

Integrating HybridSAL into the META Language and Design Flow

Tihamér Levendovszky and Xenofon D. Koutsoukos

January 2011

Table of Contents

[Introduction 3](#_Toc283305755)

[Integrating HybridSAL into the META Language and Design Flow 5](#_Toc283305756)

[META Concepts 5](#_Toc283305757)

[Overview of HybridSAL 6](#_Toc283305758)

[Semantic Mapping between Simulink/StateFlow and HybridSAL 7](#_Toc283305759)

[The Simulink/Stateflow to HybridSAL Transformation 8](#_Toc283305760)

[The Transformation of the Case Study 11](#_Toc283305761)

[Compositional Verification of META languages with BIP 12](#_Toc283305762)

[BIP Summary 12](#_Toc283305763)

[Translation to BIP and back 13](#_Toc283305764)

[Conclusions and Future Directions 14](#_Toc283305765)

# Introduction

This document describes the conceptual integration of the META workflow and the HybridSAL hybrid model checker and the BIP (Behavior, Interaction, Priority) tools. Currently, the META is an ongoing project, and we are going to use the recent available information about the META workflow and language as of writing. An important concern is, however, that we incorporate certain degree of flexibility to adapt our techniques to the changes in the META tools and concepts.

We are going to describe our ideas with a case study of a Cabin Air Compressor (CAC) provided by Honeywell. The overview of the Simulink model is described in Fig. 1a.

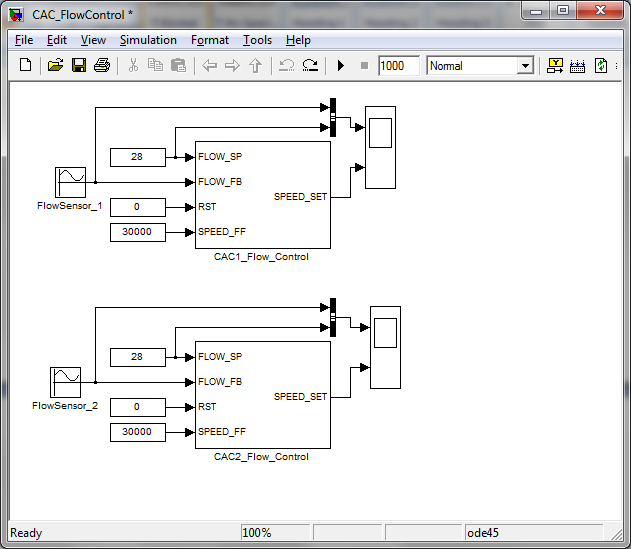


Figure 1. a. CAC Control Overview

In this system, there are two parallel path of two compressors; both are working together to control the air and the pressure in the cabin. Although the two systems work on the same rate, that is, they have to provide the same rate of flow, the speed of the compressors can be different. The figure depicts an open loop simulation, practically, *FlowSensor\_1*/*FlowSensor\_2* originate from the plants.

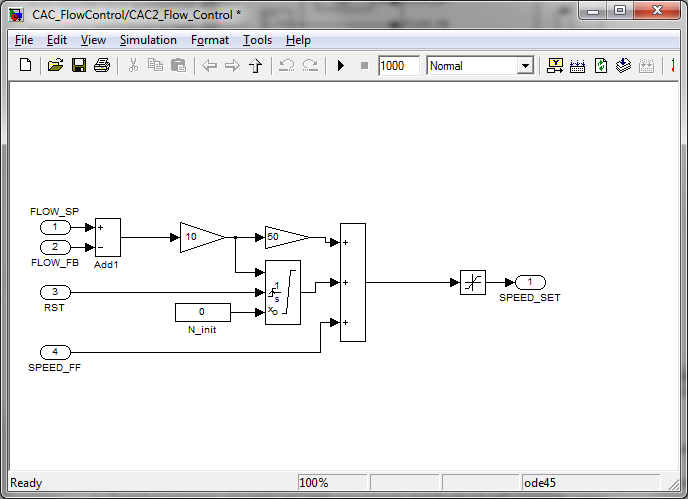


Figure 1b. Flow Control Internals

Fig 1b depicts the internals of the flow control boxes. Essentially, the solution is a PI control realization. The signal *FLOW\_SP* is the set point coming from an upper layer, while *RST* is a reset signal. Moreover, *FLOW\_FB* is the feedback, i.e. the actual sensed data. *SPEED\_FF* is used when the sensor fails. It is a signal describing the altitude; from the altitude an approximation can be computed and used instead of the erroneous feedback.

This document is organized as follows. Firstly, we briefly summarize the information necessary to justify the method described in the next sections. Namely, we show the most important META languages in their current state; then we briefly introduce the HybridSAL system. Then we show a conceptual description of our method that creates a semantic mapping between the META languages and HybridSAL. Then we give an overview of the integration with the BIP tools. Finally, we analyze the possible directions of future work.

# Integrating HybridSAL into the META Language and Design Flow

## META Concepts

The META language, called **CyPhy**, is composed of several modeling languages. The most important for us are the Embedded System modeling Language (ESMoL), and the Bond Graph language.

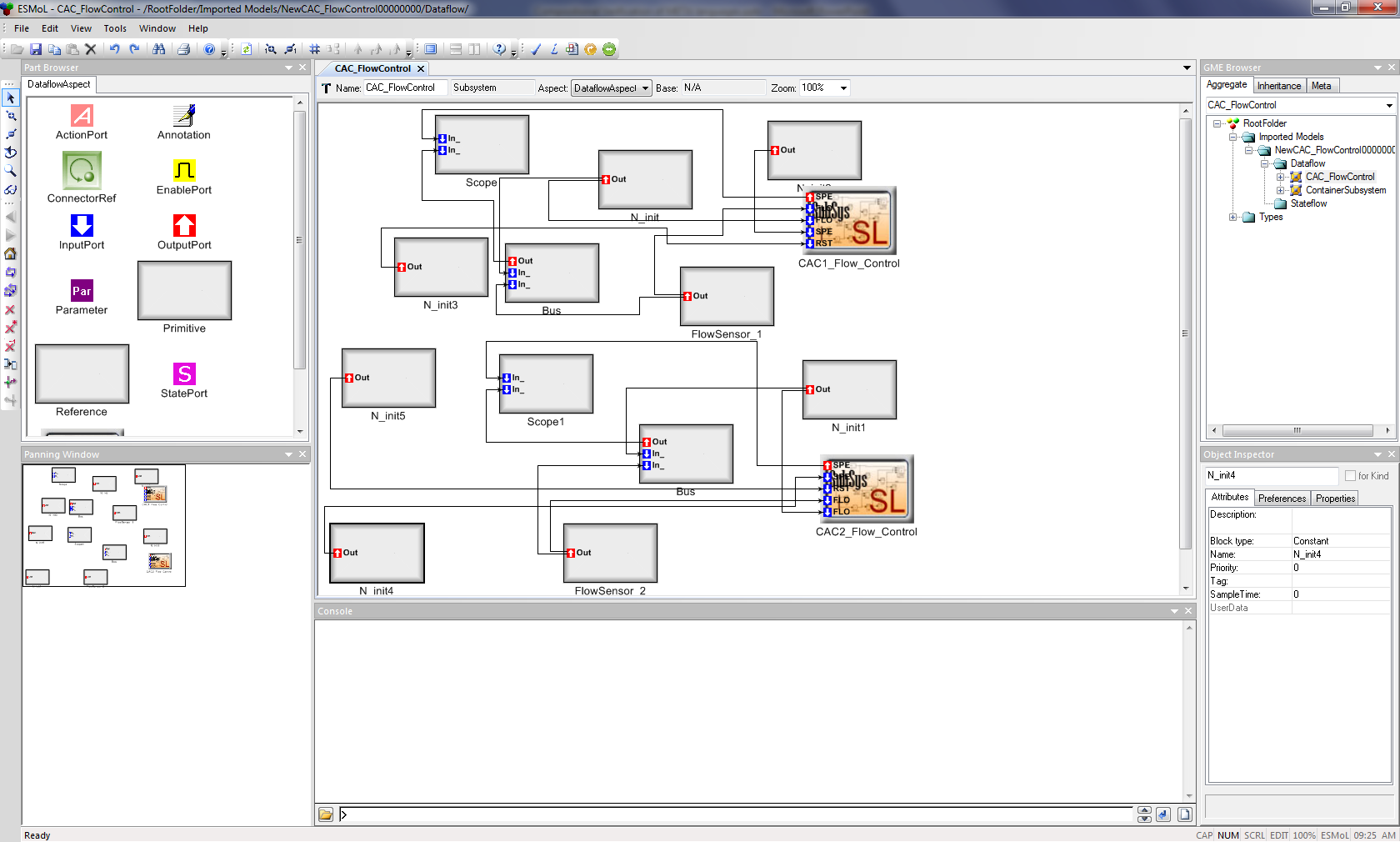


Figure 2. The case study in ESMoL

**ESMoL** (Fig. 2) is a modeling language that is able to describe discrete models, and facilitates the step-by-step refinement of the design, along with physical implementation and scheduling in real time environments. The **BondGraph** language is capable of describing plants and processes from different physical domain, such as mechanical, thermodynamical, and electrical domains. Furthermore, META has an architecture and a requirements language as well. Among others, the requirements language captures the validation results. It is integrated with DESERT, the META design space exploration tool. It is significant for us in the sense that the variables in the requirements model store the result of the validation. The relevant part of the META workflow is shown in Fig 3.

The META toolset includes a translator tool from MATLAB Simulink/Stateflow to ESMol. As of writing, this translation supports both discrete and continuous time systems, however, the tool chain makes use of the discrete part only.

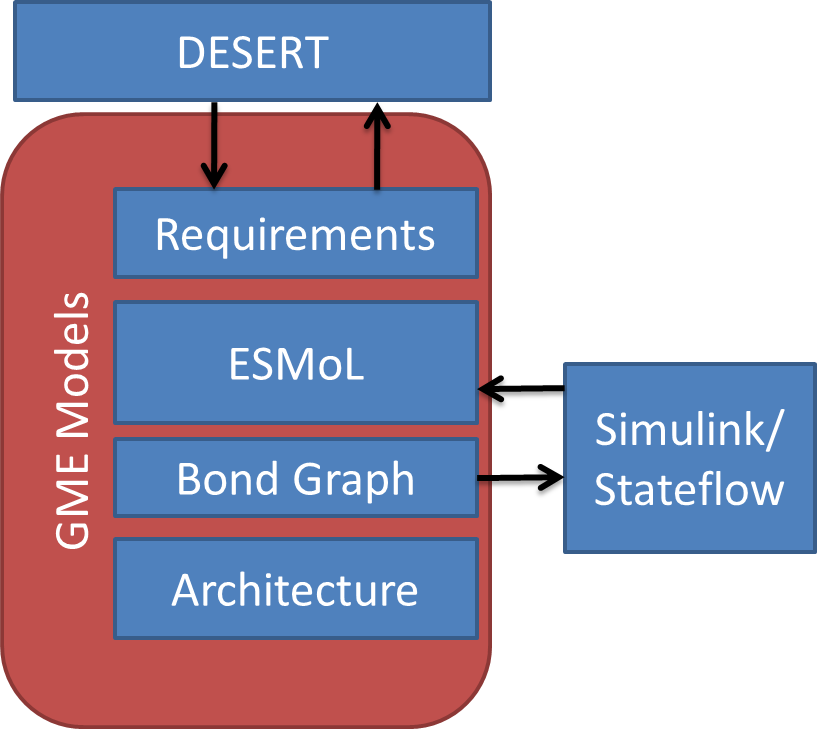


Figure 3. META Workflow

Also, a translator from Bond Graphs to MATLAB is provided as a part of the META workflow. It is conceptually important here that the abstraction level is significantly different: bond graphs are low level formalisms compared to a Simulink/Stateflow model. In case of this transformation, this means information loss, which implies that the inverse transformation is really hard to create, and involves ambiguity.

## Overview of HybridSAL

HybridSAL is used to verify safety properties of hybrid automata models. A sample hybrid automaton is depicted in Fig. 5. Roughly, a hybrid automaton is a state machine, where the states contain ODE-s (state space description of a linear continuous time system), and the transitions are enabled by Boolean expression defined over the state variables.



Figure 4. A Sample Hybrid Automata[[1]](#footnote-1)

What makes the hybrid automata representation used by HybridSAL different from the general notion is that the transitions are not taken immediately when the guard conditions enable them. HybridSAL uses state invariants (last line in the states) to force the transition before it is too late, i.e. they invalidate the state. HybridSAL takes a textual representation as its input, however, on the conceptual level, it is satisfactory for us to use the graphical representation in this document.

## Semantic Mapping between Simulink/StateFlow and HybridSAL

Based on the META workflow and the capabilities of HybridSAL, both discussed above, we develop a conceptual framework to integrate HybridSAL into the META workflow. Recalling Fig. 3, we can easily come to the conclusion that the connection points of the integration are the ESMoL model along with the requirements model. As far as the latter is concerned, it is quite natural to bring the validation result to the format of the requirements model. Thus, DESERT can be executed over the results, enabling one of the most crucial paths of the META workflow automatically.

With respect to Simulink/Stateflow, we have two equivalent representations: one is Simulink, while the other one is the ESMoL part of CyPhy. Since the feedback from the verification tool is going to be stored in the requirements model, it is reasonable to use another CyPhy component as an input. This is an implementation detail: it is not important whether the tool connects to Simulink, or, after a conversion, it obtains the same information from an ESMoL model. We refer to the model as Simulink model throughout this document.

Moreover, the transformation from Bond Graphs to Simulink/Stateflow might be problematic, since it allows generic functions that might not have an equivalent expression in the domain of ODEs. This problem is orthogonal to the model representation; it is conceptually present in Simulink and in ESMoL.

Fig. 5 shows the overall architecture of the integration.

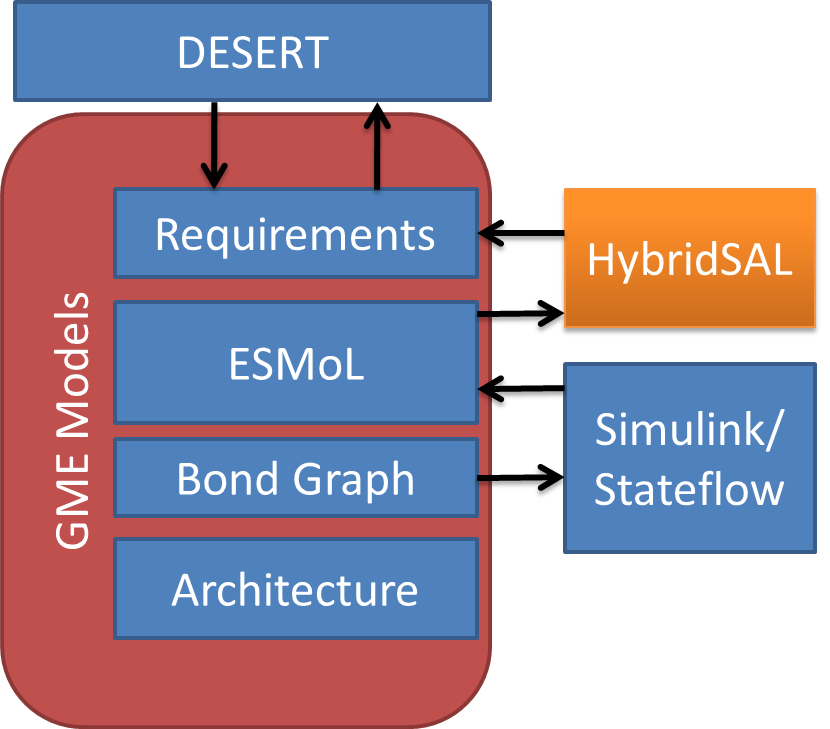


Figure 5. META-HybridSAL integration: overall architecture

In the next sections, we give a detailed explanation of the Simulink/Stateflow to HybridSAL transformation.

### The Simulink/Stateflow to HybridSAL Transformation

The main challenge of this transformation is that while hybrid automata represent the continuous time part and the discrete behavior separated, these concerns are scattered across a Simulink/Stateflow model. The transformation consists of two main steps:

1. Identifying the states of the Hybrid automata, and constructing the conditions
2. Converting the continuous time blocks into algebraic differential equations

Below we describe each of these steps[[2]](#footnote-2).

#### Identifying the states

Considering Fig. 6, we can see a Simulink/StateFlow diagram. The discrete behavior is defined by the discrete switch elements and a state machine. We can observe that the state *Low* does not define a value for *Switch3*. This means that *Switch3* can be either on or off, which needs a separate state to describe the different states for *Switch3*, i.e. state *Low* must be split into two states.

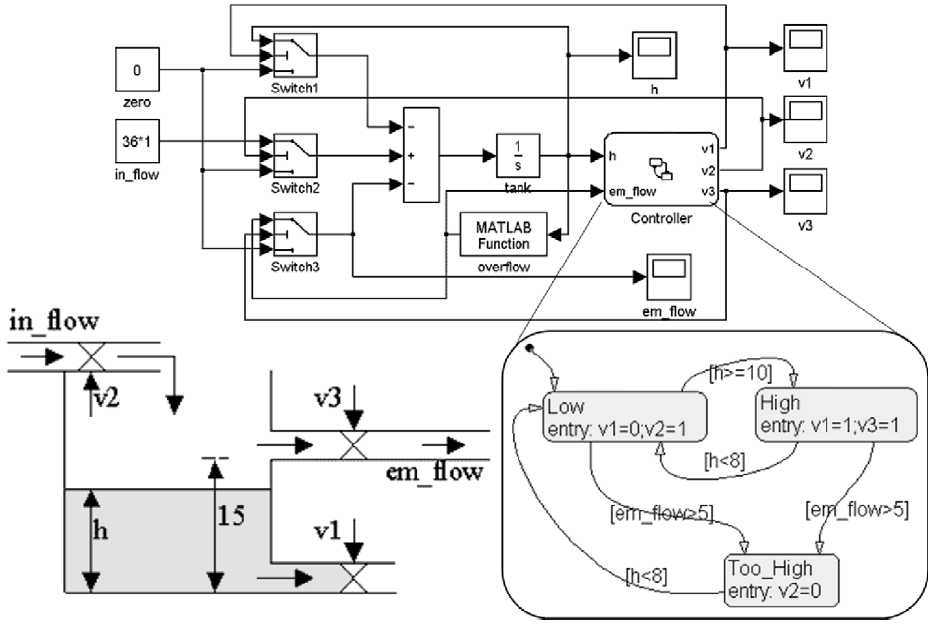


Figure 6. Sample Simulink/StateFlow model

This gives the notion of undefined state, for which new states must be generated. Some of these states might turn out to be dead states, so the resulting automata must be pruned after this state generation.

If we have a closer look at Fig. 2 again, we can see two additional examples for discrete behavior. The saturation block limits with a lower and upper bound:

* If the signal is between the lower and upper bounds, the result is the signal.
* If the signal is below the lower bound, the result is the lower bound.
* If the signal is above the upper bound, the result is the upper bound.

This can easily be modeled by a hybrid automaton (Fig. 7):



Fig.7. Hybrid Automata Model of the Saturation Block

The integrator with saturation is very similar:

* When the integral is less than or equal to the **Lower saturation limit**, the output is held at the **Lower saturation limit**.
* When the integral is between the **Lower saturation limit** and the **Upper saturation limit**, the output is the integral.
* When the integral is greater than or equal to the **Upper saturation limit**, the output is held at the **Upper saturation limit**.

Thus, we have to model this case as an integrator and a saturation component. There is one more component in our case study that does not have either a continuous or discrete counterpart: the bus creator groups signals together, which does not change the signals but group them together.

#### Creating Continuous Time Blocks

Since the Simulink representation is not in state space format, we have to transform the complex frequency domain components to a state space representation. This conversion has a well-known solution. For the elements with discrete behavior, the hybrid automata states identified according to the method described in the previous section determine the state of the switches and saturation components. Using these states, we need to focus on the continuous behavior in a certain state only.

### The Transformation of the Case Study

The result of the transformation is depicted in Fig. 8. The discrete behavior is defined by the saturation components.



Figure 8. The Hybrid Automata model of the CAC case study

We modeled the system with two automata sharing the signal *y*. The *SPEED\_SET* is labeled as *out*, while the *FLOW\_FB* is labeled as *FS*. The reset signal can easily be modeled by adding an extra state for each integrator state, and setting the input signal to the initial value, in our case, to zero.

# Compositional Verification of META languages with BIP

The compositional verification assumes that the individual components are correct (i.e. deadlock-free), and based on the integration, it infers properties for the composed system. It is a discrete problem, where components are in various states with respect to their interactions. To examine the composed system, we need to analyze their communications.

## BIP Summary

The central idea of the BIP (Behavior, Interaction, Priority) method is to build correct systems from correct components: it provides rules for the preservation of individual properties of the building blocks when they are composed. The BIP approach takes components and joins them with glue operators. Glue operators are first class concepts that describe the integration of the individual components. BIP models the interactions with ports and connectors. A connector is a set of ports that can be involved in an interaction, i.e. they are connected. Port attributes are used to model rendezvous and broadcasts. Interactions support passing data between ports.

The individual components are modeled with an automaton that keeps track of the communication state of the component. BIP models are interpreted by the BIP execution engine that is capable of evaluating certain properties such as deadlock freeness. BIP uses a transition system (modal flow graph) to describe the internal states. The implementation provides a rich set of communication primitives.

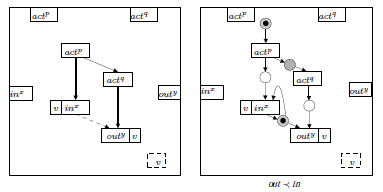


Figure 9. Synchronous BIP Sample[[3]](#footnote-3)

Synchronous BIP is a subset of the BIP language, where components are strongly synchronized by a common action that initiates steps of each component.

The behavior of a component in a step is described by an extended priority Petri net. The composition of synchronous components is defined by composing their Petri nets and their priority orders. The BIP tools are capable of proving deadlock freeness, based on its internal models. More precisely, Priority Petri nets are replaced by modal flow graphs (MFGs). In this representation, deadlock-freedom and confluence can be decided at low cost. For the subclass of well-triggered MFGs, deadlock-freedom and confluence can be guaranteed using simple syntactic conditions:

* Deadlock-freedom is guaranteed if the guards of ports having strong causes are trivially true
* Confluence is guaranteed by the non-interference of actions attached to independent ports

The composition properties can be ensured as follows. Given a set of synchronous components, a product component is obtained by gluing together the ports interconnected by interactions.

## Translation to BIP and back

This translation uses the results contributed by Sfyrla et al.[[4]](#footnote-4) In the description of the translation, we restrict ourselves to the discrete time part of ESMoL. Conceptually, the ESMoL discrete time primitives can be mapped to BIP components. Compound components must be recursively unfolded in order for us to obtain a flattened component model. ESMoL messages can be translated BIP events. Since ESMoL dataflow models are a different representation of Simulink models, we will use the Simulink terminology.

The compositional translation of discrete-time Simulink models can be described in detail as follows.

* Each Simulink block is translated to a unique synchronous component
* Dataflow and activation links are translated into connectors
* Basic Simulink blocks (e.g., operators) are translated to elementary synchronous components
* Structured Simulink blocks (e.g., subsystems) are translated recursively as composition of components
* A Simulink model requires an additional synchronous component which generates all activation events

A subsystem that consists of further subsystems A and B is validated if and only if Subsystem A and Subsystem B are validated, and BIP has been executed on the communication of Subsystem A and Subsystem B. Each and every subsystem has an attribute “validated”, which is set to false by default. After executing the validation tools these attributes are filled out. These attributes are part of the requirements language, and accessible by various design space exploration tool, such as DESERT.

# Conclusions and Future Directions

We have transformed our case study to hybrid automata digestible by HybridSAL. We have extended the existing conversion steps with modeling the reset signals and the saturation components.

Also, we have identified the following future directions besides implementing the current algorithms.

* Treating discrete blocks in hybrid states.
* Handling additional functions with non-polynomial elements. This must be an extension to the hybrid automata part.
* We need more case studies, e.g. plant/process to control with a physical model to extend our results to compositionality problems. Also, compositionality support must be added to HybridSAL.

1. The image is taken from HybridSAL: Modeling and Abstracting Hybrid Systems by Ashis Tiwari, 2003. More details on HybridSAL can be found in this document. [↑](#footnote-ref-1)
2. We rely upon research described in Semantic Translation of Simulink/Stateflow Models to Hybrid Automata Using Graph Transformations by Aditya Agrawal, Gyula Simon, Gabor Karsai, Electronic Notes in Theoretical Computer Science, Volume 109, 14 December 2004, Pages 43-56. Where we extend the results included in the paper, we give more explanation in the text, otherwise we refer the reader to the paper of more details. [↑](#footnote-ref-2)
3. Figures from V. Sfyrla, et al., “Compositional Translation of Simulink Models into Synchronous BIP”, Verimag Research Report, TR-2010-16, June 2010. [↑](#footnote-ref-3)
4. V. Sfyrla, et al., “Compositional Translation of Simulink Models into Synchronous BIP”, *Verimag Research Report, TR-2010-16, June 2010.* [↑](#footnote-ref-4)