Development of Building Automation and Control Systems

Yang Yang, Qi Zhu, Mehdi Maasoumy, and Alberto Sangiovanni-Vincentelli
University of California

Editors' notes:
This article addresses the challenge of realizing the building automation and control system using a distributed network of embedded computers. A specification methodology and design space exploration framework are proposed to raise the level of abstraction at which building control systems are designed, to reduce design effort, and to lower implementation cost.
—Yuvraj Agarwal, University of California, and Anand Raghunathan, Purdue University

THE BUILDING STOCK in the United States accounts for 39% of total energy consumption and 68% of electricity consumption. Limits on carbon emissions are driving new regulations that will require buildings to be energy efficient according to standards that are likely to be more stringent than the ASHRAE 90.1. The design of low energy green buildings—zero energy in the ideal case—is very challenging. There are examples of zero energy buildings today, however, they are the results of *ad-hoc* designs that are not easy to generalize.

The design methodology used today for large buildings is top-down. Different subsystems are designed in isolation by domain experts following design documents flown down after the bid process. This methodology is not suitable for low energy buildings that require interaction among architects, mechanical engineers and control engineers. Consider for instance adopting low energy solutions such as natural ventilation and active facade. In this case, architectural design (e.g., building orientation), the design of the mechanical equipments of the HVAC system and the design of the control algorithms cannot be done in isolation. In this new context, the design of the building automation system (including the embedded processors, the networks supporting the building operations, and the software running on them) is nontrivial. Control algorithms become multi-input, multi-output, hybrid, and predictive, as opposed to single-input single-output controllers coordinated by simple switching conditions as of today. Moreover, several subsystems such as HVAC, lighting, fire and security, and vertical transportation will interact through the network to allow information sharing.

To address these challenges, we propose a design flow for building automation systems that focuses on two main aspects—*heterogeneity* and *automation*. The flow bridges the gap between a desirable design entry point—at a high abstraction level using model-based design tools such as Simulink [4] and Modelica [3]—and the available back-end tools. The flow enables the integration of *heterogeneous input models* from different high-level languages, allowing the interaction between domain experts. It also *automatically* optimizes the implementation of the control algorithms on a distributed platform by selecting computation and communication resources, and by performing software synthesis while

Digital Object Identifier 10.1109/MDT.2012.2201130
Date of publication: 24 May 2012; date of current version: 05 October 2012.

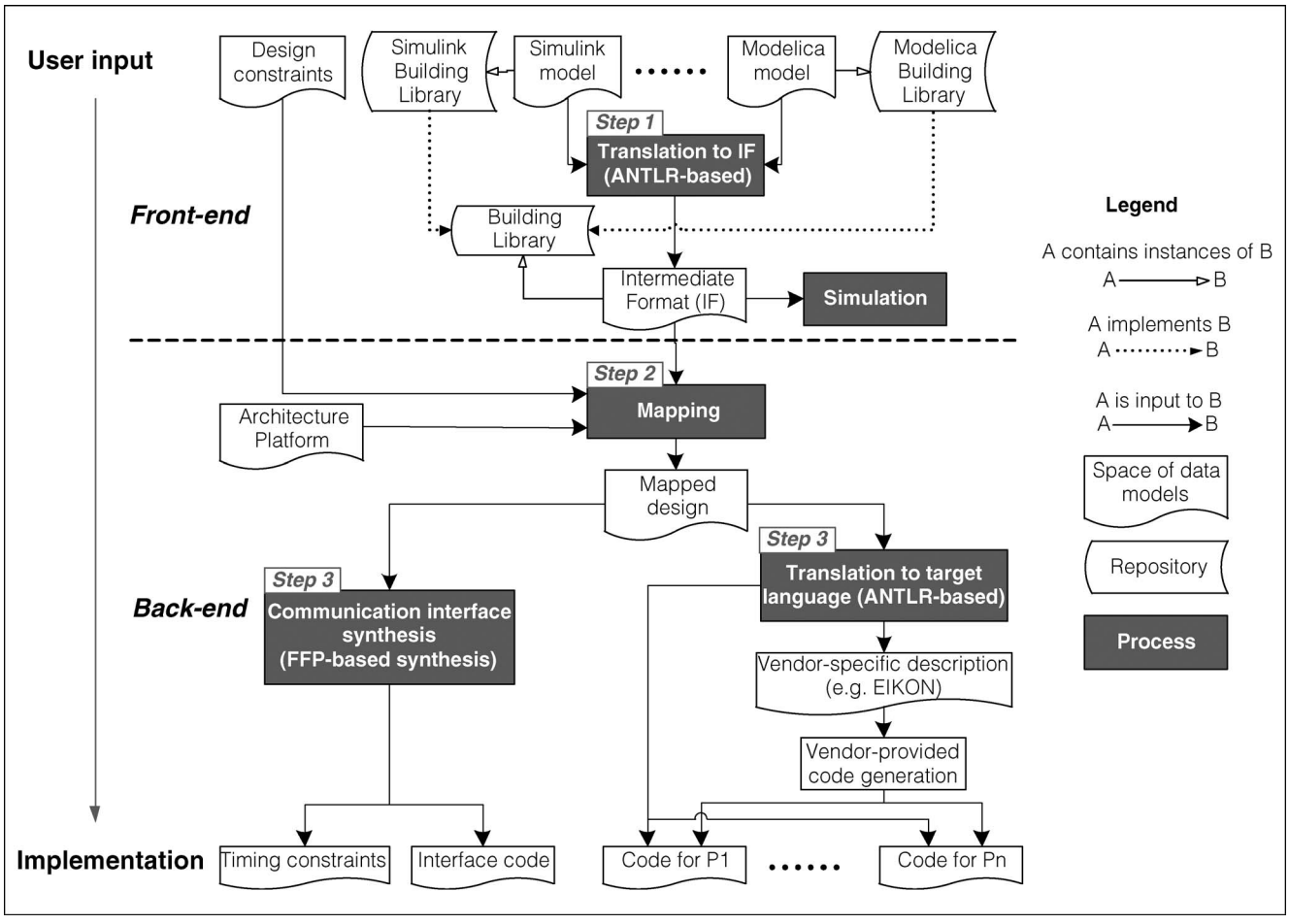

Figure 1. Design flow for building automation and control systems.

meeting the specification and reducing the communication load.

BAC design flow

The proposed design flow consists of a front-end and a back-end, as shown in Figure 1. The front-end is used to model the system including the control algorithms and the behavior of the environment. The back-end includes a set of tools that, given the specification of the control algorithms and a set of available resources, automatically refines the specification into an optimal distributed implementation. The front-end and the back-end exchange models using an *intermediate format (IF)*. The introduction of this intermediate layer is essential for the integration of heterogeneous inputs, the leverage of back-end tools, and automatic design space exploration. It enables building a design flow that is general with respect to the user input (e.g., Simulink and Modelica), and to the output implementation code (e.g., C and EIKON [2]). Using an IF, pieces of the input specification expressed in different languages can be composed. This feature will hopefully foster collaboration among experts in different disciplines by allowing them exchange models and evaluate designs taking into account the interactions with other subsystems. IF also allows targeting of multiple implementation platforms. Building control system vendors usually provide architecture-specific languages for programming their platforms, along with tool chains for simulation, analysis, debugging, and code generation. These tools can be leveraged by translating the IF into vendor specific languages. We define a denotational IF based on the Metropolis Meta Model semantics [9] which can be further translated into an executable model in the METROPOLISII framework and simulated.

In *Step 1* of our design flow, input models are translated into the IF representation. As a proof of concept, we developed a translator based on the

ANother Tool for Language Recognition (ANTLR) framework [1] to automatically translate Modelica models into IF. The translation process may become very involved given the expressiveness of model-based languages. Our approach for dealing with the complexity of this step is to define a *library* of primitives at the intermediate level to capture a large class of building control algorithms. This library is then mirrored by equivalent libraries defined in the source languages to facilitate translation.

In *Step 2*, the back-end automatically *maps* the functional model described in the IF to the architectural model that captures the implementation platform. The part of the functional model to be mapped is a control algorithm. The architecture platform captures computation resources (e.g., terminal control units, embedded processors, and workstations), communication resources (e.g., wired buses and wireless links), sensors (e.g., temperature sensors and CCTV video cameras) and actuators (e.g., valves and switches). During mapping, the functional model in IF is abstracted into the composition of functional tasks and messages. The architecture platform is described in the form of a library of available architectural components that are characterized by their functionality, cost, performance, etc. The mapping problem is cast into an optimization problem of finding the best mapping from the tasks and messages in the functional model to the components in the architectural model, with respect to a set of objective functions and design constraints (including physical constraints).

Step 3 of the design flow conducts software synthesis starting from the mapped design. The synthesis process includes code generation for individual processors in the distributed system, and communication interface synthesis for process communication. During code generation, we translate the functional tasks mapped onto each processor to either generic C code or vendor specific languages. As a demonstration, we developed a translator for translating IF into the EIKON language based on ANTLR. The synthesis of communication interfaces is essential to ensure the correctness of the system when the architecture platform does not directly support the semantics of the functional model. For instance, a synchronous Simulink model is not naturally supported by an asynchronous architecture that is common in building control systems. Our approach includes two main aspects: interface synthesis to guarantee stream equivalence on distributed embedded systems while reducing communication load, and timing constraints to preserve the semantics with consideration of the interaction with physical environment.

Details of the three steps are introduced as follows.

Translation into intermediate format

In Step 1 of the proposed design flow, models capturing the specification of the control algorithms and of the environment are translated into an IF, which is defined based on the Metropolis Meta Model semantics and the nomenclature introduced in [9] to facilitate mapping and code generation. In particular, *processes* (also called *actors*) are the basic entities for specification. They are categorized into continuous processes and discrete processes. Each process is defined by a set of *parameters*, *ports* and *equations*. Parameters are set for configuring the process. Ports constitute the communication interface of the process and can be either input or output port. Equations capture the behavior of the process in the form of an input–output function. Multiple processes may be connected through *channels* to form a *netlist* at the higher level and eventually build the entire system. During execution, the equations in the processes are executed according to an order determined by an *equation manager* (EM) that is local to the process. The set of processes in the system is scheduled by an *equation resolve manager* (ERM).

The translation process may be done manually or through automatic translators. We developed a translator for Modelica based on the ANTLR framework, a parser generator that uses $LL(^*)$ parsing. We chose ANTLR because it provides comprehensive support and consistent syntax for specifying lexers, parsers and tree walkers, and supports generating code in common languages such as C, Java, Ada, and Objective-C. As shown in Figure 2, we define the Modelica grammar and the associated actions for IF generation in ANTLR. Based on such definition, ANTLR will generate a lexer, a parser, and tree walkers for parsing the Modelica language (currently without full support of inheritance and algorithm) and generating IF.

To enable fast translations, we define a domain specific IF library for HVAC control systems in buildings, and we export the library to different specification languages. We reviewed 71 HVAC-related component models in the GPL language from Johnson Controls, 70 in Automated Logic EIKON

language, 42 in Honeywell Spyder, and 59 in the HVAC library defined by the Lawrence Berkeley National Laboratory. Based on this information, we defined a set of basic components used in HVAC control systems and the corresponding processes in the IF, including mathematical and logic functions, signal processing functions, time functions, and psychrometric functions. More details of the library, including an example of the PID (proportional-integral-derivative) component are presented in [11].

The IF first generated by ANTLR is denotational. We may further translate it to an executable IF for simulation-based model validation and exploration. In particular, we choose the METROPOLISII framework [5] for modeling and simulating the executable IF, since its semantics also derives from the Metropolis Meta Model and it provides strong support on modeling heterogeneous systems. The translation from denotational IF to METROPOLISII model is straightforward because of the similarity of their semantics. Processes in IF are translated to *components* in METROPOLISII, with equations translated to *constraints*. The ERM and EMs in IF are translated to *constraint solvers* and *schedulers*, which govern the resolution and scheduling of constraints in a three-phase execution semantics. Below is a code snippet of the PID component described as an executable IF in METROPOLISII.

```
M2_COMPONENT(PID) {
  m2_port⟨i_nb_var_read⟩ setPointPort;
  m2_port⟨i_nb_var_read⟩ procVarPort;
  m2_port⟨i_nb_var_write⟩ outPort;
  double Kp, Ki, Kd, Kc, upper, lower;
  double err, sum, out, outDiff;
  ......
  intg* intg1;  // m2_equation type
  deri* deri1;  // m2_equation type
  void exec() {
    setPoint = setPointPort->read_var();
    procVar = procVarPort->read_var();
    err = setPoint - procVar;
    sum = Kp * err
      + Ki * intg1->calculate(step, err)
      + Kd * deri1->calculate(step, err)
      + Kc * outDiff;
    // update output
    if (sum > upper)
      ......
    outDiff = out - sum;
    outPort->write_var(out);
  }
};
```

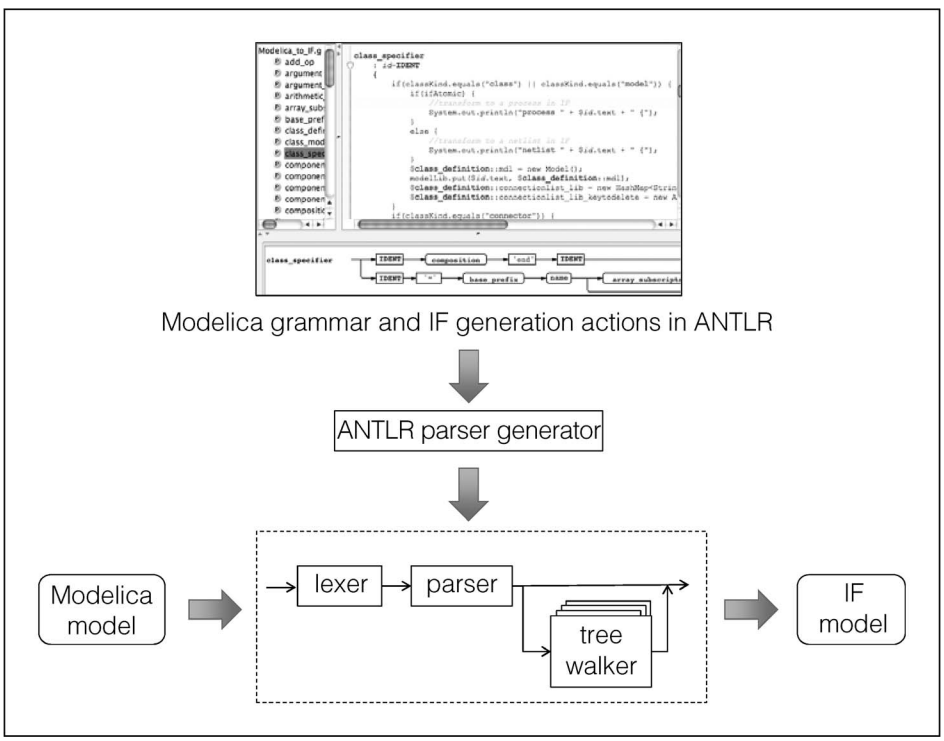

Figure 2. ANTLR-based Modelica to IF translation.

Design space exploration in mapping

In Step 2 of the design flow, mapping is conducted to explore the design space, including the selection of computation resources, the allocation of control functions onto processors, and the synthesis of communication network. Note that traditionally, the mapping step focuses on bridging a given functional model and a given architectural model. In this work, we extend its scope to include the exploration of the computation and communication resources in the architecture platform. We propose a mapping flow as shown in Figure 3. The inputs include a functional model that is derived

from the IF, an architecture platform that captures the resources for realizing the functional specification, and a set of design constraints and objectives.

The functional model represented in IF includes processes and channels. Through automatic extraction based on ANTLR, processes and channels are abstracted to tasks and messages, by hiding their internal implementation while computing cost and performance metrics of interest. Formally, the functional model is represented as a directed graph $F = \{T, M\}$, where T is the set of tasks and M is the set of messages that are communicated between tasks.

The architecture platform is defined as a library of architectural components $A = \{A_k = (P_k, L_k) : P_k \subseteq P, L_k \subseteq L\}$, where a component A_k is the composition of a set of basic computation components P_k through a set of basic communication components L_k. The set P contains all available basic computation components such as sensors, actuators and processors. Similarly, the set L contains all basic communication components such as wired or wireless communication links, routers, and repeaters. Note that P and L may contain virtual components, which are place holders that can be refined to real components in later design stages.

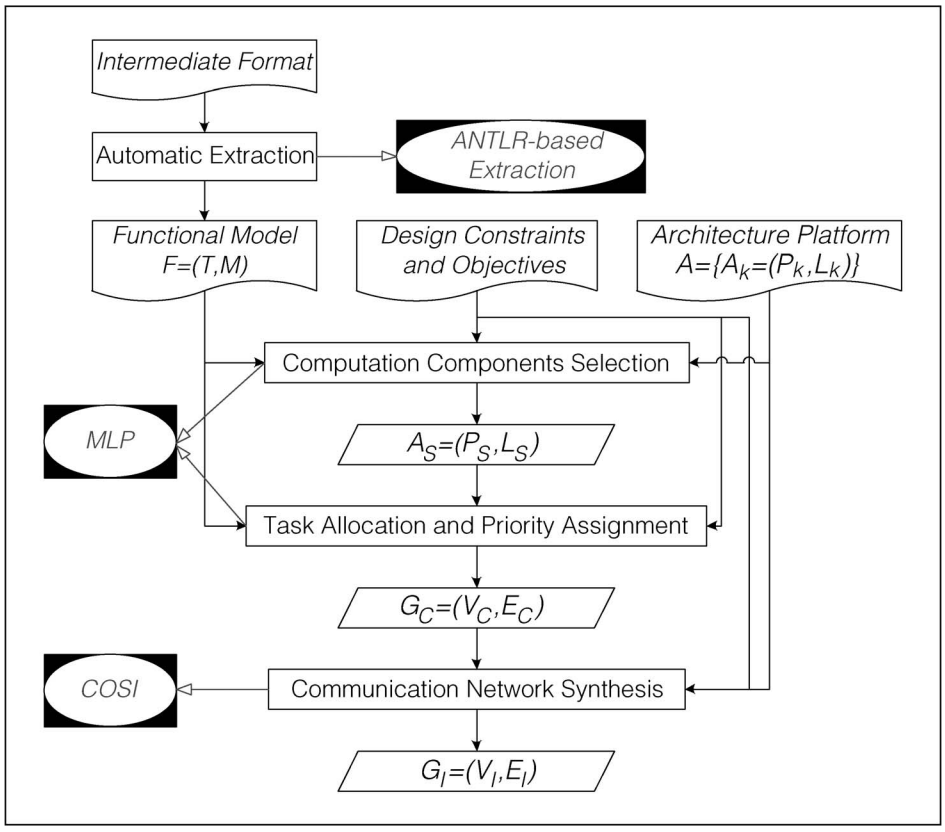

Figure 3. Mapping flow.

Conceptually, there are three steps in our mapping flow. In the first step, a set of computation components P_S is selected from the architecture platform and connected by virtual communication components L_S. This constitutes an architectural model A_S onto which the functional model can be mapped. In the second step, the tasks in the functional model are allocated to the computation components in the architectural model, and if needed, the priorities of the tasks are assigned. The messages are temporarily allocated to the virtual communication components. The output is the mapped model $G_C = (V_C, E_C)$, where V_C denotes the computation components with tasks allocated onto them and E_C denotes the message-allocated virtual communication components. Finally, in the third step, the virtual communication components are synthesized to a communication network, in which the communication between two computation components may flow through multiple links, routers and repeaters, and each link may be shared across multiple end-to-end communications. The output G_I is the eventual implementation of the functional model on the architecture platform.

The mapping flow above is generic—when specific design requirements and platforms are given, each of the three steps in the flow can be formulated accordingly and solved by customized algorithms. In our work, we target a typical building design case: given the functional model F, the architecture platform A, and a set of *design constraints* including building floorplan, candidate locations of sensors, actuators, embedded processors and routers, end-to-end latency deadlines on selected paths, utilization, and memory constraints on embedded processors, we explore the *design space* that consists of the selection of computation components,

allocation of tasks to embedded processors, assignment of task priorities, and communication network, to minimize the *system cost*, which includes the prices of the components and the installation cost. For this specific problem, we combine the first and second step in the mapping flow to explore the design space in an *integrated* mixed integer linear programming (MILP) formulation, and then perform communication network synthesis using the Communication Synthesis Infrastructure (COSI) tool [8]. Details of the MILP formulation and COSI synthesis are presented in [11].

Software synthesis

Step 3 in our design flow is software synthesis, which includes code generation for individual functional tasks and the synthesis of communication interfaces between tasks.

Code generation translates the processes (corresponding to functional tasks) in IF representation to code in target languages. Based on the allocation result from mapping, the target language may be generic C code or vendor specific representation for the mapped embedded processor. Translating into vendor specific languages enables leveraging vendor tools for analysis, debugging and simulation. As a proof of concept, we developed a translator in ANTLR for translating IF to EIKON, a language for modeling BAC systems developed by Automated Logic. Similarly as shown in Figure 2, we define the IF grammar in ANTLR, along with the associated actions for EIKON generation. ANTLR then generates a translator that includes a lexer, a parser and tree walkers. The lexer and parser parse the designs described in IF to an abstract syntax tree (AST), from which the tree walkers generate the target EIKON description. EIKON provides a library of microblocks (control functions) for developing various control sequences. A process will be translated into the microblock that implements the same functionality. In the case that a process does not have corresponding microblock in the EIKON library, the translator implements its functionality in OCL (Operator's Control Language) defined in EIKON. OCL provides a number of mathematical and logical functions in the syntax. For the set of equations in a process, the translator constructs an OCL block by mapping the equations to a set of functions provided in OCL.

Communication interface synthesis preserves the semantics of the input functional model when the architecture platform does not directly support it. A typical case in BAC is that the functional model is synchronous, which eases the design by orthogonalizing functionality and timing, while the architecture platform is distributed and asynchronous. To address this problem, we propose an approach by extending the work from [10] and [6]. In [10], a semantic preservation method is used to implement synchronous functional models on a Loosely Time Triggered Architecture (LTTA), where the computation components are triggered periodically by local clocks that are not synchronized but deviate from each other by bounded drift and jitter. This method guarantees the data value stream on any communication link in LTTA is the same as in the synchronous model, based on the transformation through an intermediate layer called Finite FIFO Platform (FFP). An FFP model consists of a set of sequential processes communicating via bounded FIFO queues. The key to guarantee stream equivalence is to enforce that a process skips a round when any of its input queues is empty or any of its output queues is full.

In the building automation domain, the architectures typically follow the same LTTA paradigm, where periodic sampling from the sensors and discrete-time control are common for applications such as HVAC. Therefore, we leverage the approach from [10] in our communication interface synthesis, and extend it in two aspects—one is to reduce the communication load in original approach by removing redundant data transferring for Triggered Synchronous Block Diagrams (SBDs), the other is to guarantee the stream equivalence in open systems (i.e., taking into account the physical environment) by enforcing additional timing constraints.

The first extension focuses on optimizing communication for Triggered SBDs. At the heart of many synchronous languages such as Simulink, SBDs are usually chosen as the model of computation. The fundamental component in an SBD is a block that can be modeled as a state machine with inputs and outputs à la Mealy. Outputs of blocks are connected to inputs of other blocks to form a diagram. Provided the diagram has no cyclic dependencies, all blocks "fire" in a certain order within a synchronous step, so that the external outputs of the diagram are computed by propagating the external inputs throughout the diagram. Triggered SBDs are an extension of SBDs where the firing of a block may be controlled by a Boolean signal called a *trigger*. At a

given synchronous step, if the trigger is true, the block fires normally; otherwise, the block stutters, for example, keeps its local state and local outputs unchanged until the next step. Triggered SBDs are useful for modeling multirate systems, where different parts of the system operate at different time scales. The triggering patterns need not be periodic. A trigger signal for one block may be produced by another block or an external input of the diagram. A special case of Triggered SBDs is *Timed* SBDs, where triggering patterns are known statically ("at compile time").

The semantic preservation problem studied in [10] applies to "pure" SBDs, where all blocks fire at every synchronous step. Its method can be applied to Triggered SBDs through a *trigger elimination* procedure that transforms triggers into standard inputs, however this often results in unnecessary communication overhead: a block always reads input messages and sends output messages even when its trigger is false. In our work, we eliminate this overhead from two directions: first, a process does not send messages to its successor processes that are not triggered; second, a process which is not triggered need not send a full data message to its successor processes, but only a flag indicating that the data are the same as in the previous step. To achieve these optimizations while still preserving behavior equivalence, "backward" signals are added to transmit the trigger information of a reader to its writers, and each process is restructured into multiple stages to avoid potential deadlock caused by the additional "backward" signals. This method is especially critical in systems where communication is expensive, for example, in wireless applications where the channel capacity is limited, or where energy savings are essential. Details of this approach, communication saving analysis, formal proof of its correctness, and further optimization for Timed SBDs as a special case are presented in [12].

For the second extension, we observed that the assumption that every process (or task after mapped to LTTA) can freely skip a round does not hold if we want to preserve stream equivalence in open systems. Specifically, the sensing tasks in the BAC systems periodically sample inputs from the constantly-changing physical environment. Skipping a round on these tasks means the "old" environment inputs will be overwritten by the "new" inputs, and the data stream is no longer equivalent to the synchronous model. To preserve the synchronous specification, we propose a set of timing constraints on task periods and the drifts of local clocks, as shown in [11].

Case study

We conducted a case study on a room temperature control system example to illustrate our design flow. The functional model captures a two-level control algorithm as shown in Figure 4. The higher level linear-quadratic regulator (LQR) controller determines the set points for lower level PID controllers. The LQR coordinates among multiple rooms (three rooms in the example) to optimize the total energy consumption while maintain a certain comfort level. The PIDs track the set points and interact with the physical environment. The inputs to the plant model include the air mass flow into each thermal zone,

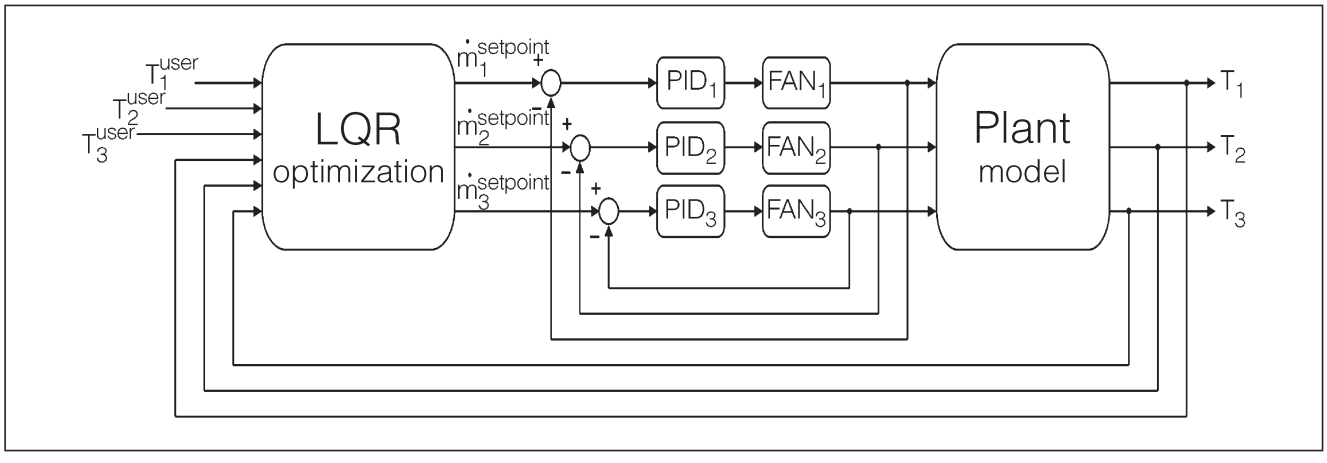

Figure 4. Room temperature control system.

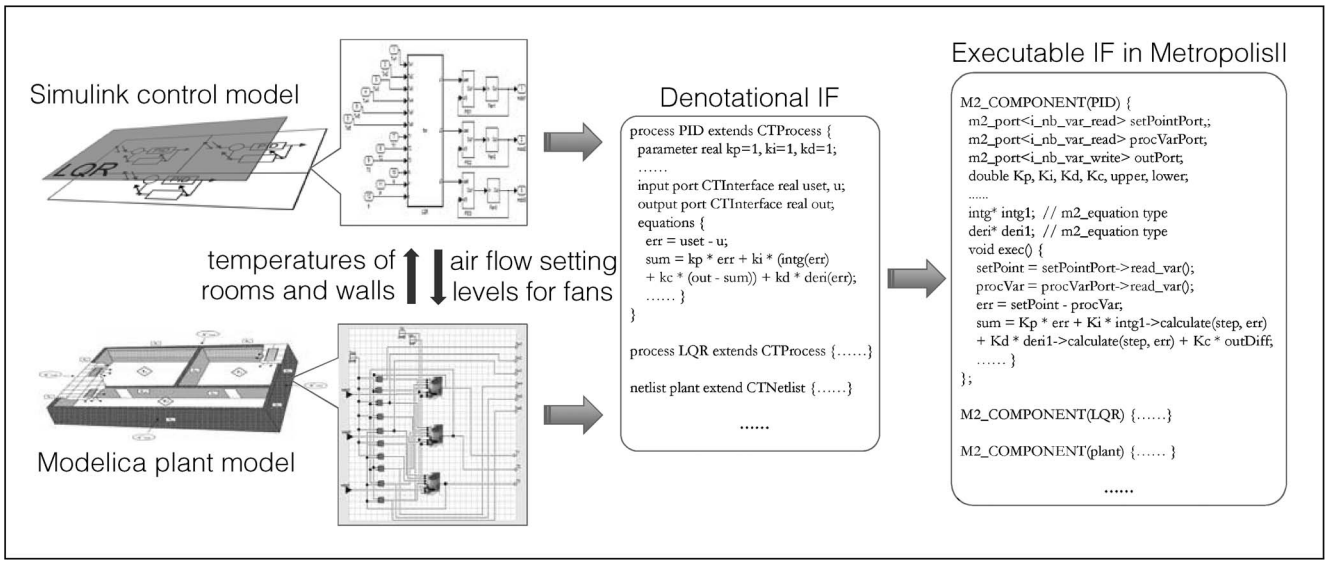

Figure 5. IF translation for room temperature control system.

and the outputs are the temperature of each thermal zone and the temperature of walls. More details on the model and control can be found in [7]. As design input, the controller (including PIDs and LQR) is modeled in Simulink, while the plant is modeled in Modelica.

In the first step of our design flow, the *heterogeneous* input model is translated into a uniformed IF representation, as shown in Figure 5. The Simulink controller model is translated into IF manually, with one-to-one correspondence between the components in Simulink and the processes in the IF representation. The resulted IF includes one LQR process and three PID processes. The Modelica plant model is translated into IF through the automatic ANTLR-based translator. Basic models (classes) in Modelica are translated into corresponding processes in IF, with their equations translated to IF equations.

In order to validate the accuracy of our IF translation, we further translate the denotational IF to an executable model in the METROPOLISII framework as in Figure 5, and compare the simulation in METROPOLISII versus the simulation of original input model. For the simulation of original heterogeneous input model, first the Modelica plant model is imported into Simulink through the Dymola-Simulink interface (Dymola is a modeling and simulation environment for Modelica language). In this case, the entire plant is imported into Simulink as a DymolaBlock, which wraps an S-function MEX block that contains the C-code generated by Dymola for the Modelica model.

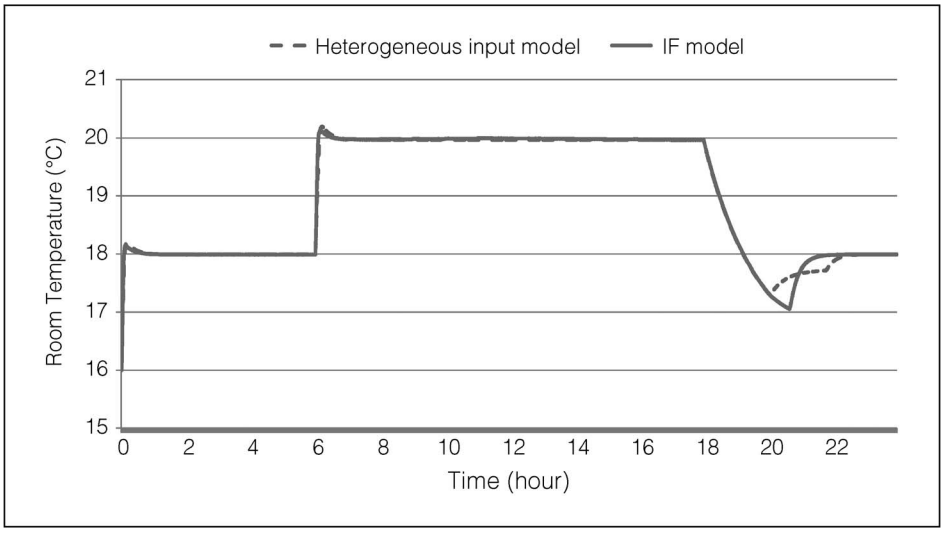

Figure 6. Comparison of heterogeneous input model in Simulink/Modelica and IF model in METROPOLISII.

The comparison of METROPOLISII simulation versus the heterogeneous input model is shown in Figure 6 and Table 1. Figure 6 shows

the temperature of Room 1 from the simulation of two models over an entire day. The other rooms have similar plots. Table 1 summarizes the average and maximum temperature differences for all three rooms. Overall, the simulation results of two models are close, which shows the accuracy of our IF translation. We believe the remaining differences are caused by the difference in implementing ODE solvers.

In the second step of the design flow, mapping is conducted to explore the design space, as explained in design space exploration in mapping. To test the scalability of the algorithm, we extended the number of rooms from 3 to more than 40 (with 61 thermal zones), while keeping the same structure. The building floorplan and physical constraints are obtained from a real office building. The architecture platform is characterized by a library of computational components including sensors, actuators and processors, and a communication library including ARCNET daisy chain buses and routers. The details of the mapping and the cost of the final solution are shown in [11]. The result demonstrates the importance of optimizing both computation and communication of the system.

In the third step of the design flow, we choose LabVIEW from National Instruments (NI) as the target platform, and generate code in NI's *G* language, for which both simulation and C code generation are provided by LabVIEW (we did not choose EIKON for this input model because it does not provide direct support on the matrix operations in LQR). One interesting aspect is that the code generation from Simulink to IF and then from IF to LabVIEW may be conducted at different levels of abstraction. In [11], a PID-level translation and a lower-level translation (with smaller components inside PID as processes) are demonstrated. The comparison between the two shows that the lower-level translation can improve the translation accuracy by 10^1 to 10^3 times, at the expense of more complexity.

In addition, we conducted experiments to demonstrate the ideas of communication interface synthesis. In [11], we focus on the timing constraints for semantic preservation. We model a mapped *distributed* system in LabVIEW with the synthesized communication interfaces, then compare it through simulation to the *synchronous* functional specification in LabVIEW (obtained from IF translation). The detailed experimental results in [11] demonstrate that the two models produce the same results when all the timing constraints we set are satisfied. When we reduce the task periods to intentionally violate some of the timing constraints, we start observing the differences between two models, which confirms the importance of those constraints in our communication interface synthesis.

Table 1 Comparison of all room temperatures between simulink/ modelica model and IF model.

	Room 1	Room 2	Room 3
Avg. differences (°C)	0.033	0.034	0.125
Max. differences (°C)	0.56	0.60	1.13

In another set of experiments, we focus on optimizing communication for Triggered SBDs. To study the average case, we use TGFF to generate random directed acyclic graphs, with number of nodes ranging from 100 to 1000 (each node corresponds to a SBD block). We randomly pick some of the links between nodes as trigger links, and assign a probability of *not* being triggered to every block that has a trigger. The experiment results in Figure 7 show the

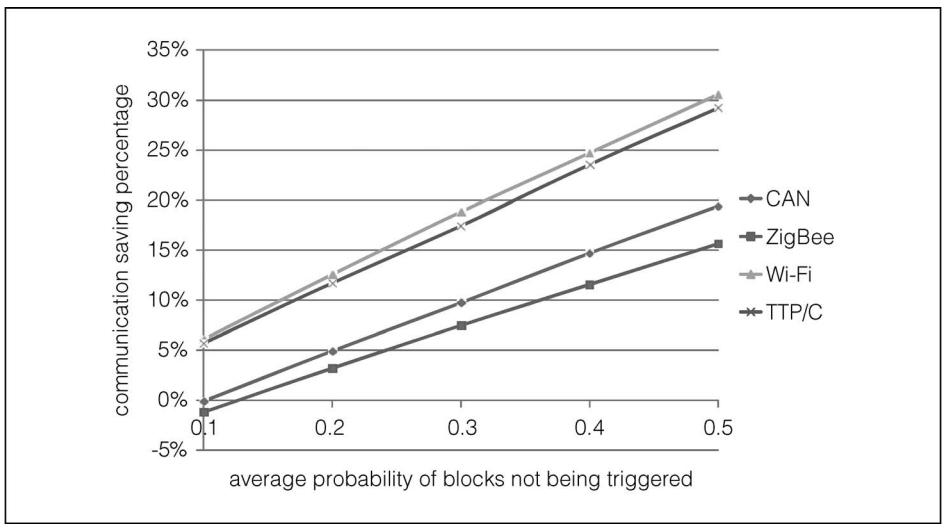

Figure 7. Savings from communication optimization for triggered SBDs.

communication saving percentages (w.r.t. original communication load) for those Triggered SBDs when the average probability of blocks not being triggered ranges from 0.1 to 0.5. We analyzed four communication protocols—CAN bus, ZigBee, Wi-Fi, and TTP/C, which have different message overheads and maximum data payloads. The communication saving achieved by our method increases approximately linearly w.r.t. the probability of blocks not being triggered, and could be very substantial when that probability is high.

In this paper, we proposed a design flow for BAC systems that enables integrating heterogeneous input models, conducts automatic design space exploration, and performs software synthesis on distributed platforms while guaranteeing correctness and reducing communication load. We believe these capabilities can enable the building designers to better adopt model-based design methodologies, and facilitate them to improve design productivity, optimize system performance, and reduce cost.

In the future, we plan to extend the design flow to codesign the building control algorithms and the architectural platform. We believe the characteristics of the architectural platform are important for optimizing the control algorithms. We would like to extend the design space exploration to include more physical aspects such as occupancy, and consider wireless platforms. We would also like to apply our design methodology to codesign multiple subsystems such as HVAC and lighting to achieve better overall system. Because it is likely that different subsystems employ heterogeneous models in terms of semantics and languages, the unified IF representation provided in our design flow will be particularly important to facilitate the integration of those models. ■

■ References

[1] ANTLR. [Online]. Available: http://www.antlr.org/

[2] EIKON-LogicBuilder for WebCTRL. [Online]. Available: http://www. automatedlogic.com

[3] Modelica. [Online]. Available: http://www.modelica.org

[4] Simulink. [Online]. Available: http://www.mathworks.com

[5] F. Balarin, M. D'Angelo, A. Davare et al. "Platform-based design and frameworks: Metropolis and Metro II," in *Model-Based Design for Embedded Systems*, G. Nicolescu and P. J. Mosterman, Eds. Boca Raton, FL: CRC, 2009.

[6] M. Di Natale, A. Benveniste, P. Caspi et al. "Applying LTTA to guarantee flow of data requirements in distributed systems using controller area networks," in *Proc. Design, Autom. Test Eur. Workshop D ependable Softw. Syst.*, 2008.

[7] M. Maasoumy, A. Pinto, and A. Sangiovanni-Vincentelli, "Model-based hierarchical optimal control design for HVAC systems," in *Proc. 2011 ASME Dynam. Syst. Contr. Conf. (DSCC)*, 2011.

[8] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "COSI: A framework for the design of interconnection networks," *IEEE Design Test Comput.*, vol. 25, no. 5, May 2008.

[9] A. Pinto, A. L. Sangiovanni-Vincentelli, L. P. Carloni et al. "Interchange formats for hybrid systems: Review and proposal," in *Proc. 8th Int. Workshop Hybrid Syst. Computation and Contr.*, 2005, pp. 526–541.

[10] S. Tripakis, C. Pinello, A. Benveniste et al. "Implementing synchronous models on loosely time triggered architectures," *IEEE Trans. Comput.*, vol. 57, no. 10, pp. 1300–1314, Oct. 2008.

[11] Y. Yang, A. Pinto, A. Sangiovanni-Vincentelli, and Q. Zhu, "A design flow for building automation and control systems," in *Proc. IEEE 2010 31st Real-Time Syst. Symp. (RTSS)*, Dec. 3–30, 2010, pp. 105–116.

[12] Y. Yang, S. Tripakis, and A. L. Sangiovanni-Vincentelli, "Efficient distribution of triggered synchronous block diagrams, EECS Dept., Univ. California, Berkeley, Tech. Rep. UCB/EECS-2011-115, Oct. 2011.

Yang Yang is a PhD candidate at the EECS Department of the University of California at Berkeley. Her research interests include computer-aided design, real-time distributed embedded systems, and embedded software. Yang has an MS degree in Electrical Engineering and Computer Science from the University of California at Berkeley. She is a Student Member of IEEE.

Qi Zhu is an Assistant Professor at the EE Department of the University of California at Riverside. His research interests include design methodologies for distributed embedded systems, cyberphysical systems, and computer-aided design for circuits. Zhu has a PhD degree in Electrical Engineering and Computer Science from the University of California at Berkeley. He is a Member of IEEE and ACM.

Mehdi Maasoumy is a PhD candidate at the Mechanical Engineering Department at the University of California at Berkeley. His research interests include modeling and optimal control of linear, nonlinear and hybrid systems, and energy efficient building control systems. Maasoumy has a Master's degree from the University of California at Berkeley in mechanical engineering. He is a Student Member of IEEE and ASME.

Alberto Sangiovanni-Vincentelli is Buttner Chair of the EECS Department at the University of California at Berkeley. He is also Synopsys Cofounder, and a Member of the GM Science and Technology Board, the UTC Technology Advisory Council, the Italian Institute of Technology Exec-Committee, and NAE. He is a recipient the Kaufman Award for ''pioneering contributions to EDA'' and the IEEE/RSE Maxwell Medal ''for groundbreaking contributions with exceptional impact on development of electronics.''

■ Direct questions and comments about this article to Yang Yang, EECS Department, University of California, Berkeley, CA 94720; yangyang@eecs.berkeley.edu.