Context-sensitive Synthesis of Executable Functional Models of Cyber-Physical Systems

Arquimedes Canedo
Siemens Corporation
Princeton, USA
arquimedes.canedo@siemens.com

Eric Schwarzenbach[*]
Princeton University
Princeton, USA
easchwar@princeton.edu

Mohammad Abdullah Al Faruque[†]
University of California
Irvine, USA
mohammad.alfaruque@uci.edu

ABSTRACT

The high complexity of cross-domain engineering in combination with the pressure for product innovation, higher quality, time-to-market, and budget constraints make it imperative for companies to use integrated engineering methods and tools. Computer engineering tools are mainly focused on a particular domain and therefore it is difficult to combine different tools for system-level analysis. This paper presents a novel approach and tool for integrated cyber-physical systems (CPS) design based on the FBS (Function-Behavior-State) methodology where multi-domain simulation models capturing both the behavioral-structural aspects of a system are automatically generated from its functional description. Our approach focuses on simulation-enabled FBS models using automatic and context-sensitive mappings of standard Functional Basis elementary functions to simulation components described in physical modeling languages (i.e. Modelica). Using a real electro-mechanical CPS application we demonstrate how our context-sensitive synthesis approach generates industry-quality executable functional models of higher quality than state-of-the-art approaches using manual mapping.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Representation languages*; H.1.1 [**Information Systems**]: Models and Principles—*General systems theory*

General Terms

Cyber-Physical Systems, Mechatronics, Design, Languages

Keywords

Functional Modeling, Behavioral Modeling, Simulation

[*]The work described in this paper was performed while the author was at Siemens Corporation, Corporate Technology.

[†]The work described in this paper was performed while the author was at Siemens Corporation, Corporate Technology.

1. INTRODUCTION

Product development — from basic consumer products to complex systems — is facing an unprecedented level of complexity due to the interactions among hundreds to millions of heterogeneous components, tight time-to-market and production cost, need for innovation and new services, safety and environmental regulations, and consumer requirements [17, 35]. Although systems engineering methodologies facilitate the intensive cross-domain collaboration in temporally and geographically distributed teams to create better performing products, the main bottleneck in mechatronics and cyber-physical systems (CPS) design still is the difficulty of realizing concurrent engineering, where the tasks that occur at the various stages of product development are overlapped in time to reduce errors, cost, and time [1]. A CPS is a heterogeneous and/or hybrid system that can be geographically distributed and/or integrated within a single form factor to perform control functionality in software to actuate and/or manipulate a physical, chemical, or biological process [28]. In such a system, modern high speed communication plays an important role for integrating the subsystems. Opposite to the traditional mechanical-centric approach where software and control are an afterthought or a last-minute effort, an integrated CPS product development relies on cross-discipline modeling and early identification of system-level problems to provide a highly flexible and less expensive approach for creating innovative and truly integrated CPS. Currently, as observed by researchers and practitioners, the main limiting factor in complex industrial CPS development is the lack of tools to support the *early concept design* phase [1, 10, 13, 36]. It has been estimated that 70-80% of the cost of a product is determined by the decisions taken at this phase [9]. In this paper, we present a general method and a tool to improve the current practice of early concept CPS design of complex industrial systems from machine tools and robots to transportation, energy, manufacturing systems, and nano-/micro-scale implantable medical devices [18].

Modeling and simulation have become standard practice in the development of CPS because it is an economical and effective way to design, validate, and test the complex industrial CPS. Products can be *virtually* explored and analyzed without the need for physical prototypes because software is used to estimate the dynamic behavior of the system under a large variety of conditions. This, in principle, enables the early identification of system-level problems. In practice, however, modeling and simulation have evolved independently on the different engineering domains and coupling different tools for a holistic analysis and therefore, multi-domain simulation is very complicated and sometimes not even possible. We believe that integrated CPS design requires a general cross-disciplinary approach capable of characterizing how the design changes in one domain affect the rest of the domains in the system and the system-level behavior. The FBS methodology (Function-Behavior-State) [37] is a general framework for system development where *functionality* (a discipline-independent description of what the system does) is used to unify the reasoning, communication and understanding between engineers and tools of various disciplines. Functions are related to states (structures or entities and attributes of entities) and behaviors (a change of states over time). States and behaviors are concrete realizations of functions that can be, theoretically, used to simulate the system. Unfortunately, the support for the FBS methodology is either limited to in-house tools or heavily focused on fundamental research [10]. Our objective is to create a more practical approach and tool based on FBS principles that improves the early concept design, modeling, and simulation of complex industrial CPS systems.

Our method automatically generates industrial CPS simulation models from functional models and we present a novel synthesis method that contextualizes functions in a functional model according to their surrounding flows to create function-to-component mappings that are comparable in quality to manual selection of components done by experts. Rather than creating behavioral and structural models independently, and manually mapping functions to behavior and structure, our synthesis tool auto-generates multi-domain Modelica [22, 23] simulation models that capture both the structural and behavioral aspects of a functional model as shown in Figure 1. Our method improves concurrent engineering by allowing cross-domain team collaboration through functional modeling, and eliminates the effort associated with manual behavior and structural modeling by automating the creation of multi-domain simulation models suitable for integrated CPS design. Compared to the previous work in FBS, the novel contributions of this paper can be summarized as follows:

1. A practical FBS approach where functional models are automatically translated into behavioral-structural simulation models using physical modeling languages (i.e. Modelica [22]). Traditionally static (non-executable) FBS models are now transformed into dynamic (executable) *simulation-enabled FBS models* that provide an integrated multi-disciplinary system simulation, thus eliminating the problem of simulation tool incompatibility. This system-level simulation capabilities make integrated CPS design more cost effective and reduces the time-to-market.
2. A synthesis method and tool for automatically generating high-quality system-level simulation models from functional models based on the contextualization of flows and feedback flows in functional models to improve the functions-to-component selection.
3. A new `recycle` function in the Functional Basis language [12] to facilitate the generation of closed loop system-level simulation models, an essential concept for control, software, sensors, and actuators of a CPS.
4. A use-case of an electro-mechanical CPS application that highlights the key benefits of our approach for the design of large-scale, small-scale, highly distributed, centralized, and electro-mechanical CPS.

The rest of the paper is organized as follows. Section 2 introduces the key benefits on functional modeling and system simulation for CPS design. Section 3 presents our new context-sensitive synthesis method. The details of an electro-mechanic use-case and our tool implementation are presented in Section 4. Section 5 presents the related work and discusses the possible directions for future research. Section 6 provides the concluding remarks.

2. MULTI-DISCIPLINARY MODELING AND SIMULATION FOR CPS

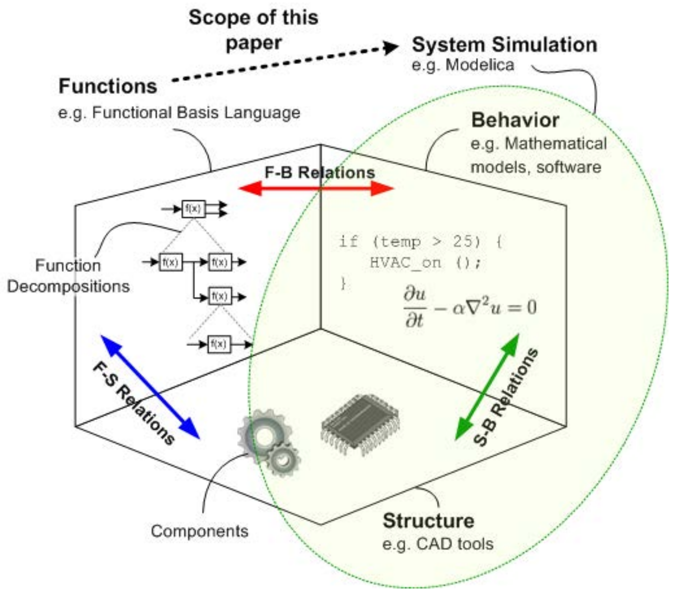

Figure 1: Traditional FBS provides non-executable system representations. This paper proposes the use of system simulation languages (e.g. Modelica) to create executable behavioral-structural models automatically from functional models.

2.1 Functional Modeling

Functional Modeling (FM) is a Systems Engineering (SE) activity where products are described in terms of their functionalities and the functionalities of their subsystems. Fundamentally, a functional model reflects *what the system does*. Because a Functional Model decouples the design intentions (functions) from behavior and/or structure, it can be used as the basis for communication among engineers of different disciplines. Functional modeling reflects the design intentions that are typically driven by the product requirements and the human creativity. Functional Modeling creates a bird's-eye view of a system that any person in an organization, regardless of the domain of expertise and responsibility, can understand.

Functional modeling is a creative human exercise and it is acknowledged by many researchers and practitioners to be a subjective process [10], therefore suitable for concept design. Defining a system in terms of its functionality[1] may seem simplistic and unnecessary but this is exactly what improves the systems engineering process by consolidating multiple engineering paradigms (e.g. electrical, mechanical, software, thermal engineering) into a unified system representation. By *making explicit the implicit knowledge of the engineers*, a functional model exposes the obvious facts about the system that people can easily understand, regardless of their domain of expertise. This improves the communication among different disciplines because it brings the minds of the domain experts to the same level of abstraction that is facilitated by natural language. In FM, specialized knowledge is irrelevant and discouraged. For example, the sentence "`transport people`" describes the goal of any transportation system and it intentionally ignores the different embodiments such as an automobile, a train, or an airplane. A functional model can also be refined into more specific descriptions in a process referred to as *functional decomposition*. The "`transport people`" function can be decomposed into sub-functions such as "`navigate airspace`"

[1]Functionality of a system is defined as its purpose, intent, or goal.

or "navigate highways" implying the design of an airplane or an automobile. Furthermore, sub-functions can be decomposed creating a *functional decomposition tree* where the root node represents the top-level function and the leaf-nodes represent elementary functions such as "transform chemical energy to mechanical energy".

Formalization of functional modeling [12, 27] is critical for implementing this highly subjective and creative process into the systems engineering practice. The de-facto functional modeling syntax is a block flow diagram where a block represents a function (process) with inputs and outputs (flows). An interesting practical problem of functional modeling is that different designers tend to create substantially different functional models of the same system. An important improvement has been the creation of the Functional Basis language [12, 33] that has well defined syntax and semantics. By using a constrained vocabulary and well defined function and flow semantics, functional modeling practice can be normalized as different designers can rely on the same language to create and understand systems. The Functional Basis defines three flow categories (*material*, *energy*, and *signal*) that expand into a total of 18 flow types, and 8 function categories with a total of 32 primitive functions. We use the Functional Basis language in our CPS implementation because is the de-facto standard for functional modeling and also to enable compatibility with other tools and methods [21].

In this paper, we present a practical approach that uses functional models as the "glue" to maintain consistency between different models and people from different disciplines. In addition, we present a method for automatically generating dynamic system simulation models from functional models. Inferring a context from a functional model is the key enabling technology for generating high-quality simulation models that can be customized for different purposes including discipline-specific analysis and detailed simulation, trade-off studies of design variants, or a holistic system simulation involving multiple disciplines.

2.2 System Simulation

Highly specialized domain-specific modeling languages and simulators have been developed to support particular industries such as VLSI and robotics. These tools typically rely on a *mathematical modeling* approach where behavioral descriptions of the system represented by differential and discrete equations are translated into mathematical models. The main challenge with mathematical modeling is finding the appropriate system of equations to describe the system. Many high-accuracy and robust algorithms are available for the numerical solution of mathematical models but there is a demand for tools to assist the creation of mathematical models and finding out these system of equations [32]. Moreover, coupling of different mathematical models created by different tools is very complicated due to the inherent differences between engineering domains. Mathematical modeling and simulation is a discipline-specific approach and therefore not suitable for the multi-disciplinary nature of CPS design.

When developing a heterogeneous complex system, coupling models and simulation of different engineering disciplines (e.g. mechanical, electrical, software) is necessary because it speeds up the development time, increases the understanding and communication, facilitates design and optimization, and enables virtual prototyping and verification. *Physical modeling* languages such as Bond Graphs [8], Modelica [22], and Simscape [19] have been developed for inter-disciplinary modeling and simulation of complex heterogeneous systems. Physical modeling relies on the interconnection between physical *components* (i.e. resistor, pump, gear box) that encapsulate behavioral descriptions (mathematical models). Components interact through physical *ports* while honoring the energy conservation principles using the effort-flow variables such as voltage-current in electrical, temperature-heat flow in thermal, and angular velocity-torque in rotational mechanics domain. This modeling paradigm based on energy conservation principles couples various disciplines, allows the integration of user-defined disciplines, and facilitates model reusability. Because each component defines a behavioral model using energy conservation principles, physical models can be translated into compatible mathematical models and numerically solved in a holistic simulation.

Physical modeling enables system-level simulation of the various subsystems and disciplines in a complex heterogeneous system. This is beneficial for the CPS engineering process because it provides a unified system model that engineers can use throughout the various stages of the development. Currently, system-level simulation models are manually created and maintained by experts making it a lengthy and error-prone process. In this paper, we demonstrate how functional models can be used to automatically generate system-level simulation models according to the latest engineering specifications. This not only eliminates the manual effort, but creates a more efficient concept design phase where engineers are allowed to try alternative designs at the functional level and obtain the corresponding system-level simulation models automatically. Although we target Modelica language, our method is general to functional modeling and system simulation and can be easily adapted to other physical modeling languages such as Bond Graphs and Modelica.

3. CONTEXT-SENSITIVE SYNTHESIS OF SYSTEM SIMULATION MODELS FROM FUNCTIONAL MODELS

Finding components to fulfill the functionalities of a system under certain constraints is one of the fundamental principles in engineering that requires domain-specific knowledge, experience, and creativity. **Any tool that attempts to automate this process must produce simulation models that are comparable, in terms of the level of detail and quality[2], to models created by system simulation experts**. This is a challenging requirement because one function may be realized by multiple and different components, and one component may realize multiple functions. In other words, multiple valid simulation models exist for a given functional model, but only a few are useful for modeling the actual system. In this Section, we argue that a functional model is inherently incomplete and therefore insufficient to *reliably* generate high-quality and relevant simulation models. Fundamentally, a functional model is created to satisfy *functional requirements* (what the system is supposed to do) and it does not contain any information regarding the *non-functional requirements*[3]. However, to overcome these limitations, we present a novel context-sensitive synthesis algorithm that infers the missing information from functional models to generate high-quality simulation models based on the following key observations:

- Feedback flows and recycle functions[4] are necessary in a functional model for synthesizing efficient designs that are comparable in quality to designs created by experts. Interestingly, the existing work on functional modeling does not emphasize the importance of feedbacks (loops) in the

[2]We define "quality" of a simulation model in terms of its accuracy, fidelity, and correctness.

[3]Non-functional requirements are any constraint to which the system must adhere, whether it be weight, geometry, reliability, computing power, etc.

[4]The Functional Basis does not define a function that semantically "recycles" or "reuses" flows.

modeling practice and it is very rare to find any feedback in functional models. Feedback loops are critical for synthesizing models with sensors and actuators, essential components in modern CPS. In addition, with more functionality being realized by software, feedback information is essential to the design of closed-loop control systems.

- Merely defining function-to-component relations is not sufficient during system-level synthesis. Traditionally, components *realize* functions and this creates a relationship between functions-to-components. However, due to the fact that many components can realize the same function (and vice-versa), without knowledge of the input/output flows of a function there is no context in which to generate the correct component. A flow-based contextualization allows for robust synthesis of complex CPS products.

3.1 Recycle Function

Most functional models lack feedback information because, in principle [27], a function is a process applied to an input flow to produce an output flow. Similarly, the entire system is seen as a black-box function that applies a process to its inputs to produce outputs. When the functional model without feedbacks is being transformed into a system-level simulation, the synthesizer applies design rules on specific flows that are relevant to the design and creates feedbacks of flows with the help of a new function type `recycle`.

DEFINITION 1. *The **`recycle`** **function** has the following properties:*

- *Its syntax feeds back a flow produced by a successor function to a predecessor function (a function executed earlier in time relative to a successor)*[5].
- *Its semantics define the reuse or returning of material, energy, or signal flow to a predecessor function.*

Figure 2 shows the syntax and semantics of the proposed `recycle` function as an extension to the Functional Basis language. The selection of flows for the creation of feedbacks is a very important design objective. Currently, our method relies on general design and engineering rules created by the engineers to identify potential feedback loops in functional models. Another implementation possibility would be to automatically select various feedback flow alternatives, generate different models, and provide the user with all the design alternatives for further testing and trade-off analysis. We believe that, in some cases, feedback loop creation implies specific design decisions and by decoupling these from the functional model, we honor the purpose of functional modeling of being an abstract and implementation independent representation.

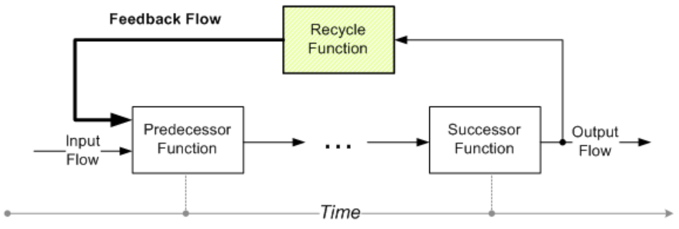

Figure 2: `Recycle` function semantics is the reuse or returning of material, energy, or signal flow to a predecessor function. Syntactically, it can be visualized as a flow from a `successor` to a `predecessor` function.

[5]The Functional Basis [12] specifies that functions execute chronologically from left to right.

3.2 Flow-based Contextualization

In order to reliably generate high-quality simulation models, every function within the functional model of a CPS must be put into context to be correctly mapped to its respective simulation component(s). As previously stated, each function can be realized by an indefinite number of simulation components, and each simulation component can realize indefinite number of functions. To find the correct function-to-component mapping for a given model, each function must be contextualized by its input and output flows. We define two types of flows for every function: primary and secondary.

DEFINITION 2. ***Primary flows*** *are the flows that are inherent to a given function (e.g. thermal energy flow in* `transmit thermal energy`*, or air flow in* `transport gas`*).*

As primary flows are fixed for every function, they add no new information to the system. Therefore, for flow-contextualization, we rely on *secondary* flows.

DEFINITION 3. ***Secondary flows*** *are the non-essential inputs/outputs of a function (e.g. air flow in* `transmit thermal energy`*). Secondary flows help specify mechanisms that functions use, and reduce the N-to-M function-to-component relation down to a 1-to-1 mapping.*

Our synthesizer uses primary and secondary flows to build a *context tree* data structure that can be used to infer a design context and drive the mapping of functions to concrete components. One context tree is built for every function in a functional model during synthesis. The root of the context tree is the *function signature*, or combination of the function and primary input and output flows. Secondary input and output flows are added as internal nodes. And leaf nodes represent the function's *realization mechanism* deduced from basic engineering knowledge and the context given by the root node and the secondary flows. For example, given the function signature `transmit thermal energy` from a `solid` to a `solid`, the most effective realization mechanism for heat transfer is `conduction`.

3.3 Physical Simulation using Structure-Behavior Components

Reusing pre-existing system-level components is possible thanks to the availability of academic and commercial libraries [23, 16, 24]. However, the components themselves are not sufficient for automatic synthesis. It is very important to define the level of component granularity required by our synthesizer to generate correct simulation models. For example, each Modelica component defines both the structure (i.e. a capacitor) and its dynamic behavior using differential equations and/or algorithms (e.g. $I(t) = C\frac{dV(t)}{dt}$). Additionally, a component's *connectors* (or ports) specify the equations to honor energy conservation principles (e.g. Kirchhoff Law's) in terms of *potential* and *flow* variables[6] equivalent to voltage and current in the electrical domain, angular velocity and torque in rotational mechanics, and pressure and flow rate in hydraulics. Finally, components have an *annotation* field that can be used to store information about the component such as its documentation or icon.

We use the namespace of the component in a library to classify its domain (e.g. Modelica.Electrical.Analog.Basic, Modelica.Mechanics.Components) and to locate the component (e.g. Resistor, Damper). Since the synthesizer works at the component

[6]These conjugate variables are known as "potential-flow" in Modelica, "effort-flow" in Bond Graphs [8], and "across-through" in Simscape [19].

level, the equations and algorithms associated with the component are never modified. However, the number of equations and number of statements are two important metrics that have to be available to the synthesizer when generating a *minimal simulation model*.

DEFINITION 4. *A **minimal simulation model** (or minimal model) is a functionally correct model that requires the least amount of simulation components and/or mathematical equations.*

The type of the connectors in a component are used to determine the correct physical interface and generate compliant simulation models. Connector types are also useful to generate the energy conservation laws when a `recycle` function relates various components. A component's annotation field can be used as the means to associate and store the mappings of components-to-functions. Given the required level of component granularity, our synthesizer uses a Modelica language parser to import the library and builds an abstract syntax tree to access a component's connectors, equations, algorithms, and annotations during code generation.

Algorithm 1 Context-sensitive Synthesis Algorithm

```
Require: Functional model $G = (V, E, L_V, L_E)$
Require: Library of system-level simulation components $\mathcal{L}$
Require: Engineering view $\mathcal{W}$
Require: Recycle function rules $\mathcal{R}$
Ensure: System-level simulation model $\mathcal{S}$
 1: $\mathcal{S} = \varnothing$
 2: Build AST for each component $c \in \mathcal{L}$
 3: if $\mathcal{W} \subseteq G$ then
 4:     Create empty (black-box) component $B = \varnothing$
 5:     Create map for all flows $e \in E(\mathcal{W})$ and for all functions $v \in V(\mathcal{W})$
        to $B$
 6:     Create connectors in $B$ for all primary and secondary flows from/to
        $\mathcal{W}$
 7:     Add component $B$ to $\mathcal{S}$
 8:     RESIDUAL = $\mathcal{W} \setminus G$
 9:     for each functions $v' \in V(\text{RESIDUAL})$ do
10:         MATCHES = get list of all matches $m'$ of function-components
            of $v'$ in AST
11:         MINIMAL = Pick component $c$ from MATCHES with the least
            number of equations and algorithmic statements
12:         Add MINIMAL component to $\mathcal{S}$
13:     end for
14: else if $\mathcal{W} = \varnothing$ then
15:     for each functions $v \in V(G)$ build context tree $T$ do
16:         $T$.root = build signature from $l(v)$ and primary
            inbound/outbound flows $l(e_p)$ to/from $v$
17:         $T$.inner = all possible combinations of secondary flows $l(e_s)$
            with $v$
18:         CONTEXT = $T$.root $\times$ $T$.inner
19:         $T$.leaf = deduce realization mechanism from CONTEXT
20:         $\forall$ levels $i \in T$ create a path $P_i$ that satisfies function $v$ and flows
            $e$ connected to $v$
21:         INFERREDDESIGN = find function-to-component and flow-to-
            component matches for $P_i$ in AST
22:         Add INFERREDDESIGN to $\mathcal{S}$
23:     end for
24:     for each rules $r \in \mathcal{R}$ do
25:         if $r$ matches $G$ then
26:             Add function $v'$ with $l(v')$ = recycle to $V(G)$ and
                feedback flow $f'$ to $E(G)$ from $r.src$ to $v'$ and from $v'$ to
                $r.dst$
27:             Add energy conservation equations to $\mathcal{S}$ using $l(r.src)$ and
                $l(r.dst)$ physical domain
28:         end if
29:     end for
30: end if
31: return $S$
```

3.4 Synthesis Algorithm

The proposed synthesis algorithm is listed in Algorithm 1. The required inputs are: a functional model and a library of system-level simulation components. In our synthesizer, the functional model is a labeled directed graph $G = (V, E, L_V, L_E)$, where V is a set of nodes or functions, E is a set of edges or flows, L_V is a set of function labels, and L_E is a set of flow labels. Each function $v \in V$ and each flow $e \in E$ have uniquely assigned labels $l(v) \in L_V$ and $l(e) \in L_E$ using the vocabulary from the Functional Basis language and the newly introduced `recycle` function label. For example, a function w with $l(w) =$ `transmit` and an inbound flow f with $l(f) =$ `thermal energy` represents a `transmit thermal energy` function signature. The component simulation library $\mathcal{L}$ is parsed into a queryable abstract syntax tree AST that, given a set of functions and flows $Q \subset G$, returns a list of components $K \in \mathcal{L}$ that map to Q according to the function-structure relations.

Optionally, the user may provide an engineering view $\mathcal{W}$ and/or recycle function rules $\mathcal{R}$ in the form of [29]. The first step (Line 2) is to parse the library into an abstract syntax tree where equations, algorithms, connectors, and annotations are accessible for the algorithm. If the user provides an engineering view $\mathcal{W}$ (Line 3), the algorithm synthesizes a minimal simulation model *without context trees* in order to reduce the number of components. Since the engineering view represents the subsystem under design, the functions covered by the view are mapped into a single empty black-box component B with a well defined interface derived from the input and output flows to/from $\mathcal{W}$ in the functional model. The reminder of the functions RESIDUAL (functions that are not in the view) are mapped to components. In case that a function maps to multiple components, our algorithm picks the component with the least number of equations and the least number of statements in the component's algorithm in order to synthesize a minimal simulation model for the CPS.

When synthesizing the entire functional model (without engineering views in Line 14), the functional model is traversed one function at a time and one context tree T is built for each function. Based on the context given by the function's signature (root node) and the secondary input/output flows (inner nodes), the realization mechanism is deduced from basic engineering principles and added as the leaf nodes. The context tree is then traversed (starting at the root) to create a path P_i according to the existing secondary flows and the appropriate realization mechanism. This path represents the flow-based contextualized function and all the functions and flows in P_i are mapped to components from the library, thus creating the most appropriate design INFERREDDESIGN. If recycle function rules $\mathcal{R}$ are provided, the last step attempts to create feedback loops in the functional and simulation models. When a recycle rule matches the functional model (Line 24), a feedback is created in both the functional model and in the mapped simulation model. It is very important to guarantee that energy conservation laws are preserved when creating a feedback loop in the simulation model. The rest of the algorithm checks the domain of the source and destination components to generate the appropriate energy conservation equations using potential-flow variables to interface the components associated with the feedback loop.

In the next Section, we present the design of a popcorn electromechanical machine that exhibits complex physical behaviors in the electrical, thermal, mechanical, and fluid domains that need to be precisely regulated and controlled by software and communication. Such behaviors are common to many next-generation CPS applications (e.g. nano-/micro-scale implantable medical devices) and therefore we argue that our methodology is applicable for the design of a broad range of emerging CPS applications [34].

4. EVALUATION

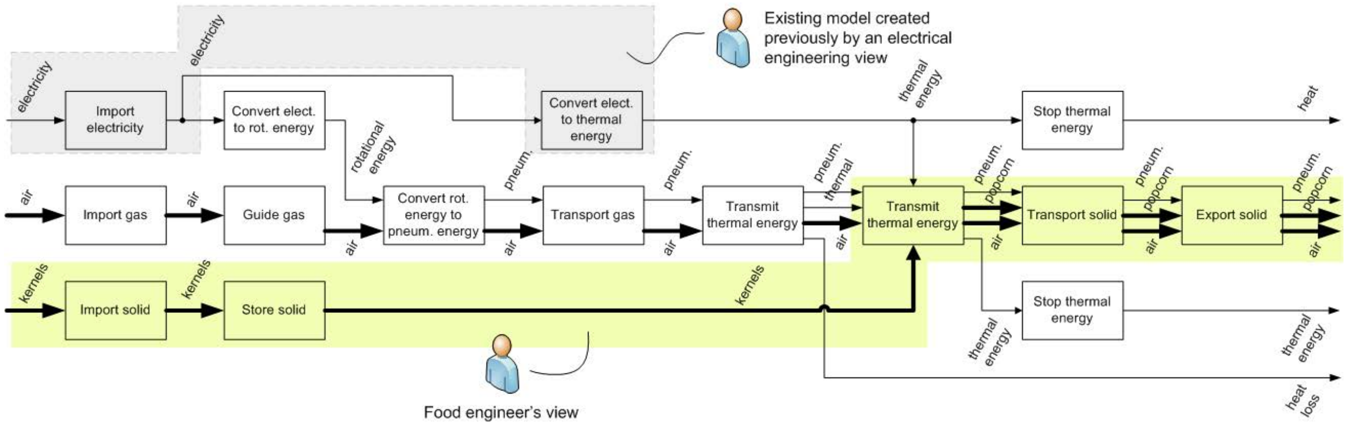

Figure 3: Functional model of an electro-mechanical popcorn machine with a *view* created by the food engineer to develop the "popping model" subsystem.

Consider the functional model of an electro-mechanical popcorn machine inspired by [12] shown in Figure 3. From the material (`air`) and energy (`thermal energy`) flows and functions, it is clear that the design intention is to pop corn kernels with hot air rather than oil. At the early design stages, the domain engineers are allowed to create *engineering views* on the functional model to highlight the areas of responsibility and auto-generate simulation models that allow them to concentrate on their problem. These *views* indicate the synthesizer that the engineer is trying to isolate its subsystem from the rest of the electro-mechanical CPS for detailed design. In this case, the objective of the synthesizer is to generate a satisfactory solution to the rest of the functional model (components that are not in the *view*) to provide a complete but *minimal* simulation model for the component under design. For example, Figure 3 shows a *view* created by the food engineer highlighting the functions for which a more detailed model will be defined. In a graphical functional modeling editor, views can be implemented through selection of functions and flows and grouping.

The synthesizer then creates a black box component for the *view* in Figure 3 and finds a satisfactory minimal solution for the rest of the functions while taking into account reusability of existing electro-mechanical CPS simulation models as shown in Figure 4. Generating a minimal solution is important because less equations produce faster simulations and therefore it increases the productivity of the engineers. Notice that the `electricHeater1` component in Figure 4 is a reusable component that was probably generated using a *view* created by the electrical engineer for the `import electricity` and `convert electricity to thermal energy` functions in Figure 3. If the `electricalHeater1` model is not yet available, the synthesizer would then select a primitive simulation component whose function is to `transmit thermal energy`. This iterative design process adheres to systems engineering practice and our synthesis method enables concurrent systems engineering because it maintains all models synchronized to the latest functional electro-mechanical CPS specifications while allowing multiple engineers from different disciplines to develop subsystems concurrently.

A complete electro-mechanical CPS simulation can be auto-generated using the same synthesis process while taking into consideration the entire of the functional model. A direct mapping of the functional model to components yields a detailed and correct electro-mechanical CPS simulation model that includes thermal (pipes, pump, convection blocks), electrical (electric heater block), and mechanical elements (DCmotor block) as shown in Figure 6.

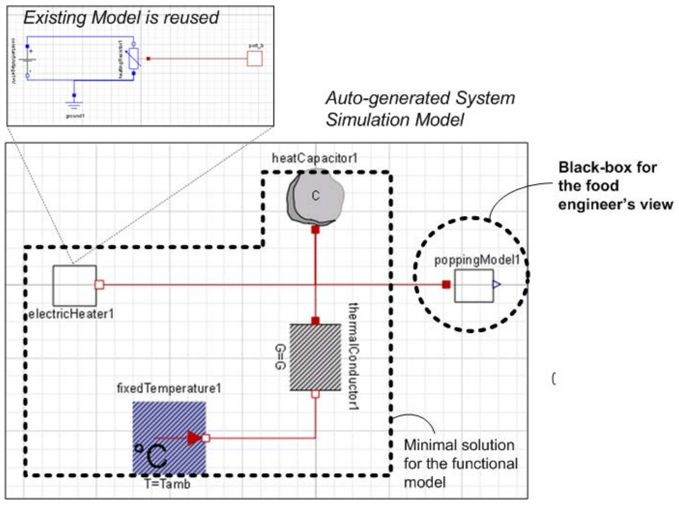
Existing Model is reused
Auto-generated System Simulation Model
heatCapacitor1
Black-box for the food engineer's view
poppingModel1
electricHeater1
thermalConductor1
G=G
fixedTemperature1
°C
T=Tamb
Minimal solution for the functional model

Figure 4: Minimal electro-mechanical CPS simulation model generated from a *view* of the functional model produces efficient simulations.

Notice that the synthesizer correctly infers the design intentions of a hot air popcorn machine and selects a pump, pipe, and convection components to simulate the air flow and the thermal energy transfer to the air flow to cook the kernels. This detailed model solves the functional model, but not efficiently because the air is imported, heated, used to heat the kernels, and it finally bleeds into the the environment. An engineer manually translating the functional model to a simulation model would "*know*" that a better design would be one that *recycles* the hot air. This point illustrates the importance of non-functional requirements. In this example, our simulation model satisfies the functional model, but it is likely that this design is incapable of achieving a sufficient temperature to pop the kernels. Here, the oven temperature is a non-functional requirement. As a result, we argue that functional models are inherently incomplete and do not provide sufficient information to reliably generate efficient designs. Our synthesis approach attempts to compensate for the lack of information in functional models by inferring the context of the design and applying engineering intuition rules to create more reliable designs. For example, in Figure 6, the function `transmit thermal energy` can be realized by any

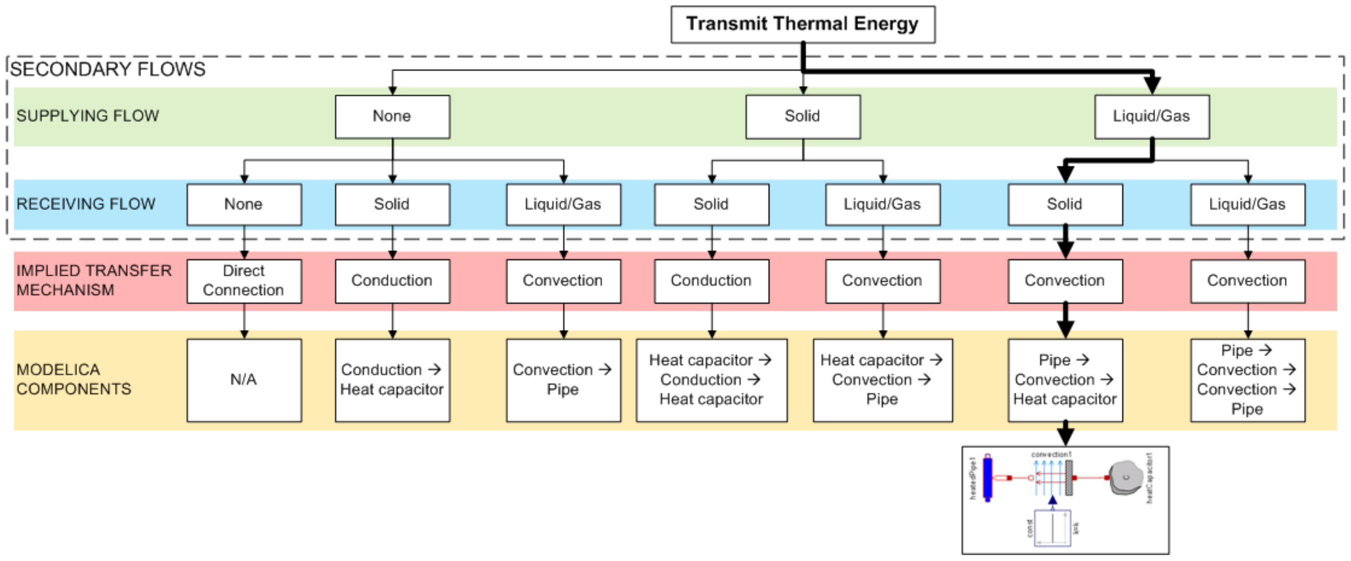

Figure 5: A context tree of the function `transmit thermal energy` in the functional model of the electro-mechanical popcorn machine. Secondary flows within `transmit` functions are separated into *supplying* and *receiving* flows. The supplying flow is linked with the primary flow prior to the function, while the receiving flow is not. By following the context tree, the synthesizer is able to infer the correct components needed to realize the function. The arrows inside the 'Modelica Component' blocks indicate the direction of the heat flow within the component set. A schematic of the connected component set is shown beneath the path chosen for the model.

of several simulation components. However, in our model, it is contextualized by its surrounding flows. The thermal energy flow entering the `transmit thermal energy` block is coupled with an air flow. *This suggests that convection is the appropriate heat transfer mechanism*. The other material flow that enters the `transmit thermal energy` block is the popcorn flow. As it is not coupled with the thermal energy flow, it can be seen as the recipient of the thermal energy, and therefore necessitates the addition of a heat capacitor in the simulation model. This flow-based contextualization allows for robust synthesis of complex electro-mechanical CPS.

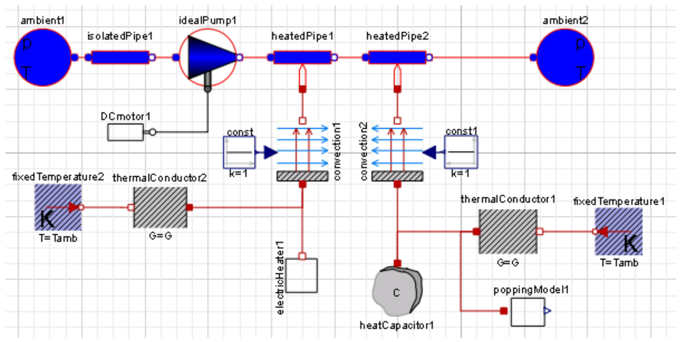

Figure 6: Complete electro-mechanical CPS system-level simulation model generated from direct mapping of functions to components produces an inefficient implementation because the hot air is not reused and escapes into the atmosphere.

Figure 5 shows the contextualization of `transmit thermal energy` using its secondary flows in the context of the electro-mechanical popcorn machine. When the synthesizer reaches the `transmit thermal energy` block, it checks the input/output flows. Thermal energy is the primary flow, so based on its rules, it looks for a *supplying* secondary flow. It finds that this is air, which is a gas, and so it chooses the appropriate path. It then looks for any *receiving* flows. It finds the kernels/popcorn flow, a solid material flow, and once again chooses the correct path. By knowing these two secondary flows, the synthesizer then deduces the correct heat transfer mechanism (i.e. convection) and the corresponding components for the Modelica simulation.

It should be noted that a similar scenario in which the supplying and receiving flows are reversed (i.e. air cooling) would result in the same transfer mechanism, but the order of the Modelica components would be reversed. This is to be expected because the synthesizer traverses the functional model by following flows, and it needs to correctly deduce the direction of the heat transfer to convert the function into simulation components.

Figure 7 shows the resulting electro-mechanical CPS simulation model generated by our context-sensitive method. The main difference with respect to the model in Figure 6 is that this model recycles the hot air and it is used to optimize the cooking time and the energy consumption of the machine. Notice that the open system in Figure 6 (`ambient1` and `ambient2` components) is now closed in Figure 7 by the `absolutePressure1` component. This is an important aspect that our synthesizer is able to infer by creating a feedback flow and inserting a `recycle` function in the functional model to keep `hot air` within the machine instead of allowing it to escape to the environment. Using this model, the system minimizes the amount of thermal energy flow leaving the system, therefore decreasing the energy use and reducing cooking time.

As well as adding a feedback flow for the `hot air`, our new model similarly uses a signal feedback flow to create a simple control law to maintain a constant temperature, as can be seen in the simulation block `constTempHeater1`, an existing model whose output is heat, and whose inputs are electricity and a temperature sensor. Using a simple on-off controller implemented as a state machine [26], the heater attempts to maintain the cooking area at the ideal temperature, as determined by the engineer. This addition further increases the model's efficiency as well as the quality of the popcorn.

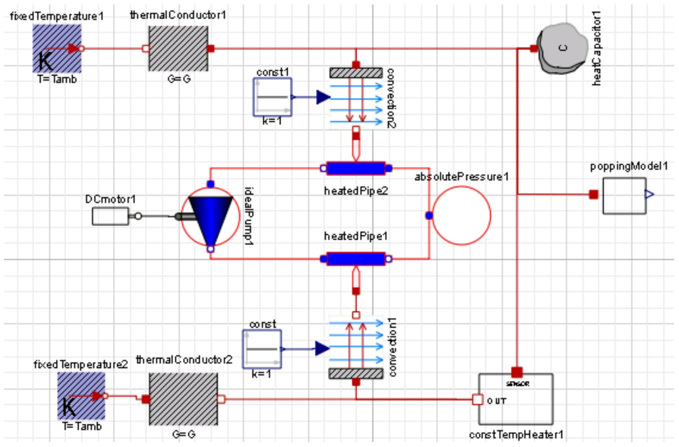

Figure 7: Feedback of `hot air` flow using a `recycle` function produces a more efficient design that optimizes cooking time and energy consumption.

4.1 Synthesis Tool Implementation

Our implementation consists of two parts: the functional model editor, and the electro-mechanical CPS simulation synthesizer. The functional model editor is a graphical editing tool fully compliant with the Functional Basis language distributed to the users as a Microsoft Visio stencil. Figure 8 shows some of the features in our functional editor. The models created with the functional editor are the input for the synthesizer. The functional editor is also capable of exporting the functional model as a matrix for compatibility with other concept design tools such as MEMIC v2.2 [21].

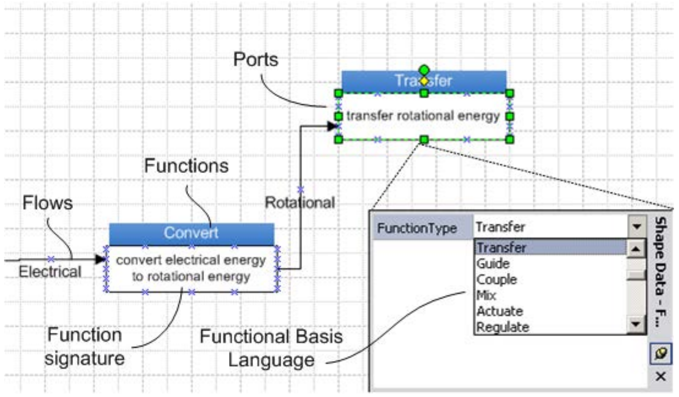

Figure 8: Functional editor features include drag&drop modeling, function signatures for readability, ports to connect functions, flow auto-routing, function and flow type selection based on menus.

Our synthesizer converts functional models into Modelica system-level simulation models. We have selected Modelica because it is an open modeling language and the Modelica Standard Library (MSL) [23] provides more than 1200 models of various engineering domains including electrical, fluid, mechanics, thermal, and state machines suitable for simulating CPS. Figure 5 shows a mapping of few MSL components to "`transfer thermal energy`" function that our synthesizer uses. Notice that our functions-to-components mapping corresponds to explicit F-S relations in the FBS methodology and implicit F-B relations because the MSL components already provide S-B relations as each component includes a mathematical description that defines its dynamic behavior. Thus, when F-S relations are directly created during the mapping, the F-B relations are indirectly created thanks to the existing S-B relations provided by the MSL. Adding support for other system simulation languages such as Simscape [19] would require a new mapping of functions-to-components for the target language.

Figure 9 shows that our context-sensitive synthesis approach improves the quality of the generated models for the electro-mechanical popcorn machine example. While both designs are correct and accomplish the same functionality (See Figure 6 and Figure 7), when the synthesizer creates a feedback flow for the hot air the resulting model is three times more efficient at cooking popcorn than the model generated without considering the feedback flows. Currently, our implementation relies on user selection of feedback flows based on design rules. In this particular example, the design rule specifies that any function bleeding hot air into the environment is compatible with any predecessor function `transmit thermal energy` that transmits heat to a gas.

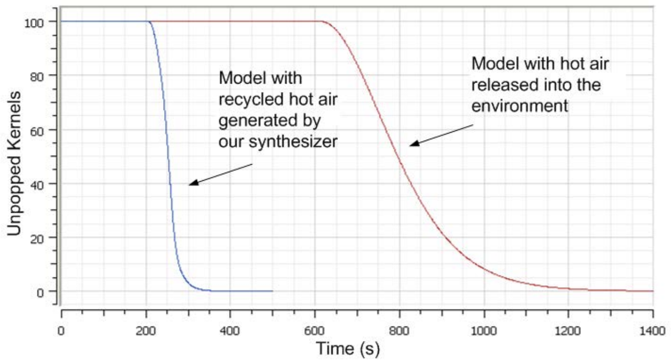

Figure 9: Popcorn popping occurs when the internal kernel temperature reaches 180-190°C. Using a dynamic model of popcorn popping [6] and 100 kernels, these simulation results show that the model recycling hot air achieves a higher popping rate and significantly less cooking time than the model that bleeds hot air into the environment.

A very important aspect of our context-sensitive approach is generating a minimal electro-mechanical CPS simulation model that provides the correct inputs to the subsystem under design. Figure 10 shows the complexity of the generated models for the electro-mechanical popcorn machine example. The minimal simulation model (Minimal Set) consists of 29 non-trivial equations and it is more than 3 times smaller than the two complete system-level models. The performance of the simulation is directly proportional to the number of equations in the model and therefore it is important to provide small but correct models when developing subsystems to reduce development time.

5. RELATED AND FUTURE WORK

The FBS methodology [37] is considered by many researchers and practitioners the most adequate approach for developing complex systems from their functional descriptions [7, 12, 39, 38]. The FBS specifies three orthogonal design layers representing functions, behavior, and state of the system as shown in Figure 1. The function layer[7] describes the design intentions, functional de-

[7]In this paper, we refer to "*functional modeling*" as the manipulation of the function layer.

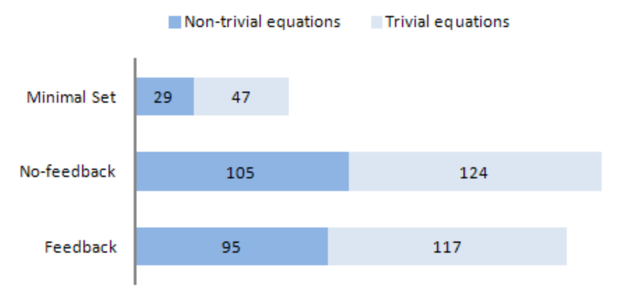

Figure 10: The number of non-trivial equations determines the complexity of the simulation and its underlying mathematical model (less is better). Generating the minimal simulation for subsystem development reduces the simulation time while providing correct inputs.

composition, and matter and energy transformations. The function layer corresponds to the "function structure" in [27]. The state, or structure[8], layer defines the physical objects with attributes (e.g. a gear with 10 teeth) that can be used to realize the system functionality. The behavior layer defines physical phenomena as changes of state, or working principles, that describe the functions and the selected structures. Semantic relations bind the three layers (F-S, S-B, and F-B relations) into a comprehensive view of the system. In this paper, we follow the FBS methodology in combination with systems engineering principles to provide a framework for concept design, product representation, and simulation of complex CPS. We strongly believe that such methodology can be also used for highly distribute CPS applications including smart grid, factory automation, and nano-/micro-scale implantable medical devices.

The different nature of the FBS layers require various modeling languages and tools. For example, the Functional Basis language [12, 33] has been successfully implemented in authoring tools for function decompositions in the function layer [5, 15, 30, 39], mechanical and electrical CAD tools have been traditionally used in the structure layer, and software modeling languages [20] and mathematical equation languages are suitable for the behavior layer. Unfortunately, these tools and languages have remained decoupled and incompatible because each layer must be treated separately using specialized tools. Some researchers have proposed the use of SysML [25] as a unifying language for the FBS approach [2, 39, 14]. SysML is an extension of UML for systems engineering and supports the various semantics and requirements of the FBS layers. However, SysML is a modeling language and it is limited to *express* models and does not specify any mechanism to *execute* these models [3]. We believe that executing FBS models is essential for the adoption of this methodology in industry. In this paper, we propose the use of system-level modeling and simulation languages to create executable FBS models.

System simulation languages (e.g. Modelica [22], Simscape [19], VHDL-AMS) allow the modeling and simulation of multi-disciplinary complex systems. Using a component-based approach where behavior is specified according to energy conservation principles, these languages provide the means for multi-disciplinary modeling and simulation where different engineering domains can be combined in the same language and in the same simulation. We argue that this is a suitable abstraction which effectively combines the structure and behavior layers in the FBS modeling approach because these two layers can be unified using the same language rather than using incompatible domain-specific tools. Specifically, we use the Functional Basis language to manipulate the functional layer of FBS representations and Modelica language to provide a consistent representation of structure and behavior layers to obtain simulations. In addition, using automatic generation of Modelica models from functional models, we streamline the association of function and behavior/structure layers of a CPS system model.

We believe that functional modeling-based approaches for control and software modeling and automatic synthesis is an area that needs further research and development. Existing "cyber"-centric (control software and embedded systems) approaches [4, 11, 31] cover only the behavioral and structural aspects of control software but lack support for high-level functional aspects. More importantly, these cyber-centric methodologies use a rather simplistic approach for modeling the physical part of a CPS. Our methodology, on the other hand, starts from the functional aspects of the system and therefore it enables a top-down design approach for the software, control, and physical aspects of the CPS. From this observation, our methodology attempts to bridge the gap between cyber-centric and physical-centric modeling tools in order to create a truly holistic CPS development methodology. Our practical approach serves as a rapid prototyping methodology that control software engineers can use to generate new *plants* (In control theory, a "plant" is defined as the physical system under control). Thanks to our functional modeling approach and synthesis algorithm, designers and engineers can easily communicate their ideas across disciplines, make physical changes to the system, and automatically synthesize new plants to validate the control and software logic through system-level simulation.

6. CONCLUSION

This paper presents a novel simulation-enabled FBS approach for the integrated development of emerging complex CPS. The enabling technology is the utilization of physical modeling languages (i.e. Modelica) to capture the behavioral-structural aspects of a system simultaneously while maintaining the functional description as a separate entity to facilitate the cross-discipline communication and engineering mechanism. We argue that functional models are inherently incomplete and propose a context-sensitive synthesis method to construct an appropriate engineering context that facilitates the selection of function-to-component mappings by considering the surrounding flows of a function. In addition, we introduce the concept of a `recycle` function to capture feedback flows that are essential for complex systems using control, software, sensors, and actuators. Using an electro-mechanical CPS design use-case, we demonstrate how our methodology can be used to support system-level design through the creation of high quality simulation models, and we also demonstrate a more efficient subsystem development through the creation of minimal simulation models.

7. ACKWNOWLEDGEMENTS

We thank Andres Alvarez Cabrera, Tetsuo Tomiyama, Benjamin Kruse, Clemens Muenzer, and Kristina Shea for their valuable insights on the FBS methodology and functional modeling. We thank Georg Muenzel, George Lo, Rainer Wasgint, and the team of the Lighthouse Project Mechatronics Design at Siemens Corporate Technology for their support and valuable feedback.

8. REFERENCES

[1] Aberdeen Group. System design: New product development for mechatronics. January 2008.

[8]In practice [10], state and structure are used as synonyms when referring to the FBS layers. In this paper, we use "structure" because in simulation a "state" refers to a time-dependent variable of a dynamic system.

[2] A. A. Alvarez Cabrera, M. S. Erden, and T. Tomiyama. On the potential of function-behavior-state (FBS) methodology for the integration of modeling tools. In *Proc. of the 19th CIRP Design Conference âĂŞ Competitive Design*, pages 412–419, 2009.

[3] L. Bassi, C. Secchi, M. Bonfe, and C. Fantuzzi. A SysML-based methodology for manufacturing machinery modeling and design. *Mechatronics, IEEE/ASME Transactions on*, 16(6):1049 –1062, Dec 2011.

[4] A. Bhave, D. Garlan, B. Krogh, and B. Schmerl. Multi-domain modeling of cyber-physical systems using architectural views. In *Proc. of the Embedded Real Time Software and Systems Conference (ERTS 2010)*, 2010.

[5] C. R. Bryant, R. B. Stone, D. A. Mcadams, T. Kurtoglu, and M. I. Campbell. Concept generation from the functional basis of design. In *Proc. of International Conference on Engineering Design, ICED 2005*, pages 15–18, 2005.

[6] J. E. Byrd and M. J. Perona. Kinetics of popping of popcorn. *Cereal Chem.*, 82(1):53–59, 2005.

[7] A. A. Cabrera, K. Woestenenk, and T. Tomiyama. An architecture model to support cooperative design for mechatronic products: A control design case. *Mechatronics*, 21(3):534 – 547, 2011.

[8] F. E. Cellier. *Continuous System Modeling*. Springer-Verlag, 1991.

[9] D. Dumbacher and S. R. Davis. Building operations efficiencies into nasa's ares i crew launch vehicle design. In *54th Joint JANNAF Propulsion Conference*, 2007.

[10] M. Erden, H. Komoto, T. V. Beek, V.D'Amelio, E. Echavarria, and T. Tomiyama. A review of funcion modeling: approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 22:147–169, 2008.

[11] D. Goswami, R. Schneider, and S. Chakraborty. Co-design of cyber-physical systems via controllers with flexible delay constraints. In *Proc. of the 16th Asia and South Pacific Design Automation Conference*, ASPDAC 2011, pages 225–230, 2011.

[12] J. Hirtz, R. B. Stone, S. Szykman, D. A. McAdams, and K. L. Wood. A functional basis for engineering design: Reconciling and evolving previous efforts. Technical report, NIST, 2002.

[13] H. Komoto and T. Tomiyama. A framework for computer-aided conceptual design and its application to system architecting of mechatronics products. *Comput. Aided Des.*, 44(10):931–946, Oct. 2012.

[14] B. Kruse, C. Muenzer, S. Wolkl, A. Canedo, and K. Shea. A model-based functional modeling and library approach for mechatronic systems in SysML. In *ASME 2012 Intl. Design Engineering Technical Conferences (IDETC)*, 2012.

[15] T. Kurtoglu and M. I. Campbell. Automated synthesis of electromechanical design configurations from empirical analysis of function to form mapping. *Journal of Engineering Design*, 19, 2008.

[16] Lawrence Berkeley National Laboratory - Modelica Buildings Library. `http://simulationresearch.lbl.gov/modelica`.

[17] E. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing, ISORC 2008, 11th IEEE International Symposium on*, pages 363 –369, May 2008.

[18] I. Lee, O. Sokolsky, S. Chen, J. Hatcliff, E. Jee, B. Kim, A. L. King, M. Mullen-Fortino, S. Park, A. Roederer, and K. K. Venkatasubramanian. Challenges and research directions in medical cyber-physical systems. *Proceedings of the IEEE*, 100(1):75–90, 2012.

[19] Mathworks. Simscape. `http://www.mathworks.com/products/simscape/`.

[20] S. J. Mellor and M. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[21] Oregon State University, Design Engineering Lab, Design Repository. `http://designengineeringlab.org/`.

[22] Modelica Association, Modelica. `https://modelica.org/`.

[23] Modelica Association, Modelica Standard Library. `https://modelica.org/libraries/Modelica/`.

[24] Modelon - Vehicle Dynamics Library. `http://www.modelon.com/`.

[25] OMG Systems Modeling Language (SysML). `http://http://www.omgsysml.org/`.

[26] M. Otter, K.-E. Årzén, and I. Dressler. Stategraph—A Modelica library for hierarchical state machines. In *Modelica 2005 Proceedings*, 2005.

[27] G. Pahl, W. Beitz, J. Feldhusen, and K. Grote. *Engineering Design - A Systematic Approach*. Springer, 3rd edition, 2007.

[28] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA, 2010. ACM.

[29] D. L. Rosenband. Hardware synthesis from guarded atomic actions with performance specifications. In *Proc. of the 2005 IEEE/ACM International conference on Computer-aided design*, ICCAD 2005, pages 784–791, 2005.

[30] S. Rudov-Clark and J. Stecki. The language of FMEA: on the effective use and reuse of FMEA data. In *AIAC-13 Thirteenth Australian International Aerospace Congress*, 2009.

[31] A. Sangiovanni-Vincentelli. Quo vadis SLD: Reasoning about trends and challenges of system-level design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[32] P. Schwarz. Physically oriented modeling of heterogeneous systems. *Math. Comput. Simul.*, 53(4-6):333–344, Oct. 2000.

[33] R. B. Stone and K. L. Wood. Development of a functional basis for design. *Journal of Mechanical Design*, 122(4):359–370, 2000.

[34] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang. Toward a science of cyber-physical system integration. *Proceedings of the IEEE*, 100(1):29–44, 2012.

[35] T. Tomiyama, V. D'Amelio, J. Urbanic, and W. ElMaraghy. Complexity of multi-disciplinary design. *CIRP Annals - Manufacturing Technology*, 56(1):185 – 188, 2007.

[36] S. Uckun. Meta ii: Formal co-verification of correctness of large-scale cyber-physical systems during design. Technical report, Palo Alto Research Center, September 2011.

[37] Y. Umeda, H. Takeda, T. Tomiyama, and H. Yoshikawa. Function, behaviour, and structure. *Applications of artificial intelligence in engineering V*, 1:177–194, 1990.

[38] T. J. van Beek, M. S. Erden, and T. Tomiyama. Modular design of mechatronic systems with function modeling. *Mechatronics*, 20(8):850 – 863, 2010.

[39] S. Wölkl and K. Shea. A computational product model for conceptual design using SysML. *ASME Conference Proc.*, 2009(48999):635–645, 2009.