System-level Co-simulation of Integrated Avionics Using Polychrony *

Huafeng Yu, Yue Ma
INRIA Rennes / IRISA
263, Av du Général Leclerc
35042, Rennes, France
huafeng.yu@inria.fr
yue.ma@inria.fr

Yann Glouche
Jean-Pierre Talpin
INRIA Rennes / IRISA
263, Av du Général Leclerc
35042, Rennes, France
yann.glouche@inria.fr
jean-pierre.talpin@inria.fr

Loïc Besnard
IRISA / CNRS
263, Av du Général Leclerc
35042, Rennes, France
loic.besnard@irisa.fr

Thierry Gautier
Paul Le Guernic
INRIA Rennes / IRISA
263, Av du Général Leclerc
35042, Rennes, France
thierry.gautier@inria.fr
paul.leguernic@inria.fr

Andres Toom
IRIT, Université de Toulouse,
Toulouse, France
Institute of Cybernetics at TUT,
Estonia
andres@krates.ee

Odile Laurent
Airbus
316 route de Bayonne
BP M0151/0
31060, Toulouse, France
odile.laurent@airbus.com

ABSTRACT

The design of embedded systems from multiple views and heterogeneous models is ubiquitous in avionics as, in particular, different high-level modeling standards are adopted for specifying the structure, hardware and software components of a system. The system-level simulation of such composite models is necessary but difficult task, allowing to validate global design choices as early as possible in the system design flow. This paper presents an approach to the issue of composing, integrating and simulating heterogeneous models in a system co-design flow. First, the functional behavior of an application is modeled with synchronous data-flow and statechart diagrams using Simulink/Gene-Auto. The system architecture is modeled in the AADL standard. These high-level, synchronous and asynchronous, models are then translated into a common model, based on a polychronous model of computation, allowing for a Globally Asynchronous Locally Synchronous (GALS) interpretation of the composed models. This translation is implemented as an automatic model transformation within Polychrony, a toolkit for embedded systems design. Simulation, including profiling and value change dump demonstration, has been carried out based on the common model within Polychrony. An avionic case study, consisting of a simplified doors and slides control system, is presented to illustrate our approach.

Keywords

heterogeneous modeling, AADL, Polychrony, Simulink, simulation, avionics, GALS.

1. INTRODUCTION

The hardware/software co-design of embedded systems is a complex engineering activity which needs to meet sometimes contradictory objectives of performance and cost. Many improvements to traditional design methodologies have been adopted to satisfy such requirements, including parallel development of hardware and software, high-level modeling and validation, component-based design, and so on.

Parallel development teams allow for an effective development using domain-specific technologies and focused expertise. Parallelism, however, leads to the heterogeneity of models and of components. Meanwhile, the increasing complexity of embedded systems encourages to take advantages of high-level modeling framework such as, for instance, UML MARTE, SystemC, Simulink, AADL. These languages help to reduce complexity and development time by raising levels of abstraction.

These improvements lead to an integration issue, as one then needs to simulate, evaluate and validate the architecture of a system whose components are described with high-level domain-specific modeling diagrams, using heterogeneous models of computation and communication. This issue has been widely studied in the Ptolemy project [12], MoBIES project [9], SML-Sys modeling frameworks [17], ForSyDe modeling frameworks [19], etc. In these projects, heterogeneous models or components, based on different models of computation, are integrated into a system either through agent-based methods or through translating them into common formalisms.

Inspired by these projects, we propose a system-level co-design approach, in the framework of European CESAR project [22], dedicated to the design of avionic applications. Our approach supports high-level heterogeneous modeling, semantic-preserving transformation and integration of models, simulation and performance evaluation of system-level architectures. In this approach, Polychrony [16, 14] is adopted as a common development platform to bridge heterogeneity between modeling and simulation. A complex system is first partitioned into functional behavior and hardware architecture components. The architectural part maybe common

*The current work has been partially supported by the European project CESAR (Cost-efficient methods and processes for safety relevant embedded systems).

across several different projects or subsystems of a larger system. In our approach, the system behavior is modeled in the synchronous model of computation of Simulink/Gene-Auto [6] [23], whereas the architecture is modeled in the asynchronous model of computation of AADL [21]. These high-level models are transformed into Signal programs [10] via SME models [14] (Signal Meta under Eclipse). Signal supports a multi-clocked synchronous model of computation allowing for the description of locally synchronous and globally asynchronous systems such as those considered here. Finally, C or Java code is generated from Signal programs. Simulation can then be carried out for the purpose of performance evaluation and VCD (Value Change Dump) based demonstration [7]. The advantages of our approach is that high-level models and transformations are developed in parallel to reduce development time, hence cost; Polychrony and its associated tools allow a fast and effective simulation, evaluation and validation of system-level architectures, with no need for additional translations into other formalisms.

The case study of a simplified model of the A350 doors management system is proposed by Airbus in the frame of CESAR project. This case study is used in the present paper to illustrate the effectiveness of our approach to system-level simulation.

2. SDSCS AND POLYCHRONY

SDSCS (Simplified Doors and Slides Control System) is a generic simplified version of the system that allows managing doors on Airbus series aircrafts. It is a safety-critical system as incorrect door closing or opening during flight may lead to fatal crashes. The reliable system design and validation is therefore very important. In addition to the fulfillment of safety objectives, high-level modeling and component-based development are also expected for fast and efficient design. SDSCS has been chosen for the demonstration of capabilities developed in the CESAR project.

An Airbus aircraft has several kinds of doors, such as passenger, cargo and emergency doors. In this paper, we focus on the management of passenger doors. Each passenger door has a software handler, which achieves four tasks: monitor door status via door sensors; control flight lock actuators; manage the residual pressure of the cabin by controlling the outflow valves, visual status indication and an aural warning; and inhibit cabin pressurization if any external door is not closed, latched and locked. The four tasks are implemented with simple logic that determines the status of monitors, actuators, etc. according to the sensor readings.

In addition to sensors and actuators, SDSCS is equipped with other hardware components, such as processing units, communication link, and concentrators. The SDSCS is implemented on the IMA (Integrated Modular Avionics) platform, in which CPIOMs (Core Processing Input/Output Modules) and RDCs (Remote Data Concentrators) are connected via the AFDX (Aircraft Full DupleX) network (Figure 1). Sensors and actuators are also connected to RDCs via AFDX. CPIOMs receive sensor readings via RDCs and communicate also with other systems via AFDX.

In addition to high-level system tools for the modeling of SDSCS, an integrated environment is also required for the following purposes: validation, particularly fast code generation for simulation; partial specification and multi-clock mechanism enabling; possibility of both behavioral and architectural specification; easy connections to validation

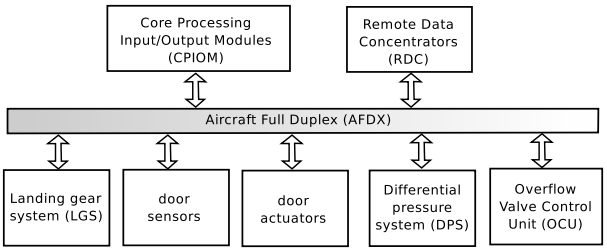

Figure 1: A simple illustration of the SDSCS system architecture.

tools. Polychrony answers this demand. It is an integrated development environment for the design of reactive embedded systems. The Polychrony toolset takes Signal as the kernel design language, and SME as a metamodel. It provides a formal framework for the system modeling at a high level of abstraction, design validation at different levels, as well as simulation for deterministic specifications. The Signal language is based on *synchronized data-flow* [10]. The Signal formal model provides the capability to describe systems with several clocks (multiclock/polychronous) as relational specifications. Relations are useful as partial specifications and as specifications of nondeterministic devices (e.g., a nondeterministic bus) or external processes (e.g., an unsafe car driver). These specifications allow code generation that enables simulation, analysis, validation and synthesis. The application domain of Polychrony includes safety-critical systems, such as avionics and automotive systems. In this paper, we adopt Polychrony as a common framework for system co-design.

3. SYSTEM DESIGN

Our proposed design process for the high-level modeling and simulation of SDSCS is illustrated in Figure 2. Three stages are presented in the design process, which include: co-modeling, model transformation, and simulation and analysis. According to the SDSCS specification, functionality and architecture are modeled in Simulink/Gene-Auto and AADL respectively at a high-level of abstraction in the **co-modeling** stage. Automatic and manual **model transformations** are carried out in the second stage so that executable code, such as C or Java, is generated. In the transformation, the SME model and the Signal model are considered as intermediate models. Semantic coherence is ensured between different models in the transformation. **Simulation and analysis** are then made possible, and performance evaluation can be obtained with regard to a specific architecture.

3.1 Behavior modeling in Simulink

Dataflow models and state machines are common models of computation adopted in the system design of avionics, automotive applications, etc. One of the most popular tools that accept these models is Simulink/Stateflow [6] in the Matlab family. One of the models of computation adopted by Simulink is extended dataflow, and Stateflow relies on the model of state machines. A typical Simulink model is defined by a set of interconnected blocks, which model entities in a system, such as sensors, actuators, and logical operations. The library of Simulink includes function blocks that

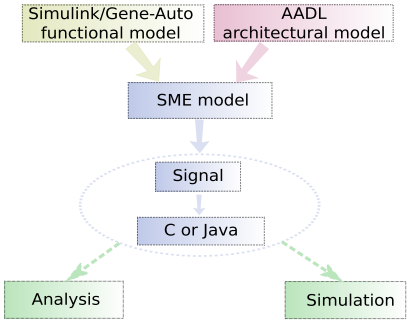

Figure 2: The design process for heterogeneous modeling and simulation within Polychrony

can be linked and edited in order to model the dynamics of the system. Gene-Auto is a framework for code generation from a safe subset of Simulink and Stateflow models for safety critical embedded systems [23]. This safe subset is also adopted in our work. From now on, Simulink is used for short to indicate the subset of Simulink and/or Stateflow languages that is adopted by Gene-Auto.

Only discrete time of Simulink is taken into account in this paper. Each block of Simulink is associated with a specific activation clock. At each tick of this clock, a block carries out a computation on its available inputs and produces new outputs. A global activation clock is used to synchronize the activation clocks of Simulink blocks. From this point of view, our Simulink model is synchronous, and each of its blocks is thus modeled as a synchronous Signal process.

The behavioral aspects of SDSCS have been modeled in Simulink and Stateflow (shown in Figure 3). Sensors, such as *flight_status*, *dps*, and *door_io_in*, are connected to four Simulink blocks, each of which implements a SDSCS task as mentioned in Section 2. Three blocks, *slide_warn_ctrl*, *pres_warn_ctrl*, and *closed_locked _and_latched* are associated with simple logic to determine actuator status from sensor readings. The fourth block, *flight_lock_ctrl* is associated with a state machine (specified in Stateflow), which decides the status of flight lock actuators.

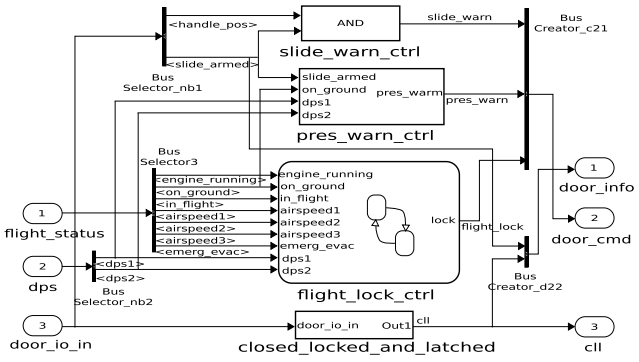

Figure 3: The door handler component of the SDSCS modeled in Simulink

The model transformation chain from functional models in Simulink to Signal is divided into several steps. The first step involves in the transformation of Simulink models to Ecore based Gene-Auto models [23]. These models are then translated into Signal via the SME metamodel through a transformation implemented in Kermeta [3]. The whole chain from high-level models to executable code is illustrated in Figure 4.

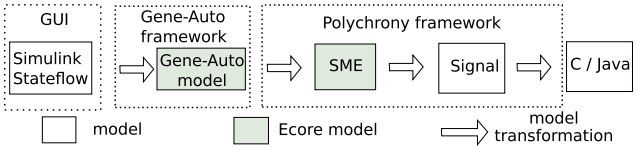

Figure 4: A global view of the functional model transformation chain.

3.2 Architecture modeling in AADL

AADL (Architecture Analysis and Design Language) [21] is an SAE (Society of Automotive Engineers) standard aimed at the high-level architecture design and evaluation for embedded systems. AADL is used to describe the structure of such systems as an assembly of software components allocated on an execution platform. AADL adopts component-based paradigm for the system description. AADL has three categories of components: *application software*, *execution platform* and *composite*. The first one includes *process*, *thread*, *thread group*, *subprogram*, and *data* components. The second one models the hardware part of a system, which includes *processor*, *memory*, *device*, and *bus* components. The last one represents a composite of software, execution platform, or other system components.

ARINC 653 (Avionics Application Standard Software Interface) [8] is a standard that specifies an API (Application Programming Interface) for avionic software, following the IMA architecture. It defines an APEX (APplication EXecutive) for space and time partitioning. An ARINC *partition* is a logical allocation unit resulting from a functional decomposition of the system. *Partitions* are composed of *processes* that represent executive units.

Simulation of AADL applications can be carried out in the framework of Polychrony once they are translated into Signal. This translation is mainly based on the temporal properties of AADL components. These properties are translated into Signal clocks. However, there are still several issues, presented in the next, to be resolved.

First, an AADL component, whose temporal properties are unknown or unset, is associated with independent clocks. It leads to non-determinism with regard to temporal behavior in the system. This specification is considered as a partial specification that can be translated into Signal. However, for the purpose of simulation, complementary signals are added in the system to bridge between partial specifications and deterministic implementations.

Second, an AADL thread may perform a computation for a specified time interval, which is defined as a temporal property. Moreover, the outputs of this thread are available and transferred to other components at *completion* time by default, or at *deadline* time in the case of *delayed* port communication. As the computation of a Signal process is instantaneous, different clocks that model scheduling activity, including *start*, *completion*, *deadline*, *dispatch*, etc., have been integrated into each component.

Figure 5 shows an overview of the SDSCS modeled in AADL. The whole system is presented as an AADL *system*. The two doors, *door1* and *door2*, are modeled as *subsys-*

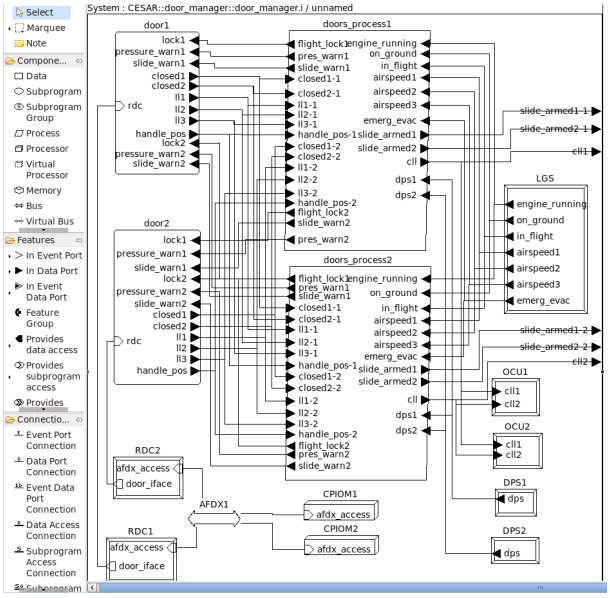

Figure 5: An overview of the SDSCS system architecture modeling in AADL

tems. They are controlled by two processes *doors_process1* and *doors_process2* respectively. These processes are bound to two *processors*: *CPIOM1* and *CPIOM2*, to perform the computation independently. *Sensors* and *Actuators*, such as *LGS*, *DPS*, *OCU*, etc., are modeled as AADL *devices* that interface with external environment of the system. All the communication between the *devices* and *processors* is through the *bus*: *AFDX1*. SDSCS has three *threads* to manage doors: *door_handler1*, *door_handler2*, and *doors_mix*. These threads are implemented by Simulink models. In addition, each *processor* runs one *doors_process*. These two components are placed into one ARINC *partition*. Each *processor* is associated with an ARINC *partition_level_OS*, which is responsible for scheduling all the *processes* in the same *partition*. In this example, all the threads and devices are periodic, and share the same periodicity.

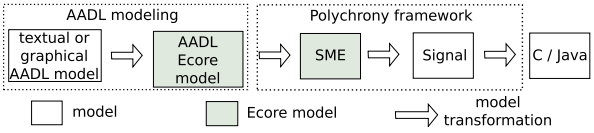

Figure 6: A global view of the architectural model transformation chain

A complete transformation chain from AADL models to executable code is illustrated in Figure 6. The AADL Ecore model is translated into the SME model using the ATL transformation language [1]. The SME model is then transformed to Signal programs. In the next subsection, several other additional models needed for the complete simulation are presented.

3.3 Additional models

In addition to the high-level Simulink and AADL models, additional models are also needed in the SDSCS for the complete simulation. They include an allocation model, a scheduler model and an environment model.

In this case study, the allocation of functionality onto architecture is specified in the AADL model. In the AADL to Signal transformation, all the threads mapped on to the same processor (CPIOM) are placed in the same partition. The generated Signal programs are annotated with allocation information. All the Signal processes translated from the same partition have the same Signal pragma `RunOn i` [10], which enables the distribution of these processes onto the same processor `i`.

According to the AADL specification, a partition-level scheduler is needed for the simulation. This scheduler takes events, such as *dispatch*, *start*, *completion*, etc., into account for the scheduling of threads in the same partition. Simple non-preemption partition-level schedulers have been coded in Signal manually for the simulation. More sophisticated schedulers, such as that provided by Cheddar [20] are expected to be integrated into the system.

Sensors and actuators are the media between SDSCS and its environment. The environment with regard to SDSCS includes the aircraft system outside SDSCS as well as the environment outside the aircraft which provides flight altitude, air speed, etc. The environment modeling is carried out directly in Polychrony. Several Signal processes are added as environment models. For instance, a process detects the *close* status of physical doors and sends the *close* signals to door sensors, another process provides air speed readings to aircraft speed signals sensors. These Signal processes that model the environment are composed with other Signal processes that are transformed from the high-level Simulink and AADL models.

3.4 Composing models

Once all the needed models are obtained, the composition of these models is possible. All the parts, such as system behavior, hardware architecture, environment, and schedulers, are expressed by Signal processes, thus a composition of these processes implies system integration. *Functional models* in Figure 7 imply the composition of all processes translated from Simulink. The communication of distributed processes is implemented by MPI (Message Passing Interface). *Architecture models* indicate the composition or connection of hardware interfaces and hardware implementations. *Simulation clocks* are used to provide reference clocks and periodical clocks for simulation. The integrated system is then used for C or Java code generation via the Signal compiler for simulation purpose.

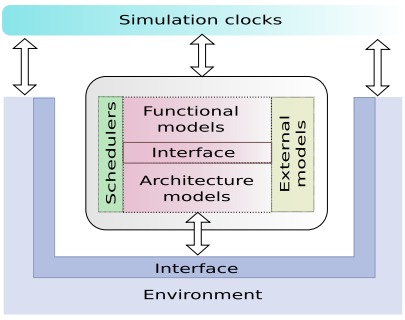

Figure 7: The system integration from the point of view of Signal processes

4. SIMULATION AND TIMING ANALYSIS

In addition to heterogeneous system specification, another advantage of our approach, compared to other similar projects, is to benefit from simulation and validation tools associated with Polychrony in the same framework. The Polychrony toolkit adopts various analysis and validation techniques: static analysis, simulation, model-checking, etc. In this paper, we present simulation related techniques, particularly, profiling and VCD demonstration.

4.1 Profiling

Software profiling is considered as a kind of dynamic program analysis through the information gathered when the program executes. This analysis is always involved in performance improvement. Profiling is also adopted in Polychrony for the performance evaluation of Signal programs [15] [13]. The profiling process includes: temporal properties specification, temporal homomorphism, and co-simulation.

In the framework of Polychrony, profiling refers to timing analysis through associating *date* and *duration* information to Signal programs. Each signal x in the program is associated with a date signal, $date\_x$, to indicate its availability time. This date signal may be specified with metric clock, logical clock or clock cycles. In the first case, date signals are positive real numbers, and in other cases they are positive integers. Each operation in Signal programs is associated with the *duration* information, which has the same data type as date signals. The *duration* is represented by a pair of numbers corresponding to the worst and best case.

Morphism of Signal processes represents a series of transformations of Signal processes without changing their synchronous semantics. Temporal properties are introduced in the morphism of Signal processes so that they are used to reveal the timing aspect of these processes. A Signal process can be considered as a directed graph of signals and operations, where signals are arcs and operations are nodes. A temporal morphism of Signal process preserves the graph structure. However, nodes are replaced by operation durations and arcs are replaced by date signals.

In addition, *duration* parameters of Signal operations allow the parameterization of homomorphism. These parameters together with their values allow specifying the execution of operations on specific architecture (particularly processing elements). They also enable to import and use specific library of cost functions in the homomorphism.

As the temporal homomorphism preserves the synchronous semantics, the homomorphic Signal processes can be composed together for the co-simulation. The latter exhibits the timing behavior with regard to previously mentioned temporal properties. Figure 8 illustrates a schema of the co-simulation that has been carried out successfully. *SDSCS* is the original Signal program, whose inputs are provided by *Inputs*. *T(SDSCS)* is the temporal homomorphism of *SDSCS* with regard to specified *Temporal properties* and a parameterization of *Library of cost functions*. *Date* provides date signals to *T(SDSCS)* according to *I*. The input signals are synchronized to their corresponding date signals. Control values of *SDSCS*, which decide specific traces of execution, are sent to *T(SDSCS)* so that they have the same execution traces. Date signals of inputs and outputs of *T(SDSCS)* are finally sent to *Observer* in order to obtain the simulation result *V*.

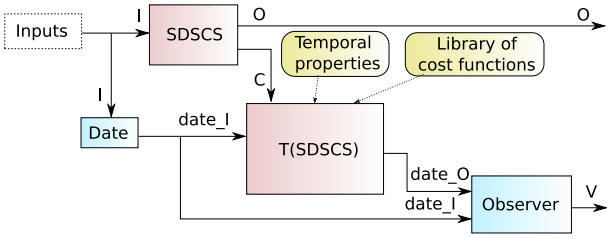

Figure 8: The co-simulation of Signal programs with regard to its temporal behavior

4.2 VCD demonstration

In addition to profiling, another simulation has also been carried out. It aims at the visualization of value change during the execution of programs via VCD. VCD files are generally generated by EDA (Electronic Design Automation) logic simulation tools, and they adopt an ASCII-based file format, whose simplicity and compact structure allows a wide spread in application simulation. Moreover, the four-value VCD format has been defined as IEEE Standard [7] together with VHDL (Verilog Hardware Description Language). In our simulation, traces are recorded in VCD format. The VCD files are then used for the visualization of simulation results through graphical VCD viewers, such as GTKWave [2]. Figure 9 shows a visualization result of the simulation. In this figure, the change of signal values with regard to the fastest clock is shown.

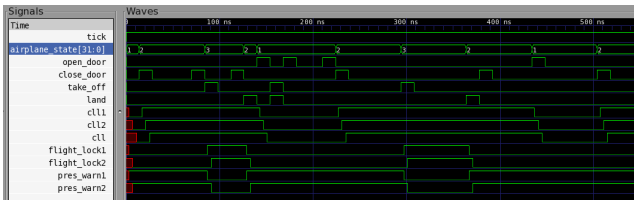

Figure 9: The simulation is illustrated by a VCD viewer: GTKWave

5. RELATED WORK

Due to the heterogeneous nature of embedded systems, modeling and integration of heterogeneous subsystems has always been an important issue. It has been widely studied in the Ptolemy project [12], MoBIES project [9], SML-Sys modeling frameworks [17], ForSyDe modeling frameworks [19], etc. In these projects, certain common formalisms or agent-based methods are proposed to model or express heterogeneous components, which generally rely on different models of computation. We follow the similar approach: SME/Polychrony is adopted as a common formalism, which is used to express heterogeneous models, i.e., Simulink and AADL models. An advantage of our approach is that formal verification, simulation and analysis can be directly carried out on this common formalism, without supplementary translations into other formalisms.

Validation of AADL models via formal methods has been studied in [18] and [11], where AADL specifications are associated with formal semantics so that formal verification is performed. [18] proposes VTS (Visual Timed Scenarios) as a graphical language for the specification of AADL behavioral

properties. Then a translation from VTS to TPN (Time Petri Nets) is presented. Model-checking of properties expressed in VTS is enabled using TPN-based tools. [11] studies a general methodology and an associated tool for translating AADL and behavior annex specification into the BIP language. This translation allows simulation of AADL models, as well as application of formal verification techniques developed for BIP. Two kinds of verification have been applied on generated BIP model: deadlock detection by using the tool Aldebaran, and verification of thread deadlines and components synchronization by using BIP observers. In our work, we lay emphasis on the overall system integration of heterogeneous models and fast performance-related simulations.

6. CONCLUSIONS

In this paper, we presented an approach to address high-level and heterogeneous model based system co-design in the domain of avionics. Functional behavior, expressed with the synchronized dataflow model, is modeled in Simulink/Gene-Auto, whereas distributed hardware architecture is modeled in AADL. SME/Polychrony, based on the polychronous model of computation, is adopted as a common formalism to bridge between two heterogeneous models. Model transformations from Simulink to SME, and from AADL to SME have been developed to support our work. Simulation and timing analysis is then enabled in the framework of Polychrony. An avionic case study, called simplified doors and slides control system is briefly presented in this paper to demonstrate our approach. Our approach enables an early phase simulation via profiling in consideration of timing constraints and also a simulation demonstration by VCD viewers. The perspective of our work includes connections to other architecture exploration and timing analysis tools, such as Syndex [5] and RT-Builder [4]. In addition, integration of automatic test case generation and real-time scheduling tools are also expected.

7. REFERENCES

[1] ATL. `http://www.eclipse.org/atl/`.

[2] GTKWave. `http://gtkwave.sourceforge.net/`.

[3] Kermeta. `http://www.kermeta.org/`.

[4] RT-Builder. `http://www.geensoft.com/`.

[5] Syndex. `http://www-rocq.inria.fr/syndex/`.

[6] The MathWorks: Simulink. `http://www.mathworks.com/products/simulink/`.

[7] IEEE Standard for Verilog Hardware Description Language (VHDL), 2006. IEEE Std 1364 -2005.

[8] Airlines Electronic Engineering Committee. Avionics Application Software Standard Interface (ARINC 653), 1997.

[9] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, and G. P. snd O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proc. IEEE*, 91(1):11–28, 2003.

[10] L. Besnard, T. Gautier, P. Le Guernic, and J.-P. Talpin. Compilation of polychronous data flow equations. In S. Shukla and J.-P. Talpin, editors, *Correct-by-Construction Embedded Software Synthesis: Formal Frameworks, Methodologies, and Tools*, 2010.

[11] M. Y. Chkouri, A. Robert, M. Bozga, and J. Sifakis. Translating AADL into BIP - Application to the Verification of Real-Time Systems. In *International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES'08)*, Toulouse, France, September 2008. Springer-Verlag.

[12] J. Eker, J. Janneck, E. Lee, J. Liu, J. Ludwig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity: The Ptolemy approach. *Proc. IEEE*, 91(1):127–144, 2003.

[13] A. Gamatié, T. Gautier, and L. Besnard. Modeling of Avionics Applications and Performance Evaluation Techniques using the Synchronous Language SIGNAL. In *SLAP'03*. Elsevier Science B.V., 2003.

[14] INRIA ESPRESSO team. Polychrony. `http://www.irisa.fr/espresso/Polychrony`.

[15] A. Kountouris and P. Le Guernic. Profiling of Signal Programs and its application in the timing evaluation of design implementations. In *Proceedings of the IEE Colloq. on HW-SW Cosynthesis for Reconfigurable Systems*, HP Labs, Bristol, UK, 1996.

[16] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann. Polychrony for System Design. *Journal for Circuits, Systems and Computers*, 12(3):261–304, 2003.

[17] D. A. Mathaikutty, H. D. Patel, S. K. Shukla, and A. Jantsch. SML-Sys: a functional framework with multiple models of computation for modeling heterogeneous system. *Design Automation for Embedded Systems*, 12:1–30, 2008.

[18] D. Monteverde, A. Olivero, S. Yovine, and V. Braberman. VTS-based Specification and Verification of Behavioral Properties of AADL Models. In *International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES'08)*. Springer-Verlag, 2008.

[19] I. Sander and A. Jantsch. System modeling and transformational design refinement in ForSyDe. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, 23(1):17–32, 2004.

[20] F. Singhoff and A. Plantec. AADL modeling and analysis of hierarchical schedulers. In *ACM international conference on Ada (SIGAda'07)*, 2007.

[21] Society of Automotive Engineers (SAE). Architecture Analysis & Design Language (AADL, SAE standard ASS5506). `http://www.sae.org`.

[22] The CESAR project. Cost-efficient methods and processes for safety relevant embedded systems. `http://www.cesarproject.eu`, 2010.

[23] A. Toom, T. Naks, M. Pantel, M. Gandriau, and I. Wati. Gene-Auto: An Automatic Code Generator for a Safe Subset of SimuLink/StateFlow and Scicos. In *European Conference on Embedded Real-Time Software (ERTS'08)*, 2008.