Towards the Integration of BIP Models to Extend ESMoL with Asynchronous Semantics

Mohamad Jaber and Joseph Sifakis
VERIMAG, Centre Équation
2 av de Vignate
38610, Gières, France
{jaber,sifakis}@imag.fr

Graham Hemingway, Joseph Porter, Gabor Karsai, and Janos Sztipanovits
Vanderbilt University ISIS
Nashville, TN 37203 USA
{heminggs,jporter,gabor,sztipaj}@isis.vanderbilt.edu

ABSTRACT

This is the text of the abstract.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.2.2 [**Software Engineering**]: Design Tools and Techniques; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design

Keywords

Model-Integrated Computing

1. INTRODUCTION

High confidence embedded systems development features multi-faceted design activities complemented by model and data analysis to ensure the correct operation of the final product. Such analysis is multi-faceted due to the heterogeneity of domains involved in the problems (e.g. cyber-physical systems), leading to high costs and extended schedules for development efforts. Analyses suffer from difficulties of scale due to state and parameter space explosion problems. Further, interactions between design domains (such as the physical continous-time and computational discrete-event) are poorly understood, requiring analyses to be overly conservative. Recently the research community for embedded systems has explored constructive techniques for design and analysis, where system correctness properties may be inferred from component correctness properties and more abstract interaction models defined on the interconnection structure of the design. Constructive methods strive to dramatically reduce state explosion problems by assuring correctness during model construction, while avoiding overly conservative approximations.

BIP [2] is a framework for the constructive assurance of computational properties in embedded systems design. BIP is a component framework for constructing systems by superimposing three layers of modelling: Behaviour, Interaction, and Priority. The first layer consists of a set of atomic components represented by transition systems. The second layer models Interaction between components. Interactions are sets of ports specified by connectors [4, 3]. Priority, given by a strict partial order on interactions, is used to enforce scheduling policies applied to interactions of the second layer. The BIP component framework has a formal operational semantics given in terms of Labelled Transition Systems and Structural Operational Semantics derivation rules.

The BIP language offers primitives and constructs for modelling and composing complex behaviour from atomic components. Atomic components are communicating automata extended with C functions and data. Transitions are labelled with sets of communication ports. Compound components are obtained from subcomponents by specifying connectors and priorities.

ESMoL is another modeling language which aims to experiment with encoding constructive techniques for high-confidence systems into a single modeling environment. Prior work has dealt with modeling and deployment for distributed control systems implemented using an implementation of the time-triggered model of computation[10, 8, 7]. While the time-triggered architecture [6] provides many guarantees for deterministic execution and fault tolerance, extensions are necessary to handle sporadic traffic. Some hardware solutions aim to address these problems, such as time-triggered ethernet[9] and AFDX[1] (to name a few of the most popular).

The execution environment addresses only part of the problem. High-confidence designs require assurances of correct execution, including guarantees for dynamic stability, schedulability and deadlock freedom. The ESMoL development effort seeks to address such problems constructively. In order to provide assurances of deadlock freedom, we aim to integrate asynchronous designs modeled in BIP into our modeling, analysis, and deployment environment.

The integration of BIP and ESMoL seeks to address a number of outstanding problems, and confers benefits on both tools:

1. ESMoL offers platform and deployment modeling, along with wrapper code generation for distributed messaging and real-time execution. Using ESMoL, BIP models can easily be tested against multiple deployment configurations.

EMSOFT '10 October 24-29, Scottsdale, Arizona

2. As mentioned, BIP offers constructive deadlock analysis, which is essential to support high-confidence design activities. ESMoL resimulation could include BIP models for communication protocols and task schedulers, leading to a more accurate determination of possible deadlocks.

3. ESMoL currently lacks asynchronous semantics in both the modeling environment and runtime, so extensions to support BIP models will address this deficiency.

In this report we describe the first phase of the integration effort: importing BIP models into ESMoL, and creating platform-specific simulations. To achieve this, a number of technical challenges must be overcome:

1. Creating an appropriate syntactic transformation from BIP models to ESMoL models.

2. Mapping BIP component interaction models to appropriate message transfer models for deployment, while preserving semantics.

3. Extending the ESMoL code generation framework to support imported BIP models.

4. Extending the FRODO runtime (used in the ESMoL tools) to support asynchronous operation.

We will cover design details for each of these challenges by describing the flow of an example model through the tools. We will also discuss current development progress for each part, and include results where appropriate. Section 2 introduces the example and describes our solution in more detail. Section 3 covers the syntactic transformation between BIP and ESMoL. Section 4 describes the semantic transformation from generic BIP connectors to send-receive messaging required to distribute BIP execution on a network. Execution environment extensions and simulation results are discussed in Section 6. Finally we visit related work (Sec. 7), future work, and conclusions (Sec. 8).

2. SOLUTION OVERVIEW AND EXAMPLE

Talk about the overall solution and motivate with an example.

2.1 Workflow

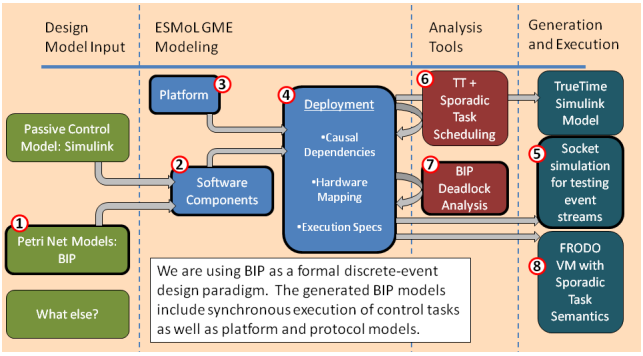

Figure 1: Flow of models in the design process.

ESMoL supports a model-based design flow for embedded systems. The process is shown in Fig. 1.

1. Import design models from other languages (here we are interested in BIP).

2. Blocks in the imported language are used to define components in the design language.

3. User specifies the platform topology and its parameters.

4. User specifies the component-to-hardware deployment mapping.

5. We generate socket implementations in order to test event-driven applications.

6. The scheduling tool uses specified timing and execution frequency bounds to determine schedulability and calculate start times for time-triggered tasks.

7. The BIP model is re-generated with the platform models included (for analysis).

8. We generate an implementation in C to run on the virtual machine.

BIP interactions are much more expressive than those of ESMoL. On the other hand, ESMoL includes explicit model concepts to describe the hardware platform and deployment of the design on a platform.

Consider two important cases:

- In the distributed case, BIP-defined components are deployed on separate processors. Then event-driven interactions between components are strictly causal, as they must communicate by sending messages. We restrict the BIP syntax to send/receive type interactions, including broadcast but excluding rendezvous. Some parallel/distributed platforms provide more advanced synchronization primitives, but our platforms are more simple.

- In the local case, BIP-defined components are deployed on the same processor. Locally, we can easily interpret send/receive and broadcast using synchronous data flow semantics. Rendezvous is also possible, but we do not handle that case yet.

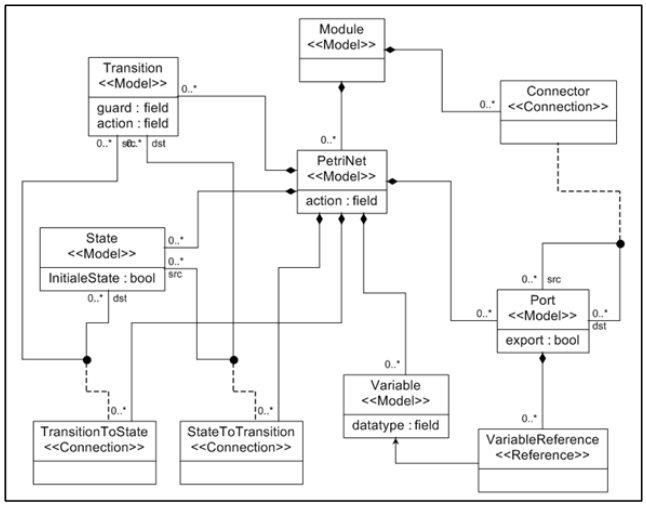

Figure 2: GME Metamodel for restricted BIP Petri net language.

The BIP GME metamodel (Fig. 2) formalizes the structure of a language where each component is specified as a Petri Net. Petri

Net components contain states, transitions, and connections between them. A transition is interpreted as handling a sending or receiving interaction, depending on the connections made to the associated port. For inbound connections (the destination end of a connector) the transition is triggered by an arriving message. For outbound connections, the transition sends a message.

2.2 Send and Receive translation for interactions

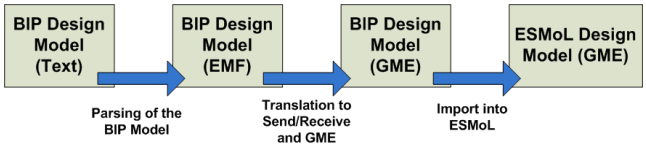

Figure 3: Process for importing BIP models into ESMoL.

Todo: Details of the send/receive translation (MJ).

2.3 Importing into ESMoL

The ESMoL language is a composite of many sublanguages. One of those sublanguages is structurally indentical to the BIP metamodel previous described. The transformation is an isomorphism from a BIP GME model into an ESMoL BIP GME model, but which navigates the structure of the ESMoL model to place the imported data into the right folders.

3. BIP-GME TO ESMOL CONVERSION

How we got the BIP model into ESMoL.

Talk about adding environment and scheduling information.

What assumptions need to be made? What parts of the model need to be updated by hand? What about platform timing information?

Creating an appropriate syntactic transformation from BIP models to ESMoL models.

Mapping BIP component interaction models to appropriate message transfer models for deployment, while preserving semantics.

3.1 Component based construction

A **port** p is defined by the port identifier p, and the data variables associated with the port.

An **atomic component** is a Petri net extended with data. It consists of a set of ports P used for the synchronization with other components, a set of transitions T and a set of local variables X. Transitions describe the behaviour of the component. They are represented as a labelled relation on the set of control locations L.

Figure 4 shows an example of an atomic component with two ports r_1, t_1, a variable x, and two control locations l_1, l_2. At control location l_1, the transition labelled t_1 is possible. When an interaction through t_1 takes place, a random value is assigned for the variable x. This value is exported through the port r_1. From the control location l_2, the transition labelled r_1 can occur (the guard of the transition is true by default), the variable x is eventually modified and the value of x is printed.

An **interaction** a is defined by 1) P a support set of ports 2) G a guard, which is an arbitrary predicate on the variables of its ports, 3) U an upward update function 4) D a downward update function. Synchronization through an interaction involves three steps: 1) The computation of the upward update function U ; 2) Evaluation of the guard G 3) The computation of the downward update function D. we note that we can define hierarchical interactions [4, 3], however, we can flatten hierarchy [5], and hence, we just consider a set

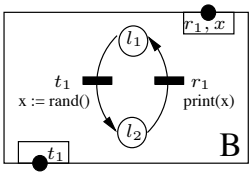

Figure 4: An example of an atomic component in BIP

of non-hierarchical interactions.

One can add **priorities** to reduce non-determinism whenever many interactions are enabled. Then the interaction with the higher priority would be chosen. These priorities can also be guarded.

A **Composite component** C is defined from existing atomic, and from a set of interactions which are connected to the atomic components. We note that a composite component obtained by composition of a set of atomic components can be composed with other components in a hierarchical and incremental fashion using the same operational semantics. However, it is also possible to flatten a composite component and obtain a non-hierarchical one [5].

4. SEMANTIC TRANSFORMATION

4.1 Engine Protocol

The operational semantics is implemented by the BIP engine. In the basic implementation, the engine computes the enabled interactions by enumerating the complete list of possible interactions in the model. At each step, the engine selects the enabled interactions from the complete list of interactions, based on the current state of the atomic components. Then, among the enabled interactions, priority rules are applied to eliminate the ones with lower priority.

The main loop of the engine consists of the following steps:

1. Each atomic component notifies the engine of its current state.
2. The engine enumerates the allowed interactions in the model and selects the enabled ones based on the current state of the atomic components.
3. Selected interactions are filtered according to the priority model to keep only those with highest priority.
4. The engine then selects one of the remaining interactions, and notifies the involved atomic components of their respective transitions to take.

4.2 Send/Receive

Why and how ? To do .....

5. CODE SYNTHESIS

Code generation for the Petri Net components as well as the FRODO wrappers (two subsections).

6. EXECUTION ENVIRONMENT EXTENSIONS AND SIMULATION RESULTS

Results using our example model with lots and lots of plots.

7. RELATED WORK

What else is out there? Metropolis – what else?

8. FUTURE WORK AND CONCLUSIONS

8.1 Future Work

What still needs to be done? What are the unexpected problems?

8.2 Conclusions

Did we adequately show that we are on the right track? Did we convince the reader that asynchronous integration is possible and useful?

9. ACKNOWLEDGEMENTS

This work is sponsored in part by the National Science Foundation (grant/contract number NSF-CCF-0820088) and by the Air Force Office of Scientific Research, USAF (grant/contract number FA9550-06-0312). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

10. REFERENCES

[1] ARINC 664 Specification. Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network, 2004.

[2] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP, Sept. 2006. Invited talk.

[3] S. Bliudze and J. Sifakis. A notion of glue expressiveness for component-based systems. In *CONCUR 2008 - Concurrency Theory*, volume 5201 of *LNCS*, pages 508–522, 2008.

[4] S. Bliudze and J. Sifakis. The Algebra of Connectors—Structuring Interaction in BIP. *IEEE Transactions on Computers*, 57(10):1315–1330, 2008.

[5] M. Bozga, M. Jaber, and J. Sifakis. Source-to-source architecture transformation for performance optimization in BIP. In *SIES*, pages 152–160. IEEE, 2009.

[6] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct 2001.

[7] J. Porter, G. Karsai, and J. Sztipanovits. Towards a time-triggered schedule calculation tool to support model-based embedded software design. In *Proc. of ACM Intl. Conf. on Embedded Software (EMSOFT '09)*, Grenoble, France, Oct 2009.

[8] J. Porter, G. Karsai, P. Volgyesi, H. Nine, P. Humke, G. Hemingway, R. Thibodeaux, and J. Sztipanovits. Towards model-based integration of tools and techniques for embedded control system design, verification, and implementation. In *Workshops and Symp. at MoDELS 2008 (ACES-MB), LNCS 5421*, Toulouse, France, 2009. Springer.

[9] W. Steiner. TTEthernet: Time-triggered services for Ethernet networks. In *Digital Avionics Systems Conf., 2009. DASC '09. IEEE/AIAA 28th*, pages 1.B.4–1 –1.B.4–1, Oct 2009.

[10] R. Thibodeaux. The specification and implementation of a model of computation. Master's thesis, Vanderbilt University, May 2008.