Functional Modeling Compiler for System-Level Design of Automotive Cyber-Physical Systems

Arquimedes Canedo*, Jiang Wan†, Mohammad Abdullah Al Faruque†
*Siemens Corporation, Corporate Technology
arquimedes.canedo@siemens.com
†University of California, Irvine
{jiangwan, alfaruqu}@uci.edu

***Abstract*—A novel design methodology, associated algorithms, and tools for the design of complex automotive cyber-physical systems are presented. Rather than supporting the critical path where most resources are spent, we preemptively target the concept design phase that determines 75% of a vehicle's cost. In our methodology, the marriage of systems engineering principles with high-level synthesis techniques results in a Functional Modeling Compiler capable of generating high-fidelity simulation models for the design space exploration and validation of multiple cyber-physical (ECUs+Physics) vehicle architectures. Using real-world automotive use-cases, we demonstrate how functional models capturing cyber-physical aspects are synthesized into high-fidelity simulation models.**

I. Introduction

Automotive cyber-physical systems (CPS) are some of the most technologically advanced and complex systems currently being produced. Modern cars are no longer mechanically-dominated systems; their physical behavior is greatly influenced by computers (electronic control units or ECUs) and network systems [1] – cyber components. For example, the remarkable advances in fuel economy, safety, performance, etc. have been possible due to the dense array of cooperating cyber components that interact with the physical processes in a car. Designing a modern car with 100 ECUs [2, 3] controlling dozens of complex physical processes is a daunting task that requires the collaboration of hundreds of domain experts from various organizations. Creating design automation tools that improve the automotive design process and allow companies to rapidly add new features, manage the heterogeneity of components, and maintaining the development time and cost targets is an equally challenging task.

State-of-the-art automotive model-based design (MBD) tools have been developed on the principle of *reducing the critical path* with the objective of making the design process shorter, cheaper, and more efficient. The critical path in complex CPS design is the *detailed design* phase [4, 5, 6] where precise engineering specifications are created by the domain experts. Thus, mechanical engineering is supported by computer-aided design (CAD) and engineering (CAE) tools; electrical engineering by electronic design automation (EDA) and wire harness design tools; control engineering by Matlab/Simulink, LabView, LMS, and Modelica; and software engineering by UML and in-house software development environments [7]. Unfortunately, these models cannot be easily – due to model, domain, and tool incompatibility – and effectively – due to performance reasons – combined to perform system-level analyses and simulations [8]. This situation has led to a serious problem because the vehicles' complexity keeps increasing but the design tools cannot answer simple but important questions such as estimating the impact of a change in the transmission ECU in the overall fuel economy of a vehicle. Tools and methodologies that allow ECUs to be developed and tested against system-level and high-fidelity multi-physics systems in model-, software-, and hardware-in-the-loop simulations are needed.

In this paper, we take a novel approach for the creation of automotive design tools. Rather than targeting the critical path, we preemptively support the *concept design* phase. The concept design phase precedes the detailed design phase; despite its short duration, it is where most of the creativity and innovation occurs. A key observation is that about 75% of the cost of a product is dictated by the decisions taken at the concept design phase [9]. Let the requirement for a new car be "*the vehicle shall detect and avoid collision with pedestrians*". During concept design, this requirement is formalized in a *functional model* that defines *what* the system does in terms of energy, material, and signal flows. Functional models provide a high-level abstraction that is used by practitioners to perform a broad design space exploration of various concepts and narrow the design space to the few designs that do satisfy the requirements. For example, the designer may quickly analyze the tradeoffs between various architectures for detecting humans using cameras, sonar, and LIDAR through simulation. Functional models are intimately related to *architecture*, defined as the allocation of physical structures to functionality [10, 11, 12]. From a design automation perspective, the high abstraction level in functional models makes them a suitable formalism for CPS design. Functional models naturally express cyber-physical processes by hiding details of the continuous and discrete dynamics, and naturally allow cross-domain collaboration [5, 6].

This paper provides a novel high-level synthesis methodology that fills the gap between "cyber" (ECUs) and "physical" design that currently exists in the siloed MBD tools. Our methodology enables the high-level synthesis of multi-physics, multi-domain simulation models from functional model specifications. This allows the testing and validation of automotive embedded systems *early* in the design. Using realistic automotive architectures, we demonstrate that our methodology can be used to synthesize high-fidelity CPS models where complex discrete interactions of ECUs with continuous physical processes may be analyzed in detail. This paper's contributions are:

- A high-level synthesis algorithm that translates architecture-mapped functional models to high-fidelity simulation models of automotive CPS.
- A decision support system that uses a functional model refinement process to back-propagate the results of the high-level synthesis as concept design suggestions.
- The automatic extraction of functional information from commercial and academic simulation component libraries.
- An implementation of a Functional Modeling Compiler (FMC) capable of mapping automotive functional models to simulation models using realistic architectures.

II. Background and Related Work

Architecture-based design, also referred to as platform-based design [13] in the EDA community, allows automotive companies to streamline the development process of complex products across different organizations [11]. Automotive architectures (or platforms) are estimated to save billions of dollars annually to companies because they allow reusability of components across different models and brands [14]. As it is discussed in [12], current CPS architecture modeling capabilities have the fundamental shortcoming of not being able to represent physical components and their interactions with cyber components. The authors in [12] propose a CPS architectural style that supports a unified representation of both the physical and cyber components and their interactions.

The key difference between our work and [12] lies in the level of abstraction; while architectures provide a high-level of abstraction, they lock the design to a particular implementation because an architecture is the allocation of specific components to functions of a system. Our novel methodology, on the other hand, uses an even higher level of abstraction provided by functional models that not only allows a unified representation for CPS, but also enables a unified representation for multiple architectures. This characteristic is beneficial during concept design where the design space is broadened in order to analyze and select the solution, and therefore the architecture, that better fulfills the requirements. For this reason, we argue that the methodology presented in [12] is better suited for the detailed design phase when *design decisions have already been made*, and our methodology is better suited for the concept design phase when *multiple designs are being analyzed in order to take the best design decision*.

III. Functional Modeling Compiler (FMC)

The objective of the concept design phase is to identify all potential engineering solutions for an automotive CPS concept, and downselect the ones that fulfill both the requirements (e.g. regulatory) and the constraints (e.g. weight). Our functional modeling-based methodology, as shown in Figure 1, supports the concept design phase by both allowing the requirements to be formalized in a functional model, and by synthesizing system-level simulation models for all potential architectural solutions that satisfy the design intent sketched by the designer in the functional model.

To the best of our knowledge, we are the first to capitalize on the Functional Basis's ability to express multiple domains and disciplines that exist in a CPS, to be meaningful to humans for inter-disciplinary communication, to be a formal abstraction that allows high-level synthesis, to be higher level and therefore more flexible than architecture models during concept design, and to formalize the so far informal functional modeling practice in the automotive industry.

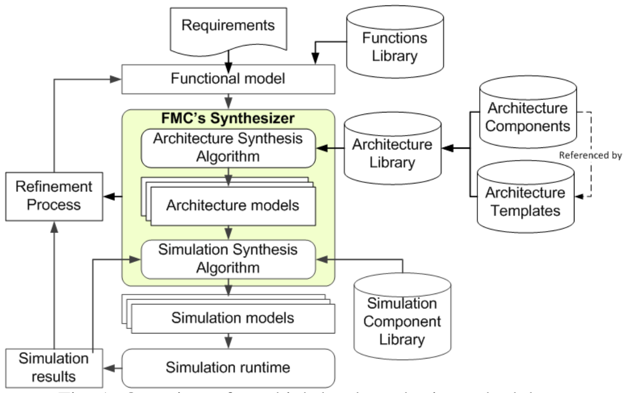


Fig. 1: Overview of our high-level synthesis methodology.

Definition 1. *A **functional model** is a labeled directed multigraph $F = (V, E, s, t, L_V, L_E)$, where each node v_i is a function in this model, each edge $e_{(i,j)}$ represents the flow from v_i to v_j, $s : E \rightarrow V$ and $t : E \rightarrow V$ are two mapping indicating the source and target $v \in V$ of an $e \in E$. 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.*

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 Functions Library consists of the vocabulary, syntax, and semantics defined by the Functional Basis language. Functional requirements are explicitly encoded in the V, E. For example, the functional requirement for an electric car is explicit in a "`convert electrical energy to rotational mechanical energy`" function. Figure 2 shows an exemplary functional model of the cold loop in an automotive HVAC system.

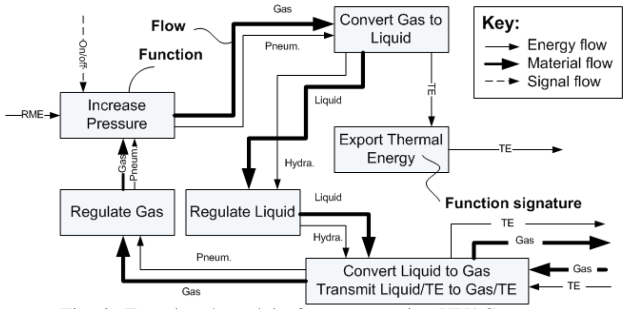


Fig. 2: Functional model of an automotive HVAC system.

Definition 2. *An **architecture component** is a pair $A = (F_{sub}, C_{list})$, where F_{sub} is a functional model, and C_{list} is a list of constraints that specify relevant architectural parameters and properties such as the number of cylinders in an engine, data path width in an ECU, etc.*

Definition 3. *An **architecture template** is a multigraph $ArchT = \{A, C, s, t, Cons\}$, where $a_i \in A$ is an architecture component, $c_{(a_i,a_j)} \in C$ is the the connector connecting component a_i and a_j, $s : C \to A$ and $t : C \to A$ are two mappings indicating the source and target $a \in A$ of an individual $c \in C$, and $Cons = \{cons_1, cons_2, ...\}$ is a list of constraints that this architecture must meet.*

Definition 4. *An **architecture library** $A_{lib} = (A_{set}, ArchT_{set})$ consists of two sub-libraries A_{set} and $ArchT_{set}$, where $A_{set} = (A_1, A_2, ..., A_n)$ is a collection of architecture components A_i, and $ArchT_{set} = (ArchT_1, ArchT_2, ..., ArchT_m)$ is a collection of architecture templates. The architecture components in $A_i \in A_{set}$ may be reused in the architecture templates $ArchT_{set}$ thanks to their object oriented properties.*

Definition 5. ***User given requirements*** *$R = \{req_1, req_2, ...\}$ is a set of requirements expressed in temporal logic that determine the expected system's characteristics. This representation is flexible to express different aspects of various cyber-physical domains and it admits a variety of efficient algorithms for translating a formula into verification code.*

Our FMC uses the Architecture Library (composed by architecture components and templates) to allocate functions to candidate architectures [15]. Architecture libraries similar to ours are used in automotive design to document successful, unsuccessful, and reference designs. They also serve two additional purposes: describe the structure and interfaces between reusable physical components, and describe what functions are fulfilled by these. Figure 3 shows four architecture components in a library for automotive HVAC systems. Notice that the `TXV` and `Condenser` components are defined in terms of the Functional Basis to specify which functionality is fulfilled and what flows (connecting strongly typed ports) are necessary. Our architecture components are object-oriented to facilitate reusability. The `HVAC` architecture is the typical configuration for controlling the air conditioning in the entire cabin [16]. It reuses `TXV`, `Condenser`, `Blower`, and `Evaporator` components (the last two not shown in the figure) and also uses two functions "`Increase Pressure`" and "`Export Thermal Energy`". The `Dual-zone HVAC`, for example, is an alternative architecture to control the air conditioning in the front row seats as well as the back row seats independently [17]; therefore, it requires two `TXV`, two `Evaporators`, and two `Condenser` components instead of one of each.

An important observation is that while the `HVAC` and `Dual-zone HVAC` are structurally different, they are both embodiments of the the same functional model shown in Figure 2 because they fulfill the intended overall functionality of the system. Therefore, during concept design, these two architectures are candidates for the "`Control Climate`" function of a vehicle. In a latter stage, the selection of architectures will be subject to the design constraints and requirements. In this example, the `Dual-zone HVAC` is more expensive as it requires more components but it provides more comfort to the passengers.

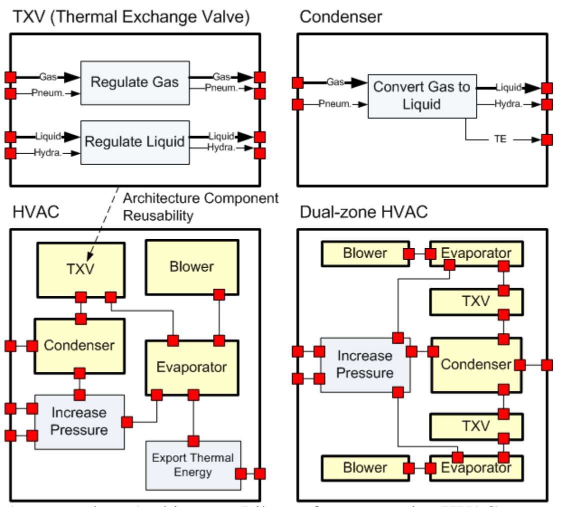


Fig. 3: An exemplary Architecture Library for automotive HVAC systems with components (top row) and templates (bottom row)

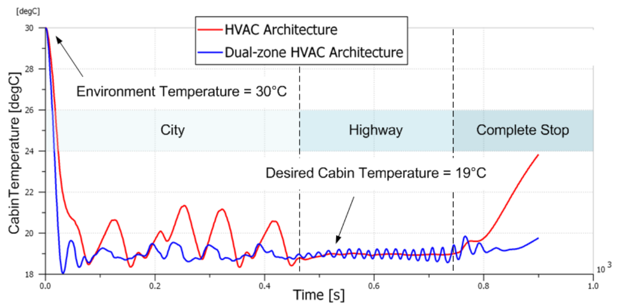


Fig. 4: Trade-off simulation analysis for `HVAC` and `Dual-zone HVAC`.

Notice that an architecture defined in terms of the Functional Modeling basis allows the synthesizer to validate the design contracts [18] because the interfaces are strongly typed and can be mapped to CPS variables (i.e. electrical, mechanical, signals, etc.) [19]. This is beneficial for both mapping functional models to candidate architectures, as well as for mapping simulation components to each candidate architecture.

Definition 6. *The **level of fidelity** is the amount of qualitative and quantitative information that can be obtained from a model.*

Definition 7. ***Incompleteness*** *of a model is the lack of fidelity necessary to answer a specific engineering question.*

Definition 8. ***Refinement*** *is the process by which the level of fidelity of a model is increased to make it less incomplete, and thus the ability to answer more detailed engineering questions.*

Automotive CPS design is a complex iterative process; one design is passed hundreds of times through hundreds of personnel in various organizations. Our algorithm observes this aspect and embraces the fact that functional, architecture, and simulation models are *inherently incomplete* and a continuous refinement is necessary to be able to realize a "napkin concept" into a real vehicle. Automotive design best practices are the key drivers behind the FMC and our ultimate goal is its integration to Product Lifecycle Managment (PLM) systems.

In most cases, the first functional model is a low fidelity model that captures the essence of the design and defines only the basic set of functions F_A to fulfill the high-level requirements. Our synthesis algorithm supports the refinement of such low fidelity functional models by leveraging the company's know-how that exists in the architecture library. For example, after the initial candidate architectures have been identified, the architecture models may contain a set of functions F_B that are not modeled in the original functional model. Thus, this information can be propagated back to create a refined functional model $F_C = F_A \cup F_B$. We refer to this refinement as *bottom-up refinement* because information is being propagated from a lower level of abstraction (architecture) to a higher-level of abstraction (functions) to make it less incomplete.

A *top-down refinement* propagates information from a higher level of abstraction (functions) to a lower level of abstraction (architecture) to increase the level of fidelity. For example, when a functional model F contains a set of functions that are not fulfilled by any of the components in the architecture library A_{lib}, then a new architecture component A_{new} with the relative complement of the functions in the architecture library in the functions in the functional model is created by $A_{new} = F \backslash A_{lib}$. The new architecture component A_{new} is then subject to having a design contract signed-off by an engineer to describe how it relates vertically to other levels of abstraction, horizontally to other architecture components, parameterized according to architectural parameters of interest, and uniquely named and described. Although these suggestions are provided to the designers, our algorithm only uses architecture components that have been signed-off by an engineer and have a design contract in place. This guarantees that the high-level synthesis is performed with the validated models.

A. High-Level Synthesis Technique for Transforming Functional Models to Architecture Models

The high-level synthesis problem for CPS can be formulated as follows: given a functional model F and the user defined requirements R as inputs, the objective is to find the set of architecture templates $ArchT_{mapped} \in A_{lib}$ that *fully* or *partially* map to F.

Definition 9. *A **mapping from a functional model** F **to an architecture template** $ArchT$ is a set of pairs $m = \{< v_i, A_j >\}$, where $v_i \in V(F)$ and $A_j \in ArchT$. To represent all combinations, we define the mapping from F to ArchT as $MAP = \{m_1, m_2, ...\}$. For convenience, we define $A_j = \varnothing$ if v_i does not exist. Furthermore, we define the mapping from F to all $ArchT \in A_{lib}$ as a $MAP_{set} = \{MAP_1, MAP_2, ...\}$.*

Algorithm 1 generates both full and partial mappings MAP_{set} from a functional model F into a set of architecture templates $ArchT_{mapped}$ based on the knowledge contained in the architecture library A_{lib}. Our synthesis algorithm also provides top-down $ArchT_{refined}$ and bottom-up $F_{refined}$ refinements as design suggestions for F and $ArchT_{mapped}$. Algorithm 1 performs a branch and bound synthesis [20] as follows. First, Lines 2-4 prune the selection of architecture templates whose constraints C_{list} do not meet the requirements R. Based on the selected architecture templates, Lines 5-12 aggregate the maximum possible mapping MAP_{set} from F to $ArchT_{mapped}$ based on functions only. In other words, if a function $f \in F$ exists in $A_k \in A_{mapped}$, then the architecture template becomes a candidate solution.

After the architecture template design space has been expanded, our algorithm performs a multi-step pruning process on each architecture template in $ArchT_{mapped}$: (initial steps) Lines 13-17 eliminate from MAP_{set} the architecture templates whose architecture component constraints $C_{list} \in A_l$ violate any of the requirements R; (final steps) Lines 18-26 eliminate from MAP_{set} the architectures whose connectivity do not match the functional connectivity (flows) in F; and eliminates the architecture templates whose architecture components were eliminated entirely by the multi-step pruning process.

Lines 27-30 generate a top-down refinement $A_{refined}$ for all partial mappings; and Lines 31-34 generate the bottom-up refinement $F_{refined}$ based on the functions F_{sub} that exist on the selected $ArchT_{mapped}$ but not in F. Notice that the algorithm aggregates *all* functions in the bottom-up refinement and gives the user the option to filter the refined functionality according to a particular domain. In practice, as the design becomes more detailed, the analysis tends to become more discipline-specific.

Let the total architecture components in the architecture library $A_i \in A_{lib}$ be N, total architecture templates in the architecture library $ArchT_i \in A_{lib}$ be M, and the number of functions in F to be K. The aggregation step in Lines 5-12 has a complexity of $\mathcal{O}(N{*}M{*}K)$. The multi-step pruning process in Lines 13-26 is a search of all the possible mappings and each step has a complexity of $\mathcal{O}(N{*}M{*}K)$. Thus, Algorithm 1 has a complexity of $\mathcal{O}(N{*}M{*}K)$.

B. High-Level Synthesis Technique for Transforming Architecture-mapped Functional Models to Simulation Models

The next step is to generate the corresponding simulation model for each of the candidate architectures identified by Algorithm 1 in order to analyze the actual performance against the expected performance of the system, and to conduct trade-off analyses between the different candidate architectures.

Definition 10. *A **simulation model** S is a strongly typed component with well defined CPS ports $p_i \in P$ and connectors $c_{(p_i,p_j)} \in C$ that obey the energy conservation principles in various domains and allow components to exchange physical energy and data. Ports can be either physical or signal ports. Physical ports have an associated effort variable e – voltage, torque, pressure, temperature – and a flow variable f – current, angular velocity, volumetric flow, heat flow –, such that the product $e \cdot f$ represents physical power*[1]*. Signal ports are represented with numeric variables. Connectors enforce that only compatible port types are connected to each other.*

S encapsulates a description of hybrid behavior – continuous and discrete. Discrete behavior is defined by a 5-tuple:

$$FSM = \{\Phi, \mathcal{I}, \mathcal{O}, \mathcal{U}, \Psi\} \quad (1)$$

where Φ is a finite set of states that may encapsulate continuous behavior, $\mathcal{I}$ is a set of input valuations, $\mathcal{O}$ is a set

[1]The Bond Graph foundations [21] allow the FMC to generate simulation models in various languages such as AMESim [22], Modelica [23], and Simscape [24].

Algorithm 1: Functions to Architecture Templates

```
Input: $F$: A functional model
Input: $R$: User defined requirements
Input: $A_{lib}$: An architecture library
Output: $MAP_{set}$: A set of mappings from $F$ to $ArchT_{set} \in A_{lib}$
Output: $ArchT_{mapped}$: A set of mapped architecture templates
Output: $ArchT_{refined}$: A set of refined architecture templates
Output: $F_{refined}$: A set of refined functional models
1  $ArchT_{mapped} = ArchT_{set} \in A_{lib}$; $MAP_{set} = \varnothing$;
   $ArchT_{refined} = \varnothing$; $F_{refined} = \varnothing$
2  foreach $ArchT_i \in ArchT_{mapped}$ do
3      if $\exists Cons_j(ArchT_i)$ contradicts with any $req_k \in R$ then
4          $ArchT_{mapped} = ArchT_{mapped} \setminus ArchT_i$
5  foreach $ArchT_i \in ArchT_{mapped}$ do
6      $MAP_i = \varnothing$
7      foreach $v_j \in F$ do
8          $MAP_i = MAP_i \times \{< v_j, \varnothing >\}$
9          foreach $A_k \in ArchT_i$ do
10             if $\exists \{f_l \in F_{sub}(A_k)\} = \{l(v_j) \in F\}$ then
11                 $MAP_i = MAP_i \times \{< v_j, A_k >\}$
12     $MAP_{set} = MAP_{set} \cup MAP_i$
13 foreach $MAP_i \in MAP_{set}$ do
14     foreach $m_j \in MAP_i$ do
15         foreach $< v_k, A_l > \in m_j$ do
16             if $\exists C_{list}(A_l)$ contradicts with any $req_m \in R$ then
17                 $MAP_i = MAP_i \setminus m_j$
18     foreach $m_j \in MAP_i$ do
19         foreach $e_k \in E(F)$ do
20             $v_s = s(e_k) \in V(F)$;$v_t = t(e_k) \in V(F)$
21             Find $A_t$ in $< v_t, A_t > \in m_j$ and $A_s$ in $< v_s, A_s > \in m_j$
22             if $A_s \neq A_t$ and $\nexists c_{(s,t)} \in C(ArchT_i)$ then
23                 $MAP_i = MAP_i \setminus m_j$
24     if All mappings in $MAP_i =< v_k, \varnothing >$ then
25         $ArchT_{mapped} = ArchT_{mapped} \setminus ArchT_i$
26         $MAP_{set} = MAP_{set} \setminus MAP_i$
27     foreach $m_j \in MAP_i$ do
28         foreach $< v_k, A_l > \in m_j$ do
29             if $A_l = \varnothing$ then
30                 $ArchT_{refined} = ArchT_{refined} \cup \{ArchT_i \cup v_k\}$
31 foreach $ArchT_i \in ArchT_{mapped}$ do
32     foreach $\exists f_{subj} \in F_{sub}(A \in ArchT_i)$ do
33         if $\exists f_{subj} \notin F$ then
34             $F_{refined} = F_{refined} \cup \{F \cup f_{subj}\}$
35 return $MAP_{set}$, $ArchT_{mapped}$, $ArchT_{refined}$ and $F_{refined}$
```

of output valuations, $\mathcal{U}$ is an update function that indicates a transition from a state and an input valuation to the next state and an output valuation, and Ψ is the initial state.

Continuous behavior is represented by a differential algebraic equation in the form of:

$$G(\dot{x}(t), x(t), z(t), u(t)) = 0 \qquad (2)$$

where $x(t)$ is a vector of the time varying states variables of the system (e.g. velocity, temperature), $\dot{x}(t)$ the time varying state derivatives, z(t) the algebraic variables, and u(t) the inputs to the system.

***Simulation parameters** $param_i(S)$ represent structural (i.e. component-dependent) and environmental (e.g. gravity) parameters. A **simulation library** $S_{lib} = (S_1, S_2, S_3, ...)$ is a collection of simulation components S_i.*

Although several commercial and academic simulation component libraries exist [22, 24, 25, 23], as of today, none is ready for high-level synthesis; they describe behavior explicitly in terms of a mathematical model, but do not explicitly describe the functionality the component fulfills. Therefore, any top-down approach that attempts to allocate simulation components to architectures and functions faces the same challenge of lack of functional information. There are two ways to overcome this problem.

The first and most straightforward approach is to manually annotate components with a list of functions it fulfills. These semantic annotations regarding individual simulation components $\sigma(S_i)$ are useful because the engineers know best what functionality is achievable with a particular component. The drawback of this approach is the extensive amount of time it takes to manually annotate models, especially in simulation component libraries with thousands of components. Furthermore, the manual approach is not systematic and this leads to few functional annotations per component.

Therefore, we use a novel automated approach for extracting functionality from simulation components. We perform an automatic interface analysis on port types, a structural (i.e. equational and algorithmic) analysis on the mathematical model, and a parameter analysis on the component's parameters to determine what functionalities it fulfills. Note that the functional extraction is performed once per library and the identified functions are automatically annotated in the library as additional semantic information for future reuse. Although we performed functional extraction on the AMESim library [22], this technique is applicable to any other physical modeling language that uses effort-flow variables and differential equations such as Modelica [23] and Simscape [24].

Definition 11. *A **mapping from an architecture-mapped functional model to simulation components** is a set of 3-tuples $t = \{< v_i, A_j, S_k >\}$, where $v_i \in V(F)$, $A_j \in Arch$, and $S_k \in S_{lib}$.*

Algorithm 2 generates simulation models for all the architecture-mapped functional models identified by Algorithm 1 in three steps. Step 1, Lines 2-15, broadens the simulation design space $S_{solution}$ for each of the architecture templates $A_l \in MAP_{set}$. The temporary variable $S_{matches}$ is a pruned set of simulation components from S_{lib} that fulfills the functionality specified in $< v_k, A_l >$. Lines 6-10 perform a search on the simulation library S_{lib} to identify the most appropriate components that match the interface $S_{interface}$, energy transformations $S_{equations}$, parameters $S_{parameter}$, and the manually annotated functional information $S_{functional}$. Notice that port matching is possible because the physical domains in the Functional Basis language can be mapped to the port types in the simulation components (e.g. rotational mechanical energy $\rightarrow$ torque/angular velocity. The matched simulation components $S_{matches}$ are then used to create the 3-tuples that define the mappings of architecture-mapped functional models to simulation components, and are added to the simulation design space $S_{solution}$. Whenever an architecture-mapped function cannot be mapped to any simulation component, we construct a top-down refinement of simulation models $S_{refined}$. Step 2, Lines 16-17, expands the simulation design space S_{set} by creating individual simulation models for all possible combinations of simulation components that match an architecture-mapped functional model. Step 3, Lines 18-25, adds connections to every model in S_{set} according to the topology in their corresponding architecture component. Ports $P(S_i)$ without connections are connected to the appropriate sources and sinks simulation components (e.g. constants, terminals, grounds, etc.); these ports and their new connections to sources and sinks are used to generate the bottom-up refinement in $ArchT_{refined}$ and $F_{refined}$.

Let the candidate architecture templates $ArchT_{set} \in MAP_{set}$ be N, and the number of simulation components in S_{lib} be M. Finding matches from architecture components $A_l \in ArchT_{set}$ to simulation components $S_{matches}$ – Lines 4-15 – has a linearithmic time complexity $\mathcal{O}(NlogM)$ because the simulation library has been pre-processed and the functional information has been already extracted and organized for efficient search; pre-processing has a linear time complexity $\mathcal{O}(M)$. The topology construction in Lines 18-25 has a time complexity proportional to $\mathcal{O}(C(ArchT_{set}) * S_{set})$. In practice, $C(ArchT_{set}) \approx N$ and $M >> S_{set}$; therefore, the time complexity of Algorithm 2 is $\mathcal{O}(NlogM)$. The simulation design space S_{set} created by Lines 16-17 has a space complexity of $\mathcal{O}(N * N_{A_l} * M_{S_{matches}})$.

Algorithm 2: Architecture-mapped Functions to Simulation

```
    Input: $F$: A functional model
    Input: $R$: User defined requirements
    Input: $S_{lib}$: A pre-computed simulation library with semantic annotations
    Input: $A_{lib}$: An architecture library
    Input: $MAP_{set}$: A set of mappings from $F$ to $ArchT_{set} \in A_{lib}$
    Output: $F_{refined}$: A set of refined functional models
    Output: $ArchT_{refined}$: A set of refined architecture templates
    Output: $S_{refined}$: A set of refined simulation components
    Output: $S_{set}$: A set of simulation models
 1  $S_{set} = S_{refined} = ArchT_{refined} = F_{refined} = \varnothing$
 2  foreach $MAP_i \in MAP_{set}$ do
 3      $S_{solution} = \varnothing$
 4      foreach $m_j \in MAP_i$ do
 5          foreach $< v_k, A_l > \in m_j$ do
 6              $S_{interface}$ = find $P(A_l)$ in $P(S_{lib})$
 7              $S_{equations}$ = find $P(v_k)$ in $P(S_{lib})$
 8              $S_{parameter}$ = find $C_{list}(A_l)$ in $param(S_{lib})$
 9              $S_{functional}$ = find $l(v_k), l(s(v_k)), l(t(v_k))$ in $\sigma(S_{lib})$
10              $S_{matches} =$
                $S_{interface} \cap S_{equations} \cap S_{parameter} \cap S_{functional}$
11              if $S_{matches} = \varnothing$ then
12                  $S_{refined} = S_{refined} \cup < v_k, A_l, \varnothing >$
13              else
14                  foreach $S_m \in S_{matches}$ do
15                      $S_{solution} = S_{solution} \cup < v_k, A_l, S_m >$
16  foreach $s_k \in S_{solution}$ do
17      $S_{set} = S_{set} \times s_k$
18  foreach $S_i \in S_{set}$ do
19      $\forall c(i,j) \in C(Arch(S_i)) \rightarrow c(i,j) \in C(S_i)$
20      foreach $P(S_i) = \varnothing$ do
21          $S_{sources}$ = find $P(S_i)$ in $P(S_{lib})$
22          $S_{sinks}$ = find $P(S_i)$ in $P(S_{lib})$
23          $P(S_i) = P(S_i) \cup S_{sources} \cup S_{sinks}$
24          $P(S_i) \rightarrow P(ArchT_{refined}(S_i))$
25          $P(S_i) \rightarrow P(F_{refined}(S_i))$
26  return $S_{set}$, $S_{refined}$, $ArchT_{refined}$ and $F_{refined}$
```

C. Automatic Extraction of Functions from Simulation Libraries

Our automatic approach for determining the set of functions achieved by a simulation component is realized by analyzing its ports and interface, as well as determining the flow (energy, material, signals) transformations that occur within the component's internal structure. This bottom-up step classifies components in a library of simulation components S_{lib} according to the functions they perform. This classification information is important in order to achieve a correlation between functions, architectures, and simulation components that design tools can use to synthesize simulation models for candidate product architectures.

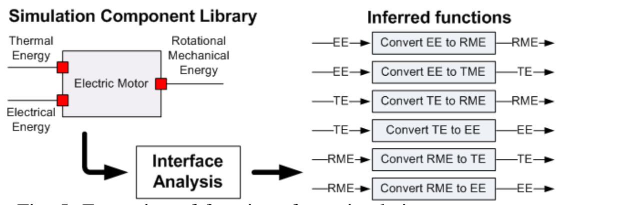


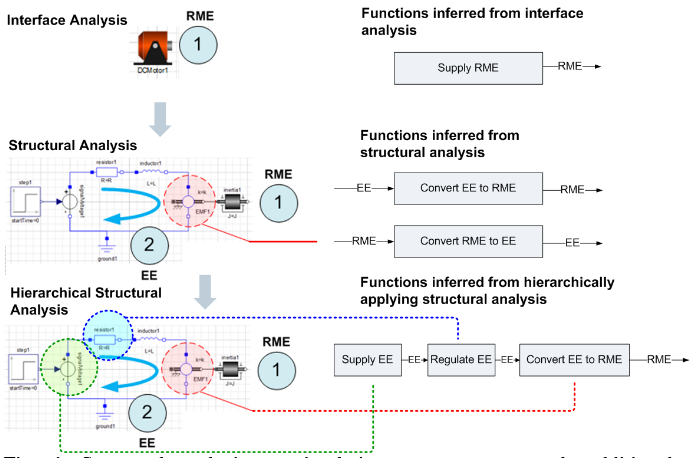
Fig. 5: Extraction of functions from simulation components.

Analyzing the components' interface determines what type of energy, material, or signals can be exchanged with other components through its ports. Figure 5 shows how the interface analysis infers the functions achieved by a given simulation component. For example, the `Electric Motor` component has three ports: thermal, electrical, and rotational mechanical energy. Typically, simulation components exchange energy through a pair of physical variables known as conjugate variables that represent effort/flow. The conjugate variables voltage/current represent electrical energy, torque/angular velocity represent rotational mechanical energy, and temperature/heat flow represent thermal energy. Because the relationship between energy at the functional level, and energy at the simulation component level is known and well defined, our algorithm determines that the `Electric Motor` is able to perform the function of "`Convert`" energy from any port to the other ports. Thus, a total of six functions are inferred.

In some cases, simulation components are modeled with additional domains *that are not visible through their interface* and therefore our algorithm performs a structural analysis – a hierarchical traversal and flattening – of the internal simulation component structure to expose all the domains (e.g. electrical, mechanical, thermal, signals, etc.). Figure 6 shows the interface of the `DCMotor1` component of the Modelica Standard Library with a single port (Rotational Mechanical Energy or RME). The interface analysis infers a single function "`Supply RME`". However, the structural analysis exposes the internal composition of the `DCMotor1` component in terms of two domains: `RME` and `EE`. Notice that although electrical components exist, they are not exposed in the component interface and performing an interface analysis would only lead to a correct, but not very detailed functionality of the simulation component. Therefore, using the structural analysis our algorithm identifies two additional functions "`Convert EE to RME`" and "`Convert RME to EE`".

A further level of structural analysis, referred to as hierarchical structural analysis, is shown in Figure 6. By applying the same principle to the electrical components in the `DCMotor1`, a graph of functions can be inferred by the algorithm. For example, the `signalVoltage1` is connected to `Resistor1`, and this is connected to `EMF1` simulation components. Using the same structural and interface analysis on the individual components, our algorithm determines the flow of electrical energy as a graph of functions where "`Supply EE`", "`Regulate EE`", and "`Convert EE to`

RME" functions are connected. This provides a more detailed functional understanding of a component that can be used to drive the synthesis of architecture-mapped functional models according to different levels of fidelity.



Interface Analysis
RME
1
Functions inferred from interface analysis
Supply RME
Structural Analysis
Functions inferred from structural analysis
Convert EE to RME
Convert RME to EE
2
EE
Hierarchical Structural Analysis
Functions inferred from hierarchically applying structural analysis
Supply EE
Regulate EE
Convert EE to RME

Fig. 6: Structural analysis on simulation components reveals additional domains and the energy, material, signal transformations between them.

IV. Case Study

An exemplary functional model of a car is shown in Figure 7. It inputs three material flows (bold arrows for Air, People, and Fuel) and outputs two material flows (People and Hot Gas) and one energy flow (arrow representing translational mechanical energy or TME). Notice that the functions (blocks) and flows (arrows) can be naturally mapped to the main subsystems of a car represented by architectural templates shown in Figure 8. For example, the function "convert chem. energy to rot. mech. energy" function can be mapped to an Engine ICE, Series Hybrid, and Parallel Hybrid architectures. Similarly, the function "transmit rot. mech. energy" is mapped to a "Manual Transmission" and not to an "Automatic Transmission" because there is a "Gear Selection" control flow flowing from "convert human energy" function to "convert chem. energy to rot. mech. energy" function.

In this case study, the user defined requirements R are given in Table I, and the constraints C_{list} for five vehicle architecture templates are given in Table II. Notice that although $< c1, c2 >$ of all architectures in Table II comply with $< r1, r2 >$, the $c3$(Electric Fuel Cell) violates $r3$ and therefore this is eliminated from the architectural design space by Algorithm 1. The mapping of functions in the functional model in Figure 7 to the remaining four architecture templates in Figure 8 eliminates the Electric architecture template from the candidate list because neither "Store EE" nor "Convert EE to RME" functions exist in the functional model. Notice that while the ICE architecture template fully matches the functional model, the Series Hybrid and Parallel Hybrid templates partially match the functional model. The functions inherited from the ICE template exist in the functional model, the additional functions inherited from the Electric architecture do not exist in the functional model and therefore *partially* match to F.

Using functional models, the design intent remains technology-independent and the FMC allows non-experts to automatically generate simulation models for various architectures that satisfy the design requirements. For example, the functional model in Figure 7 is created by a software engineer with marginal understanding of mechanical engineering principles who wanted to design the engine control system (ECU and its control software) represented by the "Sense" and the "Control" functions. Figure 9 shows the AMESim simulation model for the Series Hybrid architecture synthesized by the FMC. Notice that our high-level synthesis algorithm appropriately downselects the VCU-SH ECU simulation component to control both the ICE Engine and the Electric motor as per the topology and port structure defined in the Series Hybrid architecture template in Figure 8.

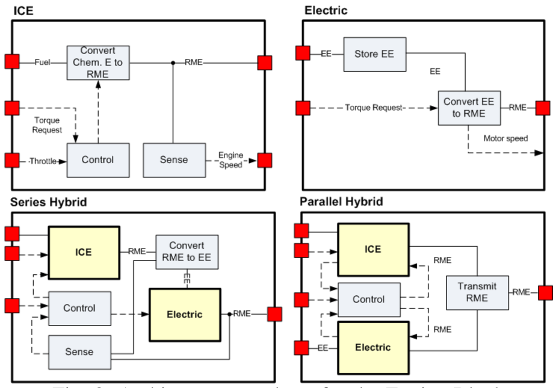


Fig. 8: Architecture templates for the Engine Block

The synthesized CPS simulation models enable domain engineers to validate their subsystems in software- and hardware-in-the-loop simulations, and to perform performance comparisons across various architectures. For example, Figure 10 uses the synthesized models for the Series Hybrid and the Parallel Hybrid to compare the state of charge of the batteries during the first 600 seconds of the New European Driving Cycle (NEDC). Notice that the "Convert RME to EE" function in the Series Hybrid architecture in Figure 8 is mapped to the Electric Generator simulation component in Figure 10. This component allows the Series Hybrid architecture to increase the state of charge of the battery from 90% to 96%. In contrast, the Parallel Hybrid architecture does not specify the functionality for recharging the battery and therefore the state of charge is reduced from 90% to 62%.

The partial mapping of the Series Hybrid architecture also helps to illustrate the bottom-up refinement capabilities of our methodology. The functions "Convert RME to EE", "Convert EE to RME", and "Store EE" in the Series Hybrid architecture do not exist in the functional model in Figure 7. As shown in Figure 11, Algorithm 1 propagates this information to the functional abstraction and creates a refined functional model with a flow from the "RME" output of the ICE-mapped "Convert CE to RME" function to a newly created functions "Convert RME to EE" and "Store EE" that do not exist in the original F. Notice that thanks to Algorithm 1, this connection also exists in the simulation model in Figure 9. We introduced the

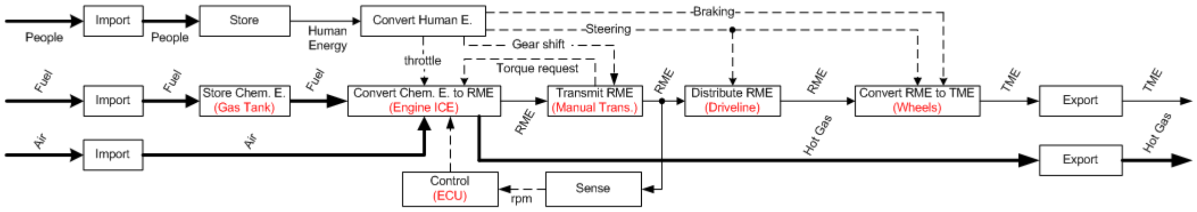


TABLE I: User defined requirements R

ID	Description	Value
$r1$	Minimum energy efficiency	20%
$r2$	Maximum weight	2000KG
$r3$	Maximum price	$40,000

TABLE II: Constraint list C_{list} of architecture templates $Arch$

ID	Description	ICE	Series Hybrid	Parallel Hybrid	Electric	Electric Fuel Cell
c_1	Max. Energy efficiency	20%	37%	37%	60%	50%
c_2	Min. Total weight	2000 kg	2000 kg	2000 kg	2000 kg	2000 kg
c_3	Min. Price	$20,000	$30,000	$30,000	$30,000	$50,000
c_4	Input Energy	Chemical	Chem., Electric	Chem., Electric	Electric	Electric
c_5	Output Energy	Mechanical	Mechanical	Mechanical	Mechanical	Mechanical
c_6	Energy Price	$3 per gallon	$0.23 per kWh	$0.23 per kWh	$0.23 per kWh	$0.15 per kWh

Fig. 7: An exemplary functional model of a car, associated requirements, and architectural constraints.

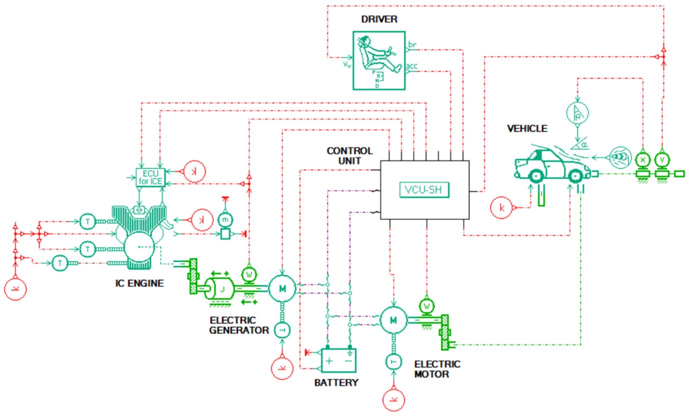


Fig. 9: `Series Hybrid` architecture synthesized simulation model.

concept of top-down and bottom-up refinement of functional, architecture, and simulation models based on the premise that these models will be always incomplete, and each design iteration refines them into higher-fidelity models capable of answering more detailed questions. Therefore, our approach is complementary to platform-based design [13] and the *meet-in-the-middle* principle is satisfied.

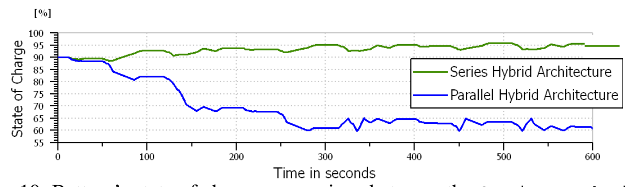


Fig. 10: Battery's state of charge comparison between the `Series Hybrid`, and `Parallel Hybrid` architectures on the New European Driving Cycle.

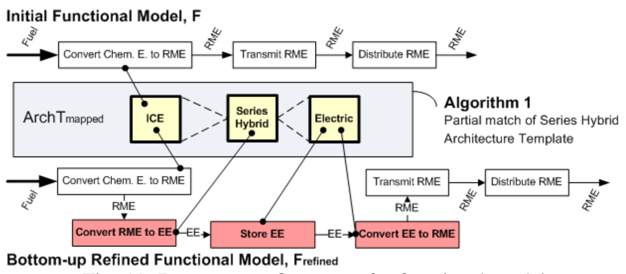


Fig. 11: Bottom-up refinement of a functional model

V. Conclusion

This paper uses the Functional Basis language to model the functionality and the requirements of automotive CPS for increased performance and efficiency. High-level synthesis techniques are used to develop alternative architecture-specific simulation models for such complex automotive CPS and facilitate the early design space exploration during the concept design phase. Moreover, during the synthesis process, a decision support system including functional model refinement to back-propagate the results as concept design suggestions is demonstrated. A real-world use case using various automotive platforms on a commercial simulation platform AMEsim is shown.

References

[1] "AUTOSAR Automotive Open System Architecture," http://www.autosar.org/.

[2] U. Abelein, H. Lochner, D. Hahn, and S. Straube, "Complexity, quality and robustness - the challenges of tomorrow's automotive electronics," in *Design, Automation Test in Europe Conference (DATE), 2012*, 2012, pp. 870–871.

[3] R. N. Charette, "This car runs on code," in *IEEE Spectrum*, February 2009.

[4] Aberdeen Group, "System design: New product development for mechatronics," January 2008.

[5] H. Komoto and T. Tomiyama, "A framework for computer-aided conceptual design and its application to system architecting of mechatronics products," *Comput. Aided Des.*, vol. 44, no. 10, pp. 931–946, Oct. 2012.

[6] S. Uckun, "META II: Formal Co-Verification of Correctness of Large-Scale Cyber-Physical Systems During Design," Palo Alto Research Center, Tech. Rep., September 2011.

[7] M. Broy, I. H. Krüger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, 2007.

[8] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[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] A. Kossiakoff, W. Sweet, S. Seymour, and S. Biemer, *Systems Engineering Principles and Practice*, 2nd ed. Wiley, 2011.

[11] M. Broy, M. Gleirscher, P. Kluge, W. Krezner, S. Merenda, and D. Wild, "Automotive architecture framework: Towards a holistic and standardised system architecture description," TUM, Tech. Rep., 2009.

[12] A. Bhave, B. H. Krogh, D. Garlan, and B. Schmerl, "View consistency in architectures for cyber-physical systems," in *Cyber-Physical Systems (ICCPS)*, 2011, pp. 151–160.

[13] A. Sangiovanni-Vicentelli, "Defining platform-based design," in *EEDesign of EETimes*, 2002.

[14] J. B. Dahmus, J. P. Gonzalez-Zugasti, and K. N. Otto, "Modular product architecture," *Design Studies*, vol. 22, no. 5, pp. 409 – 424, 2001.

[15] A. Canedo, M. A. A. Faruque, and J. H. Richter, "Multi-disciplinary integrated design automation tool for automotive cyber-physical systems," in *Design, Automation and Test in Europe (DATE'14)*, 2014, pp. 1–2.

[16] S. Kang, S. Byon, Y. Seo, and Y. Kim, "Air conditioner for vehicle," US 7,055,591 B2, 06 2006.

[17] J. Zheng and P. S. Kadle, "Dual evaporator air conditioning system and method of use," US 6,983,793 B2, 01 2006.

[18] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems." *Eur. J. Control*, vol. 18, no. 3, pp. 217–238, 2012.

[19] A. Canedo, E. Schwarzenbach, and M. A. A. Faruque, "Context-sensitive synthesis of executable functional models of cyber-physical systems," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ser. ICCPS '13, 2013, pp. 99–108.

[20] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 301–318, 1990.

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

[22] "LMS Imagine.Lab AMESim," http://www.lmsintl.com/, 2013.

[23] "Modelica Association, Modelica," https://modelica.org/, 2013.

[24] MathWorks, "Simscape," http://www.mathworks.com/, 2013.

[25] "Modelon - Vehicle Dynamics Library," http://www.modelon.com/.