diagrams (IBD). For example, in Figure 9, multiple component-structure models are composed into a hydraulic system. Parametric Diagrams (PAR) allow one to express mathematical constraints between block properties.

In this paper, it is shown how these existing modeling constructs can be used to implement reusable libraries of engineering analyses. As is illustrated in Figure 1, the analysis models in MAsCoMs are organized in a matrix structure, each linked to a component-structure model and characterized by multiple aspects. These relationships are captured in the SysML language in multiple diagrams: BDDs for the component and aspect taxonomies, a model-context BDD in which a specific analysis model is linked to these taxonomies, and a corresponding parametric diagram in which the fine-grained design-analysis relationships are established.

By defining such diagrams for a large number of analysis models, a library of formal, reusable models can be defined to capture the engineering analysis in a particular domain of interest. These libraries combined with existing SysML constructs for requirements, use-cases, functional allocations, system behavior, and test-cases provide the systems engineer with a complete language and vocabulary for efficiently and effectively defining and evaluating system alternatives in a formal fashion.

Taxonomies of Components and Aspects

Both the taxonomies are modeled in SysML using the generalization relationship, as is illustrated Figure 3. A generalization (a.k.a., inheritance relationship) signifies that all the properties of the parent block—the block pointed to by the white arrow—are inherited by the child block. Defining the taxonomy of components in this fashion simplifies the definition of additional components because most of their properties are likely to be inherited from existing component definitions. As is illustrated for the Vendor OTS Pump, SysML also allows one to further restrict the values of inherited properties. Finally, besides certain key sizing and performance properties, the blocks also define the intended interface of the component, e.g., the suction and discharge ports of the pump.

An important additional benefit of using generalization relationships is that all the engineering-analysis models associated with a parent also apply to its children. For instance, when defining an additional pump from a specific vendor, there is no need to associate explicitly an entire set of analysis models with this new structure model, because the specific pump can simply be a specialization of an existing pump model and, as such, inherit all the analysis models associated with all its parents.

To help the user browse through the set of component models, the blocks are organized in packages, as is illustrated in Figure 2 and Figure 4. This has the additional advantage that name clashes can be easily avoided because they only need to be unique within the namespace of the local package. Globally, name clashes are avoided by using fully qualified names (e.g., Component.HydraulicComponent.Actuator.Cylinder rather than Cylinder).

Model Contexts

To describe how a specific analysis model relates to a component structure model, a Model Context is defined, as illustrated in Figure 5. For each matching pair of specific analysis model and component structure, a different Model Context is needed. The idea of mapping analysis models to structure models in a specific context was developed previously by Peak et al. [24]. They introduced Context Based Analysis Models (CBAM) to bind the parameters of an analysis model to values in a structural model in the context of a specific analysis. If the analysis model is defined to be sufficiently general, it can be reused in multiple contexts. In this paper, it is recognized that, for a particular component, such bindings between analysis models and structure models often remain the same irrespective of how the component is used within a larger system. It therefore makes sense to establish these bindings at the component level so that the mapping becomes reusable.

To relate an analysis model to the elements in the component and aspect taxonomies, the SysML relationship «refine» is used. For instance, in the «refine» relationships in Figure 5 reflect that the ConPump analysis model refines the description of the Fixed Displacement Pump component and that it refines a generic hydraulic behavior model, a mechanical rotational model, etc. Note that as with most SysML diagrams, only the relevant information is shown in Figure 5. One must keep in mind that the component is related to many other components...
in the component taxonomy and that the aspects are also just references to their definitions in the aspect taxonomy.

Parameter Maps and Junction Maps

Now that the analysis model is linked to its aspects and to a corresponding component, the detailed parameters of the model can also be mapped in a reusable fashion.

As shown in Figure 6, the parameters of an analysis model can be bound to their corresponding properties in the component-structure model. The binding connector has the semantics of a noncausal equality. If necessary, additional constraint blocks can be used to bind properties that are related but exactly equal. For instance, the displacement property of the ConPump model is related to the displacement property of the Fixed_Displacement_Pump component through a constraint block that imposes the appropriate unit conversion. In addition to unit conversions, a similar constraint block could be used to map related properties to each other, such as radius to diameter or radius to surface area.

To support composition of port-based analysis models [19], the Model Context in Figure 5 also includes a detailed junction mapping. By formally linking junctions in the analysis model (e.g., p: FluidJunction) to the corresponding ports in the structural model (e.g., discharge: FluidPort), the component-level analysis models can be composed into a system-level analysis model based on the composition of component-structure models in a system configuration model. Although we have not yet implemented this composition process in an automated fashion, it is illustrated conceptually in the next section.

A final comment related to parameter and junction maps revisits the question of why they are necessary. One could have used other mechanisms for linking analysis models to component-structure models. For instance, one could have relied on the inheritance mechanism associate analysis equations with the properties in a component-structure model. However, that would require that the model equations be expressed using the same property names as used in the component-structure model. Since it is often the case that one analysis model is associated with multiple component-structure models, and that one component-structure model is associated with multiple analysis models, it would become nearly impossible to develop a reusable model library in which all the property names remain consistent across both analysis and component-structure models. The mechanism of mapping parameters and junctions in a Model Context provides the needed flexibility to define modular, reusable analysis models independently of the components with which they may be associated in the future.

5. USING MASComS FOR SYSTEMS DESIGN: A HYDRAULIC LOG SPLITTER EXAMPLE

In this section, the hydraulic system of a log splitter serves to illustrate how MAsCoMs can be used in the design process. Although a log splitter is relatively simple, it is representative for a broad class of hydraulic devices. As illustrated in Figure 7, its hydraulic circuit contains a flow device (shaft-driven pump), a servo-valve for flow control, a hydraulic actuation device (double-acting cylinder), a filter and hydraulic lines. Larger hydraulic systems can be thought of as variants of this circuit with additional actuators or more complex control valves.

For this example, assume that the designer has previously defined a particular design problem by modeling the system
objectives, requirements and functional decomposition in SysML. The designer then needs to consider which measures of effectiveness (MOE) can best be used to predict the extent to which certain objectives are satisfied. This is where analysis models play a role. The measures of effectiveness need to be chosen such that they can be predicted based on an analysis model. As is illustrated in Figure 8, the analyst may characterize the context of the analysis by specifying which measures of effectiveness need to be predicted and by defining the particular aspects that need to be considered in the system-level model that is used for this prediction. The resulting simulation can then be used in a SysML test-case to verify whether the requirement for the given measure of effectiveness is satisfied.

Notice that in Figure 8, the block for the Hydraulic System model is still empty. In the remainder of this section, the process for filling in this block is explained. The creation of this system-level analysis model starts by defining the particular system architecture that will be analyzed, as proposed in Figure 7 and formalized in SysML in Figure 9. The system architecture is a composition of component-structure models connected by their interface ports. Depending on how far the design process has progressed, these component-structure models could still be very abstract (i.e., high-up in the component taxonomy) or very specific (e.g., a specific pump from a specific manufacturer). Throughout the design process, these component-structure models are likely to be refined into more and more specific models from the component taxonomy.

If a particular component-structure is not yet available in the component taxonomy, then the user may need to create a new model. Such a new model can be defined most easily by determining where in the taxonomy it would fit and by then extending the appropriate parent models through specialization relationships. In this way, all the analysis models of the parents are also automatically associated with the new model. If additional analysis models are required then they can be added by defining additional Model Context diagrams. Note that such additional models should be defined in a local user-model rather than added to the MAsCoMs library right away; since the library is likely to be (re-)used by many different users, it should be kept under strict version control, and models should only be added to the library after extensive verification and validation.
Once the system architecture has been defined, one can use the Model Context diagrams in the MAsCoMs library provide the necessary information to identify the appropriate analysis models. Although there are potentially a large number of analysis models associated with each component in the taxonomy, the aspects that characterize the models allow the designer to home in on the few that are applicable in the given context. To be applicable, a model needs to include the same aspects as have been defined for the system-level model (as in Figure 8). The aspects also help the designer to determine whether the component models are compatible with each other. Once the appropriate models have been determined, the specific values of the model properties can be instantiate through the use of the parameter maps.

The final step towards a complete system-level model is to compose the analysis models of the individual components into a system-level model. This composition requires additional knowledge beyond what is currently available in the MAsCoM library. This knowledge is algorithmic in nature—it cannot be captured in a static diagram, but instead requires the specification of how the diagrams need to be manipulated or transformed. In the current implementation, this composition is left to the user. However, in the future, we plan to automate this composition process through the use of graph transformations as has already been demonstrated for SysML diagrams by Johnson et al. [14].

The composition process is illustrated Figure 10. Although the topology of the analysis model is very similar to the topology of the system-structure model (see Figure 9), it is not a one-to-one mapping. As is explained in more detail in [14], the connection of energy-based ports, such as a FluidJunction, requires the inclusion of a model representing the equivalent of Kirchhoff’s voltage and current laws. In Figure 11, a different composition is illustrated for reliability models. Since reliability models are not based on energy exchange through ports, their topology does not match the topology of the system-structure model. Instead, the composition corresponds to tying all component models into a system-level OR node. The definition of composition algorithms in terms of graph transformation algorithms is left for future research.

6. REUSE OF MASCOMS

The main reason for modeling the relationships between component models in MAsCoMs is to promote reuse. The opportunity for reuse exists within the context of a single design problem whenever two system alternatives are considered that share similar components or subsystems—a very common occurrence. Additionally, when solving different design problems but still within the same application domain reuse is often possible.

For instance, consider the design of the hydraulic system of a scissor lift. Although a scissor lift is quite different from a log splitter, it does share the need for compact, large-force actuation for which hydraulic components are well-suited. The schematic for a possible hydraulic system alternative is illustrated in Figure 12 as a system structure-model. This design shares the same power-subsystem as the log splitter shown in Figure 10, reusing all but two component models. To see the difference between the system models for the log splitter and scissor lift, we focus on the actuation subsystem portion of their design structure models. This difference can be seen by comparing Figure 13 and Figure 14, which are composed from Figures Figure 9 and Figure 12, respectively.
Note that the fact that the two systems use similar components does not guarantee that the corresponding models can be reused. The level of abstraction and the assumptions that are made in both analyses need to match also.

Figure 12. System structure-model for the scissor lift.

Figure 13. Dynamic behavior model for the actuation subsystem portion of the log splitter example.

Figure 14. Dynamic behavior model for the actuation subsystem portion of the scissor lift example.

7. SUMMARY

This paper presents a framework for characterizing and reusing analysis models in model-based systems engineering. Analysis models are organized into Multi-Aspect Component Models—collections of analysis models formally linked to a particular component-structure model and formally characterized by multiple aspects in an aspect taxonomy. By formally organizing the analysis models into MAsCoMs, much of the knowledge necessary to instantiate and compose system-level analysis models is captured and available for reuse.

The MAsCoMs have been defined in SysML so that they can be easily used to support decision making in systems engineering. Through reuse, the additional costs associated with formal modeling in MBSE can be amortized so that the benefits of formal modeling can be made available cost-effectively to even small systems engineering efforts.

8. ACKNOWLEDGEMENTS

This work has been funded by the ERC for Compact and Efficient Fluid Power, supported by the National Science Foundation under Grant No. EEC-0540834. Additional financial support was provided by Deere & Company and by Lockheed Martin; No Magic, Inc. provided access to its MagicDraw UML/SysML software tool. The authors would also like to thank Roger Burkhart, Sanford Friedenthal, Leon McGinnis and Russell Peak for the discussions that helped crystallize the ideas presented in this paper.

9. REFERENCES


