BIP ESMoL Integration Design Document

Mohamad Jaber, Graham Hemingway, Joseph Porter

October 23, 2009

1 Goals

Long-Term Goal: Extend ESMoL to import BIP design models and generate BIP verification models with platform effects included.

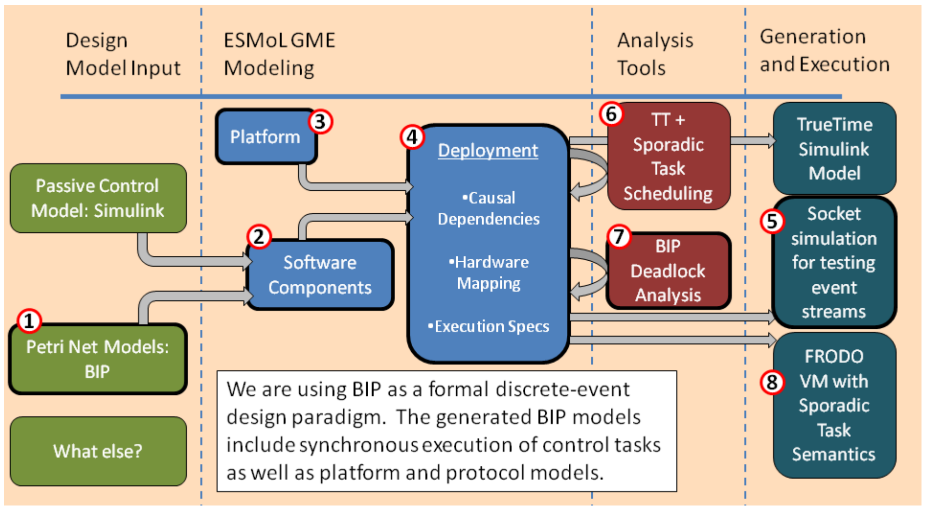
Short-Term Goals:

1. Get an example working end-to-end with the socket simulation.
2. Do another example including Simulink functions invoked by BIP.
3. Create a full BIP model from an example, including platform effects.
4. Extend the virtual machine and scheduler for sporadic, event-driven tasks.
5. Do an end-to-end example including Simulink, BIP generation, and synthesis to the virtual machine.

1.1 ESMoL Overview

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.



We are using BIP as a formal discrete-event design paradigm. The generated BIP models include synchronous execution of control tasks as well as platform and protocol models.

Figure 1: Flow of models in the design process.

7. The BIP model is re-generated with the platform models included.
8. We generate an implementation in C to run on the virtual machine.

1.2 Things to Do

1.2.1 Goal 1: Socket Simulation

1. Create a useful BIP example involving multiple event-triggered inputs.
2. Create templates for socket simulation output.
3. Integrate socket output into the stage2 generator.

Questions / issues:

1. How do we specify the timing characteristics of the sporadic input events in ESMoL?
2. How do we calculate schedulability for the specified events?

1.2.2 Goal 2: Simulink invoked by BIP

1. Extend the example to use Simulink-specified components (or make a new example).
2. Create a BIP template for a Simulink dataflow model, including preemption.
3. Extend the scheduler model to include constraints for sporadic tasks.

1.2.3 Goal 3: Generating BIP (including Platform)

1. Create BIP templates for execution of the virtual machine and communications.

2. Extend Stage2 with BIP generation.

1.3 Things Done

1. 10/14/2009 Created this spec document!!!

2 BIP GME

2.1 Metamodel Details

Describe the design of the BIP GME language, and how it's restricted from the full expressiveness of BIP (and why).

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.

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

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

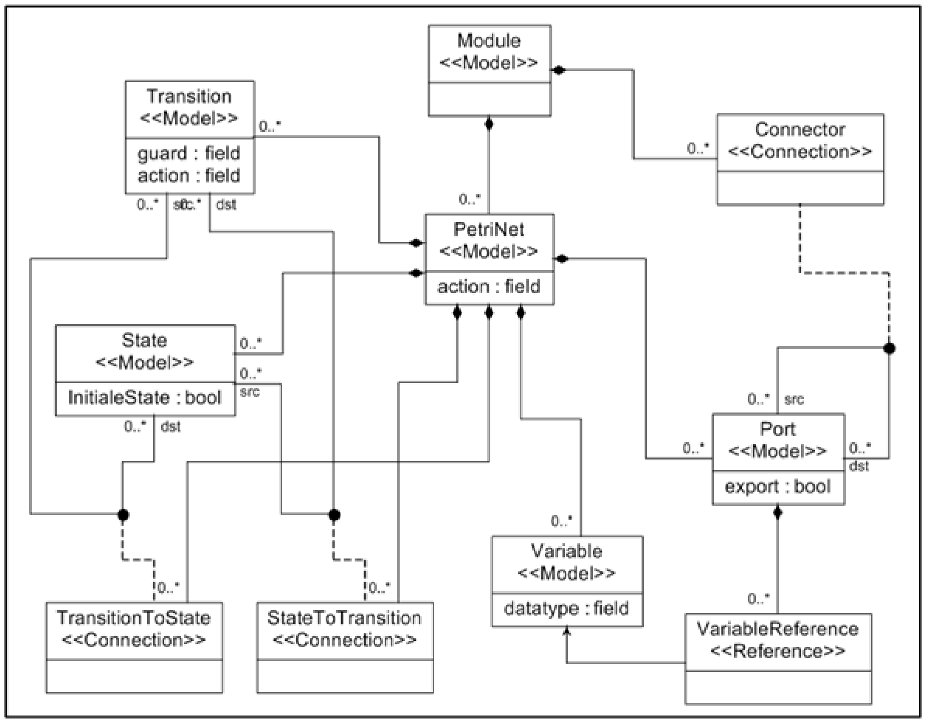


Figure 2: GME Metamodel for restricted BIP Petri net language.

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.

2.4 Future work

- Figure out how to map rendezvous to locally-deployed components.
- Figure out how to model and support distributed synchronization.

3 Petri Net Components

3.1 Component Code Generator Semantic Details

We interpret Petri net component execution as follows:

1. Initially all states marked as InitialeState start with a single token.

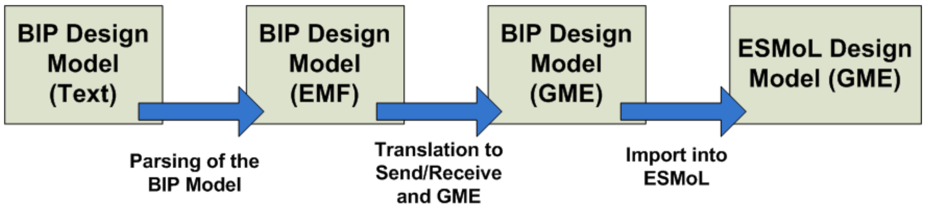


Figure 3: Process for importing BIP models into ESMoL.

2. The message updates the local variables referred to from within the port. In order to fire a transition the guard must be true, and all source states for the transition must have a token.
3. When the transition fires all source states have a token removed, and all target states will receive a new token.
4. The attached action will be invoked.
5. For sending transitions a message is sent containing the variable values. This translates to either a synchronous invocation of another local component, or the transmission of a message to a remote component.

The component code implements only the internals of the behavior. The code implementing the interactions comes later, after we create a deployment model. Each component will be run in a loop that continues with event invocations until no more transitions are available. Then the initial states will be reset.

Todo: Illustrate the semantics with an example (JP).

3.2 Code Generation using Google CTemplate

Todo: Show template code examples and describe the instantiation process.

4 ESMoL

4.1 Language Details

Put together an example that shows the parts needed to do BIP models.

5 Two-stage generation from ESMoL

Include details of relevant parts and changes from ESMoL abstract.

6 Platform-Specific Generators

6.1 Stage 2 Details

Details of how the stage 2 generator works.

6.2 Sockets-Based Execution

Our plan for simulating event-triggered models using sockets.

6.3 Virtual Machine Details

Design details for extending the virtual machine to do event-triggered tasks.

7 Scheduler Extensions

Scheduling sporadic tasks - what are the constraints, and how do we model them? How does that change the language and its infrastructure?

8 Generating BIP Models

8.1 Dataflow Component Behavior Synthesis

Describe translation from Simulink models.

8.2 Interaction Synthesis

How do we map the design interactions onto the virtual machine models? Include model of preemption, etc... Describe the semantics here.

8.3 Virtual Machine Templates

Templates for instantiating the communications controller and the task scheduler on a single node.

Need details for:

- Task to scheduler interactions
- Scheduler to comms controller interactions
- Environment interactions
- Comms controller to comms medium interactions