Flat Heterogeneous Modeling

Frédéric BOULANGER Mokhoo MBOBI Mohamed FEREDJ

Supélec – Service Informatique
Plateau de Moulon, 3 rue Joliot Curie
F-91192 Gif-sur-Yvette cedex, France

***Abstract*— Most heterogeneous modeling environments allow only one model of computation at each hierarchical level of a model. The consequences are that (1) the transformations that occur at the boundary of two models of computation depend on the modeling tool and (2) the hierarchical structure of the model is perturbed by layers introduced to allow changes of model of computation.**

We introduce "heterogeneous interface components" to allow flat heterogeneous modeling. Such components have inputs and outputs that obey different models of computation. We present an execution model that allows the use of these components with any model of computation.

I. INTRODUCTION

The modeling and design of complex systems calls naturally for the use of several models of computation. For instance, an airborne imaging system contains mechanical and optical components, cryogenics for IR captors, signal processing algorithms and feedback mechanisms. Some of these components may be implemented either in hardware or in software. Each of these components requires specific know-how and tools linked to specific models of computation. Even simple systems have at least two parts: a control part — which decides what to do and when — and a data processing part. Moreover, it is often necessary to introduce a third model corresponding to the environment of the system.

We consider component based modeling, a technique where the model of a system is built by assembling components that interact according to models of computations [6], [3]. A model of computation is the set of laws or axioms that rule the interaction of the components. Concurrent sequential processes, state machines, synchronous dataflow, discrete events and Kahn process networks are examples of models of computation. According to the vocabulary used in the Ptolemy II project [1], a domain is an implementation of a model of computation: it is a set of rules for computing the behavior of the components of a model. In the following, we often use "domain" to mean "model of computation".

Heterogeneous modeling is simply modeling using several models of computation or domains. Since most systems are heterogeneous in nature, heterogeneous modeling provides more natural and more complete models [4]. For instance, being able to use both state machines and synchronous dataflow components allows to describe explicitly the control in the model of a digital signal processing system. If we were limited to the synchronous data flow domain, control and data processing would have to be coded together and the model would be less expressive and more difficult to maintain [5].

Many modeling and design environments support heterogeneous modeling, but they generally focus on a small set of models of computation: continuous and discrete signals for electrical engineering or state machines and differential equations for hybrid systems. Since they use few domains, such environments can define the union of the domains that they support, and so allow components to obey several models of computation. In such environments, a digital to analog signal converter can be modeled as a single component with inputs that obey a continuous signal model of computation and outputs that obey a discrete signal model of computation. However, environments that support a large or extensible number of domains cannot build this union, so they require that each component obeys only one model of computation. Since components that are connected obey the same model of computation, domain changes can only occur at the boundary of

a component: the model of computation used inside the component may not be the same as the one used outside. This leads to hierarchical heterogeneous modeling [2].

II. Hierarchical Heterogeneous Modeling

Hierarchy is an efficient way of managing complexity by hiding internal details that are not pertinent at a given level of modeling. When you look inside a component, you may either see a low level description of the behavior of the component — when the component is atomic or primitive, or you may see a model of this component in the same modeling environment — when the component is composite. In both cases, since the internal structure of the component is hidden to the external model of computation, it may use a different domain. It is still necessary to define how the internal and the external domains interact and how data is transformed when crossing a domain boundary, but hierarchy makes heterogeneous modeling easier.

The main drawbacks of this approach are:

- the hierarchy of the model is coupled to the changes of model of computation and may not reflect the effective structure of the system;
- components that have inputs or outputs that obey different models of computation cannot be used;
- what happens when data crosses the boundary between two domains depends on the modeling environment.

The first issue may obscure the natural structure of a model with artificial layers introduced to allow changes of model of computation.

The second issue forbids the use of components that have terminals which obey different models of computation. For instance, an analog to digital converter could be modeled in a continuous time domain, the digital outputs being considered as continuous signals with sharp changes. If such a model is close to the reality, it is not at the right abstraction level when you want to consider the outputs as discrete sequences of values. On the contrary, the analog to digital converter could be modeled using a discrete domain where the continuous inputs would be considered as sequences of discrete samples, turning the device into a re-sampler or a no-op.

The third issue hides the transformations that occur at the boundary of two domains inside the "edge of the components". These transformations depend on the modeling tool and are therefore not explicitly stated in the model of a system. Two approaches may be used to solve this problem:

- allow the designer to edit the edge of the components to specify how data is transformed when it goes through it;
- move these transformations from the edge to the core of the components.

However, moving the transformations to the core of the component makes the internal specification of the component depend on the domain in which it is used, what impairs modularity and reusability.

These three issues disappear when we allow several models of computations to coexist at the same level of the hierarchy of a model, what we call "Flat heterogeneous modeling".

III. Flat Heterogeneous Modeling

The real issue is not hierarchy in itself but the impact of the coupling between domain changes and hierarchy on the design and maintainability of applications. Therefore, "flat heterogeneous modeling" does not mean "heterogeneous modeling without hierarchy", it means that the hierarchy is used to reflect the structure of the model without being perturbed by artifacts introduced by changes of model of computation.

With flat heterogeneous modeling, a given level in the hierarchy of the model may contain components that use different models of computation. There are two possibilities for that:

1) components obey only one model of computation, but we can make connections between terminals across domains;
2) only terminals that belong to the same domain can be connected, but a component may obey several models of computation.

We don't use the first possibility because the change in semantics between domains occurs along the connections between terminals, and connections are generally not considered as active entities in modeling environments. The second possibility preserves the semantics of data along connection between terminals and has the advantage of supporting

components that have terminals that obey different models of computation. The change of semantics between domains occurs inside those components as part of their behavior and is therefore an explicit part of the model of the system. We call such components "heterogeneous interface components" (HIC). As shown by the analog to digital converter example, HICs appear naturally in models, however they raise the issue of making them obey several models of computation in a consistent way.

We consider HICs in the context of generic heterogeneous modeling. When the available models of computation are known and in limited number, it is more efficient to use their union to support HICs. The problem that we address is the use of heterogeneous interface components with any number of models of computation that are not known in advance. A typical model of a given system may not use more than two or three domains, and that is why special purpose flat heterogeneous modeling tools are useful and efficient. However, when modeling a complete system within its environment at a high level, the number of models of computation involved increases. Even if each system does not use more than three or four models, the number of possible combinations of a small number of models of computation chosen among all possible models of computation is too large to use the "union of models" approach.

IV. Heterogeneous Interface Components

A HIC has terminals that obey different models of computation (MoC), therefore its behavior can be decomposed into as many sub-behaviors as there are MoCs, and these sub-behaviors are coupled: the behavior of the HIC according to one MoC can influence its behavior according to another model of computation.

Each sub-behavior of an heterogeneous interface component is a bridge between a model of computation and the global behavior of the HIC. When a HIC interprets an input, it translates the meaning of this input in the associated MoC into an internal meaning for the HIC. When a HIC produces output, it translates the information gathered from its inputs into the semantics of this output according to the associated MoC. Therefore, specifying the behavior of an heterogeneous interface component amounts to building an internal representation of the semantics of the inputs according to their respective models of computation, and translating this internal representation into outputs.

Heterogeneous interface components allow explicit specification of how data from one domain is interpreted, and of how this interpretation is used to produce data for another domain. They are not limited to interfacing two domains only: a HIC may use as many domains as needed.

A. Example

We consider the simple example of a signal rectifier to illustrate the difference between flat and hierarchical heterogeneous models. The system, as shown on the left side of figure 1, receives an input signal as a sequence of data samples. A multiplier multiplies each sample either by 1 or by -1 and toggles between the two behaviors according to events produced by a sign-change detector.

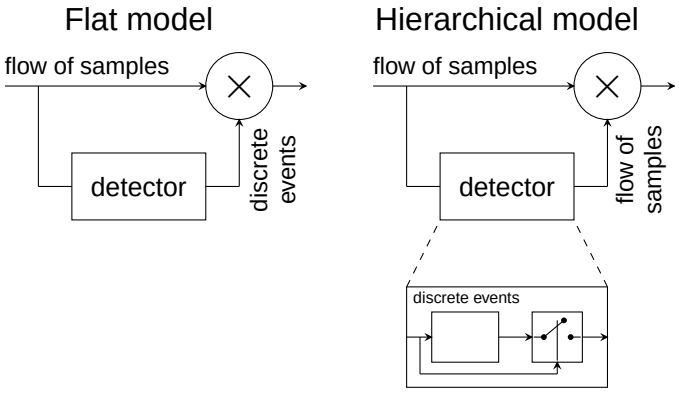

Fig. 1
Example system

The detector has an input that receives a flow of data samples and an output that produces a discrete event each time the input changes its sign. The multiplier has an input and an output that convey flows of data samples and an input that receives discrete events. This example is therefore a flat heterogeneous model of the system and contains two heterogeneous interface components.

The right part of figure 1 shows a hierarchical model of the same system. The top level uses flows of data samples, and the behavior of the detector is modeled using discrete events. When the flow

of samples enters the detector, it is converted to a sequence of valued events. When an event is produced at the output, its value is used to build a data sample in the outer domain. This is only an example of what may happen at the boundary of a component, and the important point is that these transformations depend on the modeling tool and are not specified in the model of the system.

Since the data flow model of computation in which the detector is used requires that a sample of data be produced on the output each time a sample is consumed on the input, the discrete event behavior of the detector must respect this condition. So even if the input signal does not change its sign and no event has to be produced, the detector must produce something on its output to obey the semantics of the outer domain. Here, we have put a sampler which uses the value of the last emitted event to produce an output each time an input sample is consumed. We have to put this sampler in the internal model of the detector because of the semantics of the external model of computation, so the implementation of the detector depends on the context in which it is used, what impairs modularity and reuse.

B. Supporting flat heterogeneous models

Regarding practical aspects, we must describe how the heterogeneous behavior of a HIC is specified, and how it is interpreted to compute the behavior of the model of a system. Since we don't plan to develop a full featured modeling platform, our main goal is to be able to use heterogeneous interface components with existing models of computations. However, these models of computation don't support HICs, or if they do, not in such a generic way.

Our approach is to transform the original flat heterogeneous model by adding one level to its hierarchy and to build homogeneous subsystems at the next level. The top layer is managed by a new "Heterogeneous" model of computation that schedules the homogeneous subsystems and delegates the computation of their behaviors to their respective domains. This execution model uses hierarchy at simulation time to insulate models of computations, but this has no impact on the structure of the model of the system and allows components to be designed without considering the context in which they will be used.

For each domain used by a HIC, we build a component that represents the HIC in this domain. We call this component the "projection" of the HIC onto the domain. From the point of view of the domain, this component is a regular component, with terminals that obey the semantics of domain, the other terminals being masked during the process of the projection. However, the projection of a HIC in a domain can communicate with other projections of the same HIC in other domains. The projection of the heterogeneous interface components onto their domains, the communication between the projections of a HIC, and the scheduling of the homogeneous subsystems are managed by the "Heterogeneous" domain which implements an execution model for HICs.

V. Execution Model

The first step in the execution of a flat heterogeneous model is to project the HICs onto each of the domains that they use, the behavior of each projection being computed by the corresponding domain. The projected components must behave exactly as the other regular components that belong to the target domain, so the terminals of the HIC that obey other models of computation are hidden during the projection.

A. Clustering of the homogeneous components

Once every HIC has been projected onto each of the domains it uses, the resulting components are clustered into homogeneous subsystems that contain only regular components belonging to one domain. The behavior of each subsystem can therefore be computed by applying the rules of a regular domain. Our Heterogeneous domain must schedule the subsystems in order to compute the behavior of the whole system from the behavior of the homogeneous subsystems.

This is illustrated for a very simple system on figure 2. The system contains a component A which has one output that obeys the model of computation D_1, another component B which has one input that obeys the model of computation D_2, and these components are interconnected through an heterogeneous interface component H. Since H uses

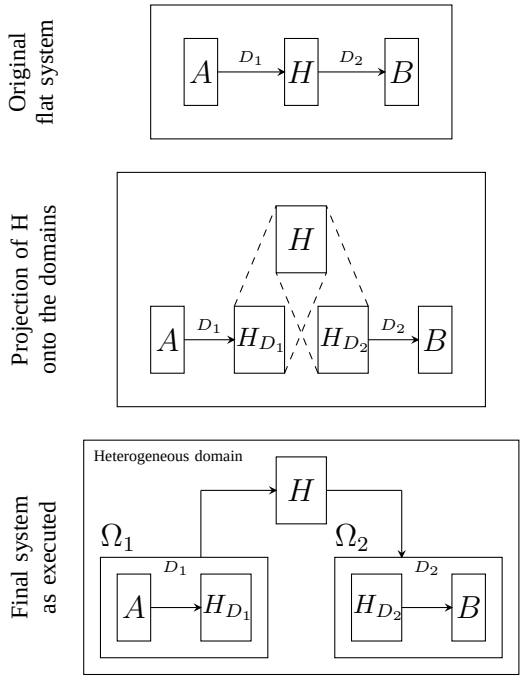

Fig. 2

TRANSFORMATIONS OF A FLAT HETEROGENEOUS MODEL

two models of computations, it has two projections noted H_{D_1} and H_{D_2}. H_{D_1} has only an input because the projection onto D_1 hides the output of H since it obeys another model of computation. Similarly, H_{D_2} has only an output because the projection onto D_2 hides the input of H which obeys D_1.

The components are grouped into homogeneous subsystems that can be scheduled by the heterogeneous domain. In our example, we have one subsystem for each domain used, and the connections from Ω_1 to H and from H to Ω_2 code the dependency of the behavior of H_{D_2} on the behavior of H_{D_1}. Inside Ω_1 and Ω_2 the scheduling of the components is done according to the local domain, but the scheduling of Ω_1 and Ω_2 is done by the heterogeneous domain according to the topology of the system.

B. Virtual communication channels

Usually, communication between components occurs along communication channels that appear as connections between components. In the clustered system that results from the transformation of a flat heterogeneous system, the communication channels between the projections of a HIC onto different domains cannot have a matching connection because the communication occurs inside the HIC. However, this communication channel must be taken into account for scheduling the homogeneous subsystems. We call such a communication channel a "virtual heterogeneous channel", and the communication over such a channel obeys the rules of the heterogeneous domain.

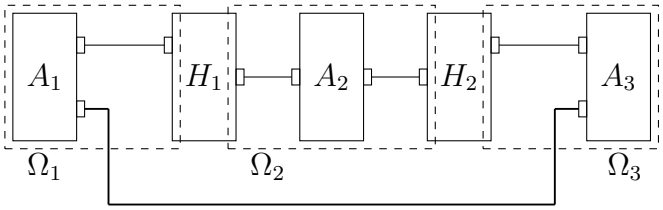

Fig. 3

HOMOGENEOUS COMMUNICATION CHANNEL

Another kind of virtual communication channel is necessary to handle the case where two components that obey the same model of computation and are connected, cannot be put into the same subsystem. The example on figure 3 shows a system with two HICs (the projections of the HICs are represented by their intersection with the dashed frame of the subsystems). Although A_1 and A_3 obey the same model of computation, they cannot be put into the same subsystem because that would lead to a system with no possible schedule. If we put A_3 in Ω_1, Ω_1 depends on Ω_2 because it contains A_3 which depends on A_2 through H_2, and Ω_2 depends on Ω_1 because it contains A_2 which depends on A_1 through H_1. However, when A_1 and A_3 are in different subsystems, we cannot connect them as they are in the flat system. We call such a communication channel a "virtual homogeneous channel". These channels are implemented using a source relay component which transmits data to a matching target component.

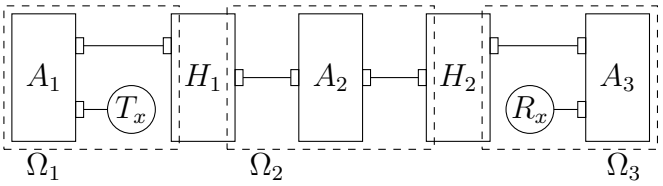

Fig. 4

RELAY COMPONENTS

On the example of figure 4 the data which is

available on the input of T_x is also available on the output of R_x, and the scheduler of the heterogenous domain will ensure that Ω_1 is computed before Ω_3 so that T_x can transmit the value to R_x before the output of R_x is used.

C. Building subsystems

The algorithm used to build the homogeneous subsystems examines each component in an order which is compatible with the topological sort of the system. For each component A, it looks for a subsystem that uses the same model of computation as the component. If such a system Ω exists and there is no path between any component of Ω and A that goes through a HIC, A is put into Ω, else a new subsystem is created to host A, and relay components are created if there is a virtual homogeneous channel between a component in Ω and A.

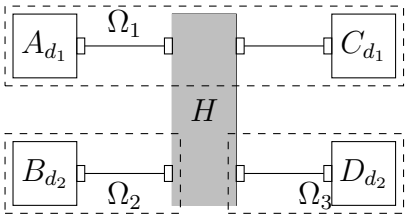

Fig. 5
AN OPTIMAL PARTITION OF A SYSTEM

These conditions ensure that there won't be any cross-dependencies between subsystems, so that it will be possible to find a schedule. However, they may lead to the creation of more subsystems than necessary. Figure 5 shows an optimal partition of a system that cannot be reached by our algorithm: we do not allow to put A and C in the same subsystem although they both obey d_1 because there is a path from A to C that goes through a HIC.

D. Fake ports

A connected system may be divided into subsystems that are not connected. For instance, the system shown on figure 6 is divided into three subsystems, and the second subsystem is not connected because the connections from A to H_1 and H_2 are lost when they are projected on Ω_2. This should not be a problem, but some models of computation do not allow unconnected systems, so we must make the subsystem connected without changing its behavior.

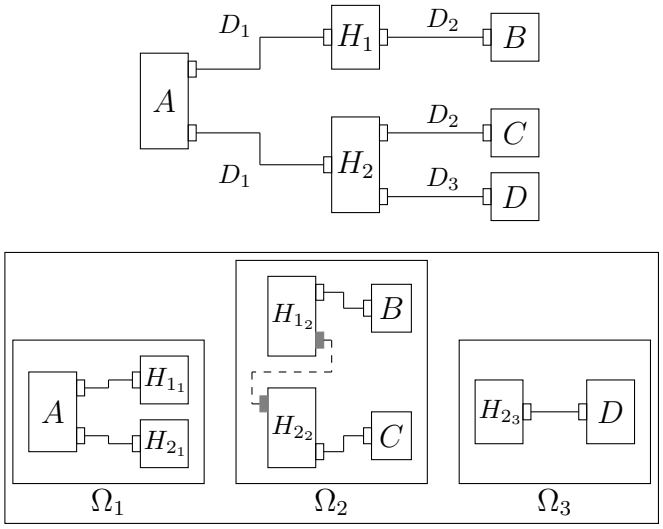

Fig. 6
VIRTUAL PORTS

For this, we add fake terminals to the projections of the HICs when necessary, and we connect them to make the subsystem connected. Such connections between fake ports have no effect on the behavior of the system. They are only "syntactic sugar" for domains that do not support unconnected systems. The connection between fake ports appears as a dashed line on the figure.

E. Scheduling

The scheduling of the subsystems must respect the dependencies induced by the virtual channels. However, virtual homogeneous channels are created only when there exists a path between two components that goes through a HIC. Therefore, the dependency induced by a virtual homogeneous channel is also induced by a chain of virtual heterogeneous channels, and it is sufficient to take only virtual heterogeneous channels into consideration when scheduling the subsystems.

Since our heterogeneous execution model is generic and works with any model of computation, we do not consider the semantics of the models of computation of the subsystems when scheduling. The only assumption we make is that a connection from an output to an input makes the consumer depend on the producer. The scheduling algorithm

is therefore simple since any ordering of the subsystems that is compatible with the pre-order induced by the topology of the connections between components is a suitable schedule.

A consequence of this algorithm is that there cannot be loops in the graph of the system. This is the price to pay for the support of any model of computation. The scheduling of a system that contains loops depends on the semantics of the model of computation. A way to break dependency loops is to insert a delay in the loop. However the semantics of a delay is itself very dependent on the semantics of the model of computation. Therefore, if we consider the subsystems and their models of computation as black boxes, we cannot allow dependency loops between subsystems. However, if a loop is local to a subsystem and if the corresponding model of computation supports loops, the loop is accepted and its semantics will be given by the domain of the subsystem.

VI. Implementation in Ptolemy II

Our flat heterogeneous execution model is implemented in the Ptolemy II framework which support numerous models of computation. The graphical user interface does not support flat heterogeneous models yet, but it is possible to create such systems through the Java API of Ptolemy II. One of the issues raised by heterogeneous interface components is the coding of their intended behavior. We discuss below how we solve this problem in the current implementation of our execution model.

A. Specifying the behavior of a HIC

Each time a projection of a HIC onto a domain is activated by the local domain, the HIC must either process inputs, produce outputs or update its internal state. The scheduling algorithm of the heterogeneous domain ensures that all the projections of a HIC that take inputs are activated before any projection that must produce outputs is activated. However, when we design a HIC, we do not know in which order its inputs will be available because we do not know how it will be projected on the domains it uses. It is neither possible to specify the behavior of the HIC globally for each of its domains, because it may happen that several terminals that use the same model of computation be projected into different subsystems.

Therefore, the only solution to specify the behavior of a HIC is to specify how its state is updated for each possible set of known inputs, and to specify how to compute its outputs from the known inputs and the current state. This makes programming HICs less simple than programming regular components because the code must check which inputs are known before processing them.

VII. Conclusion

Flat heterogeneous modeling allows more natural modeling of heterogeneous interface components and gives more control on the semantics of the interactions between models of computation to the designer. We have presented an execution model that allows the use of flat heterogeneous models with implementations of models of computation that were not designed for this. The separation between the changes of domain and the hierarchical structure of the model, as well as the specification — in the model of the system — of what happens at the boundary between domains contributes to the modularity and maintainability of the models.

References

[1] S.S. Bhattacharyya et al
"Heterogeneous Concurrent Modeling and Design in Java", Volumes I to III, Memorandum UCB/ERL M04/15 to 17 University of California at Berkeley, June 24, 2004
`http://ptolemy.eecs.berkeley.edu/papers/04/`

[2] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvigy, S. Neuendorffer, S. Sachs, Y. Xiong
"Taming Heterogeneity — the Ptolemy Approach"
Proceedings of the IEEE, v.91, No. 2, January 2003

[3] Edward Lee, Stephen Neuendorffer and Michael Wirthlin
"Actor-Oriented Design of Embedded Hardware and Software Systems"
Invited paper, Journal of Circuits, Systems, and Computers, Vol. 12, No. 3 pp. 231-260, 2003.

[4] Jie Liu, "Responsible Frameworks for Heterogeneous Modeling and Design of Embedded Systems"
Ph.D. thesis, Technical Memorandum UCB/ERL M01/41, University of California, Berkeley, December 20 2001.

[5] Xiaojun Liu, Jie Liu, Johan Eker, and Edward A. Lee
"Heterogeneous Modeling and Design of Control Systems" in *Software-Enabled Control: Information Technology for Dynamical Systems*
Tariq Samad, Gary Balas ed., Wiley-IEEE Press, April 2003

[6] H. John Reekie and Edward A. Lee
"Lightweight Component Models for Embedded Systems"
Technical Memorandum UCB ERL M02/30, University of California at Berkeley, October 30 2002.