Maintaining the Design Intent in the Synthesis of 3D and 1D System Models using Constraints

Arquimedes Canedo\textsuperscript{1} Zhi Zhang\textsuperscript{2} Ahmad Albaqsami\textsuperscript{3} Student Member, IEEE, Jiang Wan\textsuperscript{4} Student Member, IEEE, and Mohammad Abdullah Al Faruque\textsuperscript{5} Member, IEEE,

Abstract—Systems design consists of both geometry-based three-dimensional (3D) and energy-based one-dimensional (1D) models. Automatically synthesizing 1D and 3D models from the same specification is useful because it saves time and eliminates errors. State-of-the-art synthesis algorithms use a one-to-one mapping of 3D to 1D artifacts and therefore only cover the geometric aspects of the design. In this paper, we propose an approach where constraints are abstracted to the functional level and integrated with the functional models thus providing for efficient 3D to 1D mapping and allowing for 1D to 3D mapping. Additionally, we present our work in the context of Product Lifecycle Management (PLM) systems and demonstrate the novel capabilities of our software using a real-world satellite and excavator design examples.

I. INTRODUCTION

A single ship, airplane, or car design consists of thousands of high-fidelity models created using domain-specific mechanical engineering Computer-Aided Design (CAD) tools that contain precise information for manufacturing. Many of these geometric models are 3D in nature. Similarly, there are electrical, software, control, and other models created with domain-specific tools. Unfortunately, the high-level of specialization and domain-dependency makes information exchange among these tools difficult. Currently, a technology growing in popularity is the so called 1D simulation. 1D simulation tools such as Modelica \textsuperscript{1} and LMS Amesim \textsuperscript{2} combine multiple disciplines in a system-level model. This is possible because the disciplines are integrated via energy conservation principles \textsuperscript{3}. Using the 1D tools, an entire system can be analyzed holistically using energy transformation and conservation principles.

An important problem is the automatic mapping between 3D models and the equivalent 1D models. Although the manual creation of one kind of model from another is possible, it is time consuming, error prone, and tedious. Several approaches have been proposed to automatically generate 1D models from 3D CAD information \textsuperscript{4}, \textsuperscript{5}, \textsuperscript{6}, \textsuperscript{7} in a process referred to as “3D-to-1D” synthesis. These implementations exploit the redundancies in multibody systems to directly map bodies, joints, drivers, and parameters (e.g., mass, length, center of inertia, forces) in 3D models to equivalent components and some of their parameters in the 1D world. While obtaining the equivalent mechanical structure automatically is a good starting point, the rest of the disciplines must be modeled manually. A related problem is the creation of 3D models from 1D models. This inverse process is much more complex than creating 1D models from 3D models because there is less and often orthogonal information in 1D models. Therefore, it is necessary to include semantics in the process. To the best of our knowledge, we are the first to demonstrate the automatic synthesis of 3D models from 1D models in a process referred to as “1D-to-3D” synthesis.

In this paper, we introduce a constraint-based functional modeling approach to harmonize high-fidelity 3D and 1D models. The key observation of our approach is that 3D and 1D models created during the detail design phase have a common origin in functional models created earlier during the concept design phase that capture the design intent explicitly. Besides, the universal constraint laws on the functional properties or parameters apply to all models implementing the same functions. All these make it possible for us to express model constraints and build model relationships at the functional level, which can be further enforced for automatic model generations. For the first time, we demonstrate that 3D-to-1D automatic model generation can be extended beyond mechanical aspects and can be contextualized to include other aspects such as electrical, thermal, and control. Functional information can be used to enhance 3D-to-1D model generation, and to synchronize high-fidelity models and maintain a consistent set of product information across tools and engineering disciplines. At the same time, we demonstrate that the model constraint information built within functional models enables the automatic 1D-to-3D model generation. The novel contributions of this paper are as follows:

- A new algorithm that significantly improves the capabilities of automatic synthesis of 3D-to-1D models. The key innovation is the use of functional models during synthesis to infer the use of non-mechanical components interacting with mechanical components.
- An extension to the functional models with a fully associative constraint language for maintaining properties consistent across models during their lifecycle.
- An implementation of the constraint solver that tracks the changes to the instances of constraint variables, inter-

---

\textsuperscript{1}Arquimedes Canedo is with the Siemens Corporation, Corporate Technology, Princeton, NJ, USA.
\textsuperscript{2}Zhi Zhang is with the Department of Computing and Information Sciences, Kansas State University, Manhattan, KS, 66502, USA.
\textsuperscript{3}Ahmad Albaqsami is with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, 92617, USA.
\textsuperscript{4}Jiang Wan is with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, 92617, USA.
\textsuperscript{5}Mohammad Abdullah Al Faruque is with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, 92617, USA.
In [5], bidirectional translation and model synchronization is the other way around – from system-level models to 3D CAD. They allow 3D CAD to system-level generation but not the data (e.g., geometry for rendering in STL file format [4]).

Some data (e.g., dimensions, mass, center of inertia, orientation) focus exclusively on the extraction of geometric/mechanical nature of 3D CAD models, all these translators translate 3D CAD models to Matlab/Simulink/Simscape [10]. However, due to a CATIA to Modelica translator [5], and generic 3D CAD translators the problem of 3D-to-1D synthesis has been the focus of prior research. Several researchers have proposed tools that can be generated from the SysML4Modelica models [13]. The remaining 1D components are sources and sinks that can also be specified as rules. Note that the synthesized 1D model is an equivalent mechanical representation of the 3D model. Further extending the model to other disciplines requires manual effort.

This paper is organized as follows. Section II provides the technical background and presents our contribution relative to the related work. Section III presents our algorithm for 3D-to-1D synthesis and synchronization through the functional models. Section IV defines our fully associative constraint language and the constraint solver that is used for 1D-to-3D model synthesis and synchronization. Section V evaluates our system in two real-world examples: a satellite, and an excavator. Section VI provides our final remarks, and sets the direction for future work.

II. BACKGROUND AND RELATED WORK

The tight integration of computation with multiple physical elements in a system requires components to be designed and tested as part of a larger system (and not in isolation). Although system-level design tools such as Amesim [2], Matlab/Simulink [8], LabVIEW [9], and Modelica [1] are being integrated to PLM systems and adopted by industry, there are technical challenges that need to be addressed. Over the years, many companies have accumulated knowledge and intellectual property in the form of high-fidelity 3D CAD models that contain precise instructions for fabrication. Unfortunately, 3D CAD models are not readily compatible with system-level design tools. This forces engineers to manually analyze, abstract, and create new system-level models. This not only increases the number of models to be maintained by the domain experts but also presents the challenge of maintaining consistency across models. Given that a single car, airplane, or ship consists of tens of thousands of 3D CAD models, it is critical to automate the process of mapping 3D to system models, and vice versa, in order to help companies to leverage their accumulated corporate know-how.

The problem of 3D-to-1D synthesis has been the focus of prior research. Several researchers have proposed tools that translate 3D CAD models to system-level models including an Autodesk Mechanical Desktop to Modelica translator [4], a CATIA to Modelica translator [5], and generic 3D CAD models to Matlab/Simulink/Simscape [10]. However, due to the geometric nature of 3D CAD models, all these translators focus exclusively on the extraction of geometric/mechanical data (e.g., dimensions, mass, center of inertia, orientation), kinematic data (e.g., rigid bodies and joints), and visualization data (e.g., geometry for rendering in STL file format [4]).

Except for [5], all these translators are unidirectional because they allow 3D CAD to system-level generation but not the other way around – from system-level models to 3D CAD. In [5], bidirectional translation and model synchronization is realized by associating system-level models with 3D CAD models using a graphical user interface. An important aspect considered by the authors is the need to maintain a high design process efficiency in the presence of multiple engineers and designers in a PLM system. Compared to these tools, our approach is more flexible because it uses the design intent for harmonization – rather than a GUI or brittle mapping rules – and it extends the synthesis beyond mechanical aspects.

We illustrate the limitations of the state-of-the-art tools with the example of a satellite shown in Figure 1. The rules specify how 3D components map to 1D components; see how \( \{1, 2, 3, 4\} \) in the 3D satellite model map to 1D Amesim model. The remaining 1D components are sources and sinks that can also be specified as rules. Note that the synthesized 1D model is an equivalent mechanical representation of the 3D model. Further extending the model to other disciplines requires manual effort.

The inverse process, i.e., 1D-to-3D, has not been explored in the literature. One reason may be that 1D models, although multi-disciplinary, provide less detail than 3D models. Therefore, this may be seen as a useless capability. However, our work shows that function-structure relationships can be encoded in constraints and exploited by a PLM system to enable 1D-to-3D synthesis of models while maintaining parameter consistency.

Researchers in [11] have proposed the use of SysML to define both models of possible components and possible system architectures. Furthermore, they combine SysML/UML modeling techniques with Open Modelica [12] simulation tool for a unified CPS modeling environment. The same researchers have defined the SysML4Modelica model, which is an extension of SysML, to model the system level requirements, behaviors and constraints. They demonstrated that 1D Modelica models can be generated from the SysML4Modelica models [13]. The language ModelicaML [14] has very similar goals and has demonstrated similar capabilities.

The authors in [15] propose a design flow for CPS. It starts with using the SysML as a modeling language for specifying system requirements and constraints by the developers. Then, a tool supports the automatic translation from SysML to AADL models for analysis. In [16], the authors investigate the design of energy efficient buildings using SysML models. Design constraints are formally defined as design rules, which use the same syntax as modeling system functions in the
SysML models during the design time. Another relevant tool, the OpenMETA tool chain, integrates models and tools from multiple domains into a single model integration language CyPhyML. As a result, cross-domain design requirements and constraints are formulated and can be evaluated and verified together [17]. While we are not the first to use constraints to synchronize models, we are the first to use constraints in combination with functional models to harmonize 3D and 1D tools.

### A. Key Improvements Relative to Prior Art

Figure 2 summarizes the key improvements of our method relative to prior art. It also shows the newly introduced steps, both manual and automatic, and provides qualitative measures of how the different design workflows are improved relative to time (hours, minutes, seconds). Figure 2(a) shows three modeling workflows that are possible with existing methods. The most common workflow starts by creating a 3D model (hours), mapping the 3D structures to 1D structures with hard coded rules, and automatically generating the 1D model using methods similar to [4], [5], [10]. The problem with this approach is the rigidity imposed by the hard-coded rules. The other two workflows are less common but relevant to this paper. One starts with an update to the automatically generated 1D model (minutes), the propagation of changes to the 3D model by an expert (minutes), and the update of the 3D model (minutes). The second workflow starts with an update to the 3D model (minutes), the propagation of changes to the 1D model by an expert (minutes), and the update of the 1D model (minutes). Note that these two workflows are entirely manual because it needs an expert who understands both the semantics of the 1D and the 3D models. At best, these workflows take several minutes every time there is a change.

Our method, as shown in Figure 2(b), introduces a few manual steps related to the creation of the functional model, and the mapping of functions to 3D structures. However, we demonstrate that these new manual steps can be streamlined via a tool-specific implementation and in practice only take a few minutes to a few seconds to realize. Our first workflow requires the manual creation of a functional model (see Section III) and constraints (see Section IV). After the 3D model is created (hours), it also requires the manual association of functions to 3D structures (see Section III.B) that typically takes a few minutes. The rest is automatic: constraints are solved (seconds), and an equivalent 1D model is generated by searching a database of compatible 1D components (see synthesis algorithm in Section III.A). The major difference of our method against existing methods is that functionality drives the synthesis process instead of having hard-coded rules.

Our second workflow shows the benefits of our functional modeling approach. A manual update on the 3D model can be automatically propagated and updates the 1D model using the constraint solving in just a few seconds. Our third workflow shows a novel capability where a manual update on the 1D model is automatically propagated and updates the 3D model using the constraint solving in just a few seconds. These two workflows are experimentally demonstrated in Section V. Our fourth workflow is expected to be less common, but changes to the constraints, rather than on either of the models are automatically propagated via the constraint solving to the two models in just a few seconds.

In summary, our work introduces a few new manual steps, but is able to automate many of the processes required for model harmonization. More importantly, it provides a more intuitive approach for modeling based on functions, rather than on tool-specific and domain-specific structures.

![Figure 2](image1.png)

**Fig. 2.** Qualitative comparison of our approach versus the prior art.

### III. Functional Approach of 3D-to-1D Synthesis and Synchronization

Functional models describe the design intent – what the system does – in a multi-disciplinary yet implementation-independent manner [18], [19], [20], [21]. Functional models are a formal representation of requirements that engineers from different disciplines use to communicate effectively. This section presents how our approach leverages functional models to improve the capabilities of 3D-to-1D synthesis, and its integration to a PLM workflow where different domain engineers can synchronize their designs using functions.

A functional model is a labeled directed graph $F = (V, E, s, t, L_v, L_e)$, where nodes $v_i, v_j \in V$, are connected by edges $e_{(i,j)} \in E$. Each node $v_i$ is a function and each edge $e_{(i,j)}$ is a flow from function $v_i$ to function $v_j$. $s : E \rightarrow V$ and $t : E \rightarrow V$ are source and target mappings. $l(v) \in L_v$ and $l(e) \in L_e$ are unique labels, per $v \in V$ and $e \in E$. 

$$
A = \begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
$$
Figure 3 shows a satellite’s functional model consisting of 19 labeled functions such as Store EE, Regulate RME, etc. In this paper, we use functional models stored as Visio diagrams. This format allows users to rapidly generate diagrams, functions, connections between functions, and to label the functional models. For elementary functions (e.g., Store, Convert) we use the Functional Basis language [21], and for higher-level functions we use user-defined labels (e.g., Align Telescope to the Sun). These functional models are then parsed by C# code that extracts the topology and properties needed as inputs to our algorithm. In this paper, we encode functional requirements $R$ as constraints (see Section IV). Although it is out of the scope of this paper, an alternative implementation of functional models $F$ and functional requirements $R$ could be implemented in SysML [22], [23], [24].

A. Synthesis Algorithm

Our algorithm, shown in Figure 4, indirectly associates the 3D with the 1D structure through single or multiple functions. Initially (Step 1), the algorithm parses the input functional model to produce a functional model graph (See Figure 3). The algorithm also parses an input 3D model and produces a 3D model graph (Step 2). The nodes in a 3D graph represent 3D artifacts such as rigid or flexible bodies, joints, drivers, sensors, and actuators. The edges represent the relationships between the artifacts. The attributes of these artifacts (e.g., joint type, forces, tolerances) are also encoded in the 3D model graph. It is important to note that the input functional model broadly captures the design intent of the system, and the input 3D model mostly captures the geometric and mechanical structure and constraints of the system. Then, each function $f \in F$ in the functional model graph and artifacts $a \in A$ in the 3D model graph are mapped $M = F \rightarrow A$ (Step 3). Note that $M$ can be a partial mapping rather than a complete mapping [25], [26]. The algorithm then finds all the 1D components in a 1D model library whose physical structure (port types, mathematical and physical equations) and functionality match the elements $m \in M$ (Step 4). The 1D Model Library includes the functional classification of 1D components, and can be queried for the set of functions that a component fulfills [27]. There are two possibilities with the queries (Step 5): (a) that multiple 1D components satisfy a given $m_i \in M$ indicated by the "yes" condition, and (b) that no components satisfy a given $m_i \in M$ indicated by the "no" condition. Thus, the next step either returns the best component among multiple 1D component candidates according to a ranking criteria (e.g., performance, cost, weight, etc.) (Step 6), or executes the default mapping rules when no components are found (Step 7). The final step (Step 8) instantiates the 1D components and completes the synthesis of the 1D model. The final step also handles the unconnected ports in the synthesized 1D model; these unconnected ports are typically connected to sources or sinks of energy, material, or signal flows (e.g., constants, grounds, etc.). Our current implementation informs the user of such missing connections.

The most challenging step in our algorithm is the mapping of functions to 3D artifacts (Step 3) to create $M$. One of the most simple yet robust options to map functions to 3D objects is to rely on the domain experts. Domain experts know best their domains and they are already part of a PLM workflow. Thus, they can annotate their 3D models with functional information and create the mappings for our algorithm in their daily workflows.

B. Product Lifecycle Management Workflow for Mapping Functions to 3D Artifacts

In industry, PLM is the process of managing the entire lifecycle of a product efficiently and cost-effectively, from the idea conception, design and manufacture, through service and disposal. PLM integrates people, data, models, processes and business systems from the same and partner organizations through an information backbone that enables real-time collaboration and a consistent set of information throughout the entire product lifecycle. Computer-Aided Design (CAD), Computer-Aided Engineering (CAE), Computer-Aided Manufacturing (CAM), Product Data Management (PDM), and digital manufacturing converge through PLM software. It is also important to note that people are the key stakeholders in PLM systems and creating workflows that make them more efficient is a top priority. Therefore, the PLM design workflows are ideal for creating the mappings of functions to 3D artifacts.
that would be otherwise very difficult to create by algorithmic means.

In this paper we propose the use of functions as design contracts between two or more PLM users that specify how two design elements relate to each other. In the PLM workflow context, functional models are formal requirements created early in the design; they capture the design intent of the entire system. Therefore, we leverage the system-level scope of functions to propagate the design intent in 3D models. Establishing a design contract is a semi-automatic process because it requires two or more people first to agree, and then the PLM system automatically maintains this relationship.

Our proposed PLM workflow can be chronologically described in a few steps. First, the system engineer gathers requirements from the customer and regulatory agencies which are managed by the PLM software. Second, the requirements are formalized by the system engineer in a functional model (energy, material and signal flow). The functional model is then published in the PLM software and functions become visible to 3D tools. These functions represent the design intent and carry the semantics and the context that tool-specific users can allocate to components in their domain-specific models; for example, using trace links or drag & drop as shown in Figure 5. Once the association of functions to 3D artifacts is established, the PLM system tracks and maintains them. These associations represent the $M$ mapping that is fed to our algorithm to synthesize 1D models.

Fig. 5. Allocation of functions to 3D components through drag & drop in a CAD environment.

Another advanced option is to generalize our algorithm and the PLM workflow with the creation of synchronization objects. The synchronization object contains all the information of mappings, allocations, parameterization between the functions in the functional model, 3D artifacts in the 3D model, and 1D artifacts in the generated solutions in the 1D models. This synchronization object can be useful for maintaining consistency between functional, 3D, and 1D models at the variable and parameter level. This means that changes in a 3D model can be consistently reflected in the 1D model, and vice versa. For example, a change in the solar panel dimensions in the 3D model can be propagated to the area being exposed to the sun in the Amesim model. Similarly, a change in the angle between the sun and the solar panel in Amesim can change the orientation of the satellite with respect to the sun in the 3D model.

An exemplary PLM tool-chain implementation consists of the 3D CAD system (e.g., NX), the 1D modeling and simulation (e.g., LMS Amesim), and a PDM system (e.g., Teamcenter) as the data and functionality backbone including the synchronization objects and functional modeling and constraints. Another exemplary implementation consists of OpenSCAD [28] or Blender [29] for the 3D CAD system, Modelica [1] for the 1D modeling and simulation, and SysML models and additional code for the synchronization objects, communication between tools, functional modeling and constraints.

In the following section, we present a fully associative constraint language to specify model constraints as the synchronization objects between 1D and 3D models, and present a tool for automatically enforcing these constraints.

IV. MODEL CONSTRAINTS FOR 1D-TO-3D SYNTHESIS AND SYNCHRONIZATION

A. Model Constraints

Unlike functional models, which mainly focus on describing the flows of the intended system, model constraints can express the global rules governing the model construction mechanism and describe the component's inner properties. Researchers have successfully applied constraint solving systems for validating and selecting optimal models in a broad range of problems and application domains [30], [31], [32], [33], [34].

DARPA META II [35] is a project aiming to develop a design method and modeling tools for effectively modeling cyber-physical systems (CPS), which can greatly reduce both design cost and time. Their methodology combines Platform-Based Design and Contract-based Design. In their modeling environment, they provide a contract language to specify the constraints for model components. These are constraints of component properties or variables and composition rules for composition of components, which are used for validation of each component instance and the compatibility between components within the same architecture.

OpenMETA [36] is a modeling toolset for design space exploration for CPS with the support of a constraints-based pruning mechanism. Constraints element is an important feature that allows encoding different user requirements, such as system function and performance. Similar to DARPA META II, OpenMETA also supports constraints for components parameters/properties and compatibility constraints for component connections. In the backend, there exists a design constraints solver to check whether these constraints are satisfiable or not.

As discussed above, the existing model constraint mechanisms are designed for a specific modeling language. If we want to enforce some constraints on both 1D and 3D models, then these constraints have to be manually encoded within these two different models. Additionally, the main objective of existing systems is the validation of models rather than automatic synthesis between models.

In this paper, we further propose that these constraints can be abstracted to the functional level and integrated with the functional models, which can be used for both model
consistency checking and model synthesis. This is possible, because different models originate from the same function intent, thus also obey the same constraints in general. Our approach is to design a constraint language to not only specify the model properties, but also constrain the dependencies between parameters of different models, which enables automatic model reflection from one model to another once the association of functions to structures is manually created (see Figure 2).

B. Overview

A functional model is a high-level system graph describing the interactions between different function components. For each function component, it can be implemented with either one or more concrete model components. It makes sense that any constraints on model components can be abstracted and lifted to the function components. So it’s natural to extend the functional model with the constraint mechanism by providing an option for adding constraints for each function component.

Figure 6 shows how our new constraint-based functional model works in the synthesis and synchronization between models. To illustrate the idea, consider the pneumatic cylinder from an excavator shown in Figure 6. In a 1D model, a pneumatic cylinder component can be expressed by a hydraulic cylinder Amesim component. In a 3D model, this cylinder can be expressed by a geometric model consisting of a hollow cylinder created in NX. However, at the functional level, the “pneumatic cylinder” – regardless of its representation – provides the convert hydraulic energy to translational mechanical energy function. The key observation is that we encode a functional constraint along with the function that expresses a general rule for describing the structure and behavior of a “pneumatic cylinder”. For example, the constraint \( V = \pi r^2 h \) establishes the relationship between the general concept of a dead volume in the cylinder \( V \), its radius \( r \), and height \( h \). Interestingly, the dead volume concept is used by the 1D model to solve the hydraulic differential equations and can be mapped to the \( \text{AME.Volume} \) parameter in the Amesim component. Similarly, the radius and height concept are used for calculating the kinematic equations in the 3D model and can be mapped to \( \text{NX.radius} \) and \( \text{NX.height} \) parameters in the NX model. In our constraint language “\( = \)” represents equality and not assignment; this allows for declarative constraints that are solved by the constraint solver during compilation.

Notice that the system and language are capable of mapping a single constraint to more than one instance. In this example, an excavator has two instances of the pneumatic cylinder and therefore our system maps a single top level model constraint to the individual components in Amesim and NX. With all these mapping information, both Amesim and NX models are related automatically with constraints as their mediator for information synthesis. Any modifications to \( \text{NX.radius} \) or \( \text{NX.height} \) can be automatically propagated to \( \text{AME.Volume} \) through the evaluation of the constraint. An interesting situation occurs when a change in the Amesim volume happens. Since the volume is computed using two NX features (radius and height), there are multiple possible new parameterizations that satisfy the constraint. Therefore, the designer should specify which parameter has higher priority, or which parameters are locked.

![Fig. 6. Overview of Functional Model Equipped with Constraints Mechanism.](image)

Although not strictly necessary, it helps if the NX models are parametric models. Our constraint system leverages the parameters in these parametric models to map constraint variables to NX parameters. Amesim models are already parametric, thus making it straightforward to map constraints to Amesim component parameters.

The introduction of constraint mechanism to functional model enables us to specify constraints of our interested model properties, and then automatically enforce these constraints on its succeeding Amesim and NX models with our developed constraint interpreter/solver and a communication mechanism to reflect the updated information from one model to another.

C. Constraint Language

1) Syntax: The constraint language \( (C) \) consists of two main language structures, one for model property constraints \( (C_p) \) and another for model mapping constraints \( (C_m) \) as shown by the following BNF form:

\[
C ::= C_p; (C_m)^* \\
C_p ::= (E \approx E)^* \\
E ::= E \square E | \diamond E | ID | DECIMAL \\
C_m ::= (ID .)^* ID \rightarrow (ID .)^* ID | (ID .)^* \{(ID \rightarrow ID)^*\}
\]

The intuition behind model property constraints is that a model property can be expressed as an expression \( E \), and \( E \) can be represented as either a constant, which is represented by DECIMAL, such as the acceleration of gravity, or a variable, represented by ID, such as time, or an arithmetic expression with either some binary operator \( \square \) or unary operator \( \diamond \). And the constraints between related model properties can be represented as some equation formulas or other logic formulas with the operator \( \approx \), which can be \( =, \neq, \geq \) and so on. The star \( (*) \) symbol means that there can be any number of constraints.

The model mapping constraints \( (C_m) \) is used to specify the mapping between parameters or components of two related models. This builds the relationship between two models and enables automatic model updating for one model whenever
changes happen to another. The variable *ID* is an identifier and can be quite flexible, it can denote model property, which means mapping relationship between two different model properties, or it can denote model component, which means mapping relationship between two different model components, or it can be extended to express tool-specific data. *(ID.*)* represents a prefix with dot , as separator and star * means it can repeat zero or more times.

The following section will present a concrete example that shows how the constraint language looks like, and how it works in constraining the model properties and building relationships between different models.

![Excavator Model in Amesim and NX](image)

2) Semantic: This section will give an informal semantics for constraint language through the excavator example shown in Figure 7, which shows its (a) Amesim model and (b) NX model.

Both Amesim and NX provide the parameter interface ports to expose internal parameters that are used in the internal representation of the abstract model components. In this example, the volume \( V \), radius \( r \), and height \( h \) are exposed through parameter interfaces. These parameters can be connected to constraint components, where each parameter becomes a variable to be used in a mathematical equation that describes a constraint.

To make it simple, we take the component cylinder from excavator to illustrate the general ideas. The \( \text{cyl2} \) on NX model corresponds to \( \text{JACK02} \) in Amesim model and \( \text{cyl3} \) corresponds to \( \text{JACK03} \). For cylinder in NX, it has properties height \( h \) and radius \( r \), while in Amesim, \( \text{JACK02} \) and \( \text{JACK03} \) only have property volume \( V \), but they are related with the constraint: \( V = \pi r^2 h \).

Figure 7 (c) shows the details of \( \text{JACK02} \) and the cylinder through \( \text{JACK02} \) and \( \text{cyl2} \). \( \text{JACK02} \) has two ports, \( \text{port1} \) and \( \text{port2} \), which have properties called \( \text{dead}1 \) and \( \text{dead}2 \), that correspond to the volume \( v \) in the constraint. Similarly, \( \text{cyl2} \) also has two ports \( o1 \) and \( o2 \), but with two different properties \{\( r1, h1 \)\} and \{\( r2, h2 \)\}, corresponding to radius \( r \) and height \( h \) in the constraint. Note that the port names \( port1, port2, o1 \) and \( o2 \) can be any names as long as they are unique identifiers to distinguish different instances within the same component. It is important for the constraint solver to have unique identifiers, because it avoids the ambiguity of the associated constraints between two models.

(1) \( V = \pi r^2 h \);
(2) \( \text{JACK02}.\text{port1}.\{V \rightarrow \text{dead}1\} ; \\
\text{JACK02}.\text{port2}.\{V \rightarrow \text{dead}2\} \)
(3) \( \text{cyl2}.\text{o1}.\{r \rightarrow r1, h \rightarrow h1\} ; \\
\text{cyl2}.\text{o2}.\{r \rightarrow r2, h \rightarrow h2\} \)
(4) \( \text{JACK02}.\text{port1} \rightarrow \text{cyl2}.\text{o1} ; \\
\text{JACK02}.\text{port2} \rightarrow \text{cyl2}.\text{o2} \)

The above statements specify constraints for \( \text{JACK02} \) and \( \text{cyl2} \), where formula (1) specifies the model property constraints, and mappings (2) (3) (4) give their mapping constraints. Formula (1) is a global model constraint between constraint variables, while mappings (2) and (3) are mapping constraints between constraint variables and parameters of component instances in Amesim and NX. Mapping (2) specifies that \( \text{JACK02} \) has subcomponents \( \text{port1} \) and \( \text{port2} \), which have a volume property \( V \). And \( V \) maps to \( \text{dead}1 \) and \( \text{dead}2 \). Mapping (3) specifies that \( \text{cyl2} \) has ports \( o1 \) and \( o2 \). \( o1 \) has a radius property \( r \) that maps to \( r1 \), and a height property \( h \) that maps to \( h1 \). Similar mappings are provided for \( o2 \). Mapping (4) provides the relationships between \( \text{JACK02} \) and \( \text{cyl2} \), where \( \text{port1} \) is related to \( o1 \) and \( \text{port2} \) is related to \( o2 \).

As we can see from this example, both Amesim model and NX model are regarded as two different instances of the same functional model, they share and comply with the constraints defined in the functional model, which makes it possible to keep consistent between two different models by implementing appropriate constraint solver as discussed in the following section.

D. Constraint Solver

The user is in charge of writing the constraints. For example, \( V = \pi r^2 h \). During synthesis, these constraints are solved by a constraint solver and the values of the individual variables are propagated to the components where they originate. An additional checking step may include validating whether the computed parameter from the constraint solving satisfies the limits in the domain-specific representation of the component; for example, if the calculated volume does not exceed the bounds specified in the Amesim component.

To interpret the constraints automatically, we have implemented a constraint solver based on ANTLR parser generator [37]. The constraint solver works in three steps:

1) parse the mapping constraints, build the mapping information for property constraints and the related design models, and stores the results in hash maps.
2) collect constraint-related properties information of both the current working model and its associated models, which can be achieved by calling corresponding model APIs. For example, Amesim provides the Python Circuit API to retrieve and update its model information,
and NX provides the C# NX Open API to perform similar automation.

3) evaluate and solve the constraints by parsing the expression Abstract Syntax Tree, fetching values according to the model properties, mapping the values to the constraint equations, and solving the constraint system. For the current working model, a constraint is solvable if all its constraint variables are computable. That is, the system of equations is fully determined. Cyclic dependencies are not allowed in the AST, and therefore the complexity of evaluating the constraints is \( O(N) \), where \( N \) is the number of nodes in the AST.

An important use-case for the constraint solver is to communicate constraint-related model information automatically between Amesim and NX models. Figure 8 shows how the communication mechanism works among different models according to the specified constraints in the functional model. Ame-client is a client application built with the Circuit API, and it provides an interface for Amesim model to communicate with the outside models. Similarly, NX-client is a client application built with NX Open API, and it provides an interface for NX model to communicate with the other models. The coordinator in the center will coordinate all communications between different model client applications and constraint solver. If the user makes any changes to the properties of the NX model, the coordinator will call the NX-client to retrieve its updated information, and then pass this information to the constraint solver to reevaluate the constraints. The new constraint results are then pushed to the Ame-client, which will automatically enforce these new constraints according to the new updates in NX model. It is symmetric for the other direction (from Amesim to NX). This bidirectional communication mechanism allows that any changes in one model are automatically reflected in another model. Usually, this is done manually by the user, and it becomes more difficult if the Amesim model and NX model are developed by different people. So this automatic communication mechanism can greatly save human effort and make developers work more efficiently.

V. EXPERIMENTAL RESULTS

The proposed system has been implemented in a C# prototype. In this section we present the two key results for 3D-to-1D model generation, and for 1D-to-3D model generation. The 3D-to-1D model generation can be qualitatively compared to existing approaches. However, the 1D-to-3D model generation is a novel capability introduced by our work and therefore cannot be compared against other techniques.

A. 3D-to-1D Model Generation

We demonstrate the capabilities of our system using a high-fidelity 3D CAD model of the Reuven Ramaty High Energy Solar Spectroscopic Imager (RHESSI) satellite shown in Figure 1. The RHESSI mission\(^1\) consists of a single spin-stabilized spacecraft in a low-altitude orbit inclined 38 degrees to the Earth’s equator. Our focus is to automatically generate a 1D mechatronic simulation for the unfolding functionality of the 8 solar panels once the satellite is in orbit as shown in Figure 9.

![Fig. 9. Solar panel unfolding simulation.](image)

Using the 3D CAD (Fig. 1) and a subset of the functional model (Notice that Fig. 3 includes the “Convert EE to RME” function as inputs), and the Amesim library as the 1D Database consisting of over 6000 components, our algorithm generates the 1D model shown in Figure 10(a). This model includes the inner solar panel \( z \) connected to the frame (ground) via the inner rotary joint \( z \), the outer solar panel \( x \) and the outer joint \( y \). Notice that compared to Fig. 1, our algorithm matches the “Convert EE to RME” function and finds a motor component in the Amesim library that is instantiated twice in the 1D model as \( z \) and \( z \). Similarly, when we include the “Store EE” function, our synthesis algorithm finds a battery component in the Amesim library that is instantiated as \( z \) in the model shown in Figure 10.

To qualitatively compare the capabilities of our system against existing approaches consider the model shown in Figure 11. This model was generated using hard-coded mapping rules of 3D-to-1D components provided by the Step 7 in our algorithm (see Figure 4). This is equivalent to models generated by existing approaches because they always rely on the same hard-coded rules. Although it is possible change the hard-coded rules to generate models similar to the ones generated by our algorithm in Figure 10, this requires manual

\(^1\)RHESSI is a satellite dedicated solely to the study of solar flares and designed, built, and operated by an international consortium led by scientists at the University of California, Berkeley. It was launched on Tuesday, February 5, 2002 by the National Aeronautics and Space Administration (NASA).
Fig. 10. Example of a satellite’s functional model conversion of a) rotational mechanical energy and b) stored electrical energy.

Effort from experts and this can take several hours of error-prone labor. On the other hand, our algorithm can generate various models without changing any rules. This is because our algorithm relies on a search of equivalent components in 3D and 1D databases using functions as queries. The main benefit of our approach to the end user is the ability to quickly generate models that include different design intents. Notice that the models in Figure 10 are complex in the sense that multiple disciplines are combined and the design intent specified by the user using functions is preserved.

B. 1D-to-3D Model Generation

To verify the proposed method for model constraints, we have implemented a prototype of the constraint solver based on the solving procedure mentioned in Section IV. The goal is to illustrate a new capability of 1D-to-3D model generation on the excavator model shown in Figure 7.

Figure 12 shows an example where changes in the excavator’s Amesim model (1D) are automatically synchronized in the NX model (3D). The constraint expresses a synchronization object between Amesim’s “piston diameter” $\text{JACK02.diamp}$ and NX’s piston diameter expressed by $\text{cyl2.p0}$. Initially, both $\text{diamp}$ and $\text{p0}$ are 110mm and a new value of 210mm is set. After the change in Amesim, our system propagates the change to the NX model and updates the model with the new diameter. Notice that the 3D model in Figure 12 shows a larger inner diameter of the hydraulic piston.

VI. LIMITATIONS AND FUTURE WORK

In its present state, our prototype only works with 3D geometric CAD models. 3D computer-aided engineering (CAE) models are not currently supported. However, it is important to note that our approach is able to handle the CAE use-case because the constraint language can be easily adapted to the constructs and variables in the CAE models. Similarly, our approach can be adapted to control models (e.g., Matlab/Simulink). We believe that a good direction of future work is the exploration of function-harmonized 3D CAE, 3D CAD, 1D, and control models. This paper presents a step forward towards that vision.

Functional models are the enablers for our approach because they provide the unifying abstraction and semantics to harmonize 3D and 1D models. One limitation of existing design tools – not of our approach – is the lack or very limited support for functions. We believe it is necessary for the community to better integrate functional modeling as part of the data models and interfaces of the system design tools. Without proper support of functional modeling in the system design tools, our approach would likely have a slow adoption.
Our current implementation still relies on the manual mapping of functions to 3D structures. This manual step is a limitation that could be solved in the future. With access to a large database of functional and 3D models, we envision that a machine learning algorithm could be trained to create these function-to-3D mappings automatically. The biggest challenge will be obtaining a sufficiently large and representative data set of functional and 3D models. The data sets are typically siloed in a company, and distributed across the organization in different departments and teams. Collecting this data is a grand challenge.

VII. CONCLUSIONS

Automatic model generation and synchronization is significant for both reducing manual effort and improving model design procedures. In this paper, we explore a new way to take it one step further in this direction by integrating constraint-based functional model into the model generation procedures. This paper shows that the quality (in terms of disciplines and component variability) of automatically generated 1D models from 3D models can be significantly improved if functional information is considered. Our new synthesis algorithm combines the information extracted from 3D models to design intent extracted from functional models, queries for 1D components fulfilling the desired functions in a 1D database, and instantiates the 1D components according to a ranking function or default rules. Our results with a realistic mechatronic use-case show the feasibility of the idea and demonstrate that design intent originally not available in the 3D CAD model can be carried over to the automatically generated 1D models. On the other hand, the model constraints provide a construct for different models and enable the synchronization of 3D models with respect to the updated 1D models. The consistency between 1D and 3D models can therefore be automatically enforced once the association of functions to structures is manually established.

REFERENCES


