# Non-hierarchical heterogeneity

Mokhoo Mbobi
Mokhoo.Mbobi@supelec.fr

and

Frédéric Boulanger
Frederic.Boulanger@supelec.fr

and

Mohamed Feredj
Mohamed.Feredj@supelec.fr

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

## ABSTRACT

Modeling languages and platforms generally use hierarchy to combine heterogenous subsystems. This approach avoids the combinatorial explosion of the number of interfaces between computation models, but it forbids the use of components which have inputs or outputs that obey different models.
We propose here a non-hierarchical heterogeneous approach based on heterogenous-interface components (HIC). Such components allow the connection of subsystems that behave according to different computation models or domains.
This approach still supports hierarchy in order to structure complex systems, but, by allowing the use of flat heterogenous models, it makes distributed simulation easier since it is suppresses the necessity to access the internal structure of heterogenous components, and make the explicit specification of what happens at the border of two domains possible.

**Keywords:** heterogenous modeling, hierarchical design, software engineering.

## 1. INTRODUCTION

The design of a system generally involves the interconnection of several modules that interact or supervise other modules. The different parts of the system may belong to different technical domains such a signal processing, control science, or communication protocols. Each of these domains has its design tools which correspond to computation models. A single technical domain may involve several computation models: in a communication system, the protocol may be described with state machines and events, while the processing of the signal may be handled with Z or Laplace transforms, leading to a discrete or a continuous time model.

Heterogenous design languages and platforms support the concurrent use of several models of computation, but if a subsystem uses a different model of computation than another, it must be placed at a different level in the hierarchy of the system. This makes the interactions between computation models much simpler, but it leads to a strong coupling between hierarchy and model changes. Moreover, since the interface of a component appears at a single level of the hierarchy, it must use a single computation model. This forbids the use of components which work at the border of several computation models.

To make things clearer, let's consider a level-crossing detector which monitors a signal and produces an event each time the signal crosses a given level in a given direction. Such a component receives the signal on one of its inputs and must obey the semantics of this signal (continuous or discrete time for instance). The event it produces when the input signal crosses the level appears on one of its outputs, and this output must therefore obey a discrete event semantics.

Allowing the use of such heterogenous-interface components has two main advantages. First, it avoids the use of obscure hierarchical constructs to model simple functionalities, and second, it allows the explicit specification of the mapping of one semantics onto another. In our ex-

ample, with the currently available heterogenous modeling tools, we could choose to use a discrete time semantics for the level-crossing detector, and to map the occurrence of an output event to a non-null sample of the output signal. We could also choose a discrete event semantics and map the samples of the input signal to periodic events. But both possibilities do not match the effective nature of this component. It may even happen that adapting the semantics of the signals leads to incorrect behaviors. For instance, in a continuous time computation model, using a non-null sample to represent an event could lead to the loss of events since a signal which is always null excepted for a finite number of samples may be seen as a null signal by the differential equations integrator.

After a brief discussion about the different uses of hierarchy, we will analyze the support of heterogenous modeling through hierarchy, and then present our new non-hierarchical heterogenous approach. We will then give two examples using the discrete events and the synchronous data-flow semantics to illustrate this approach.

## 2. HIERARCHICAL DESIGN

Hierarchy allows to consider a complex system as composed of simpler sub-systems. It is therefore a tools to manage the complexity of a system. This complexity may come from the behavior of the system or from its structure.

### Structural hierarchy

Structural hierarchy is used to identify self-contained sub-systems and to define their interface to the other sub-systems. This interface may consist of a set of signals, or at a higher level of abstraction, of communication channels which encapsulate more complex communication protocols. Structural hierarchy allows to design each part of a system in a separate way once the interfaces have been specified.

### Behavioral hierarchy

Behavioral hierarchy is a way of considering a complex process as the result of sequential or concurrent simpler processes. The composition of processes makes use of communication and synchronization primitives such as termination detection, process activation or suspension, rendez-vous or exceptions. Exception handling may be considered as behavioral hierarchy since it can be seen as the termination of a process and the activation of the exception handling process [6].

In [7], the Ptolemy team at the University of California at Berkeley defines yet another type of hierarchy called “synchronization hierarchy” which can be considered as a special case of behavioral hierarchy.

## 3. HETEROGENOUS DESIGN

A typical heterogenous system looks like shown on figure 1. It is composed of several sub-systems which may use different computation models and may even be targeted toward either hardware or software. An execution model takes care of the activation of the sub-systems according to a suitable schedule, and a communication layer allows the sub-systems to exchange data. The heterogeneity we are talking about in this article is at the modeling and design level. It is only related to the rules that govern the interactions between the parts of the model of the system. An implementation of the system may use a single set of rules onto which the different modeling approaches would be mapped. On the contrary, two systems which are modeled using the same model may be implemented with different rules, for instance the continuous time model may be used to model both a mechanical system and an electrical system.

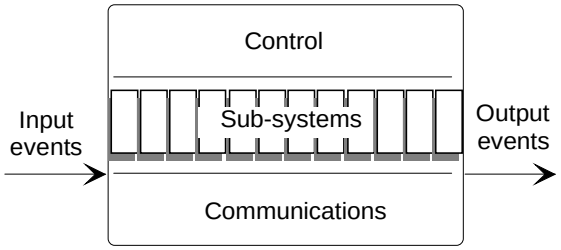
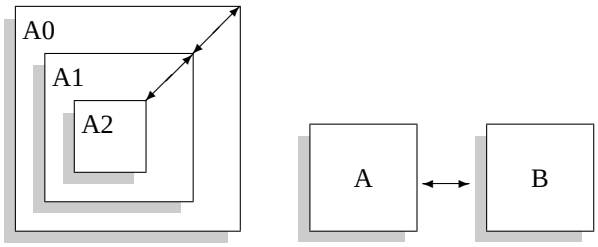



Figure 1. Typical structure of a system

### Origins of the heterogeneity

The hierarchical structure of the systems and the interaction between different parts of a system may lead to the interconnection of components that do not use the same computation model. When one of the components is contained in the other, we have “vertical heterogeneity”. In the other case, we have “horizontal heterogeneity”.




Vertical heterogeneity Horizontal heterogeneity

In both cases, we must handle the fact that the components can communicate, but have distinct notions of what communication is since they use different models of computation. When the heterogeneity is vertical, the semantics of communications is adapted when a hierarchical level is

crossed. When the heterogeneity is horizontal, we have to adapt explicitly the semantics of the messages.
A hierarchical heterogenous design [4], adds the fundamental issue of the choice of the top-level computation model which governs the global behavior of the system. For many applications, the control depends on incoming data, and the processing of data depends on the control. However, neither a control-oriented nor a data-processing oriented top-level computation model may be entirely satisfying. Using horizontal, non-hierarchical heterogeneity allows to consider both aspects of the system at the top-level.
In the following, we will not focus on a particular formal design language, but since we use the Ptolemy platform for our experiments, we will use its computation models (called *domains*) and its vocabulary. Since Ptolemy does not support non-hierarchical heterogeneity, we will have to propose an extension of its execution model.

**Hierarchical heterogenous design**

Ptolemy comes with generic actors[1] which can be used in several domains[2]. The host domain defines the way the actors are activated and communicate. A generic actor, being able to adopt a sensible behavior in several domains, is *domain-polymorph*. Domain-polymorphism is the property that allows an actor to behave properly in any domain, just as polymorphism in object-oriented languages allows to consider any instance of any subclass of a given class C as an instance of C.
Generic actors and domain-polymorphism allow the use of common services like reading data from files or displaying data in graphs in almost all domains. However, a generic actor adopts the semantics of the domain in which it is used, so there is no real heterogeneity in genericity.
In Ptolemy, a model can have only one director, so it can use only one domain. An heterogenous system must therefore contain several models, each one having its own director, and the models must be structured in a hierarchy, as shown on figure 2.
This system uses domain A at the top level. One of its actor behaves as specified by another model which uses domain B, and another behaves a specified by a finite state machine. Any state of the state machine can be refined as another state machine or as a model using domain A.
This hierarchical heterogeneity is used in many architecture description languages such as SpecC [1], Superlog [2] and so on. It is also used in heterogenous modeling and design platforms such as SystemC [3], El Greco [5] and Ptolemy II [7].

**Non-hierarchical heterogenous design**

Non-hierarchical heterogeneity allows the use of actors that obey different computation models at the same level of the hierarchy. This implies that a model can contain actors

[1] In Ptolemy, a component of a system is called an *actor*
[2] A domain is the implementation of a computation model

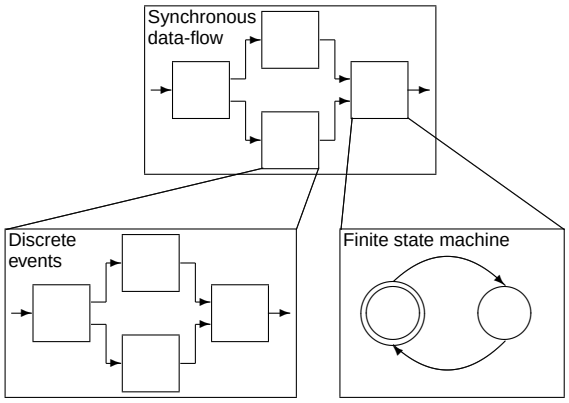



Figure 2. Hierarchy of heterogenous models

which communicate according to different semantics, and therefore that an actor may have inputs and outputs of different natures. Such actors have an heterogenous interface, and we will see that supporting heterogenous-interface actors is the key to non-hierarchical heterogenous modeling.

Since the semantics of a computation model is enforced by a director, a non-hierarchical heterogenous model may contain several directors, each director being in charge of the actors which use its computation model. An heterogenous-interface actor will be handled by as many directors as it has different types of inputs or outputs. However, these directors must be coordinated to take into account the fact that the reaction of an actor according to a computation model may have consequences on its behavior according to other computation models.

## 4. THE NON-HIERARCHICAL APPROACH

One of the main advantages of non-hierarchical heterogeneity is the ability to specify what happens when data goes from one computation model to another. In the hierarchical approach, the data is either transformed when it crosses a hierarchical boundary (for instance by adding a time stamp when some data goes from an untimed domain to a timed domain), or it must be interpreted in the receiving model (for instance by considering a null value as a lack of event and a non-null value as an event occurrence).

By using heterogenous-interface components, we eliminate the need for such implicit semantics conversions and give entire control of the interpretation of information to the designer of a system. An heterogenous-interface component communicates through its inputs and outputs according to the semantics of their associated domain. The interpretation of data from a domain into another domain is explicitly stated in the behavior of the actor.

### Structure of a HIC

An heterogenous-interface component is a component which has ports (inputs or outputs) which communicate according to different computation models, as shown on figure 3.

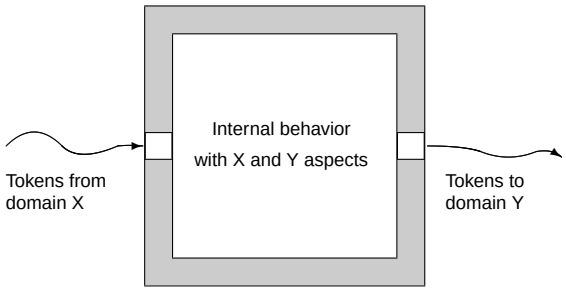



Figure 3. Structure of a HIC

Such a component may be either atomic (its behavior is not expressed in terms of other actors but coded in a programming language), or composite, in which case it will contain other actors and heterogenous-interface components.

The heterogenous communication interface of HICs implies that they have an heterogenous activation interface: such a component must be activated in the context of a director when it receives data from an actor which is managed by this director. When the reaction to its activation by a director triggers the production of data toward an actor which is managed by another director, it must be activated by this second director in order to be able to transmit this data.

This shows the necessity to define an heterogenous execution model which governs the way different directors activate a HIC in an heterogenous model.

### Heterogenous flat models

For the discussion about the heterogenous execution model, we consider a very simple model which contains two actors and a HIC. Actor D1 belongs to domain D1 and actor D2 belongs to domain D2. Both have ports that communicate only according to the semantics of their domain. Actors D1 and D2 communicate through the HIC which will make the data produces by D1 meaningful to D2. The overall behavior of this system as shown on figure 4 can be decomposed in three parts:

1. the D1 to HIC part,
2. the behavior of the HIC,
3. the HIC to D2 part.

The connections between the components lead to the following schedule for the operation of the model:

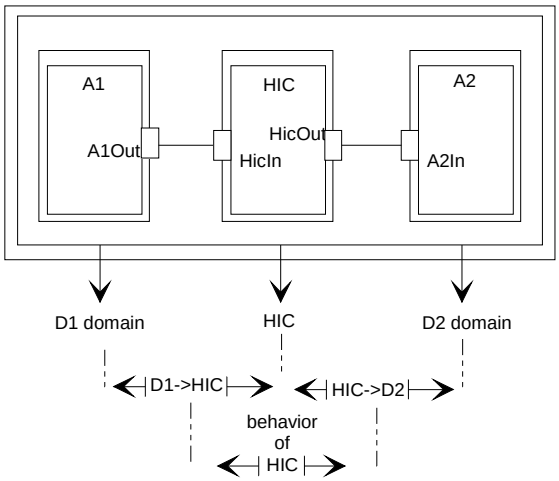
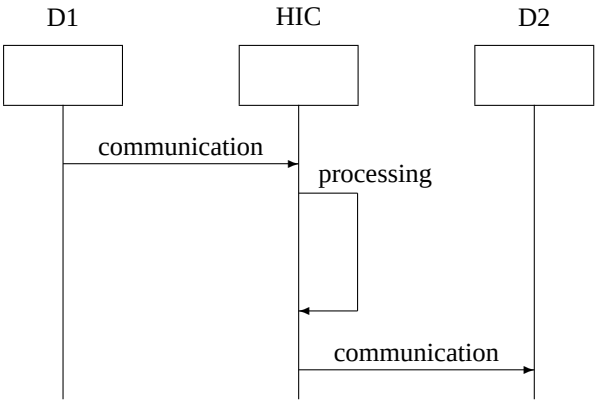



Figure 4. Heterogenous model with a HIC




### D1 to HIC part

The director of the D1 domain makes the D1 actor react. The D1 actor produces data on its output, and since this output is connected to the D1 input of the HIC, the director of the D1 domain makes the HIC react. From the point of view of the D1 director, the HIC is seen as an actor in the D1 domain. This means that the D2 output of the HIC is not seen by the D1 director.

### HIC processing

When the HIC is activated by the D1 director, it sees the data on its D1 input and processes it. During this processing, it needs to produce data on its D2 output. However, producing data in domain D2 requires that the D2 director is informed of this production so that the data can be routed to the input of the D2 actor. Therefore, the HIC must be activated by the D2 director so that it can send its data on its D2 port.

### HIC to D2 part

Since the output port of the HIC is connected to the input port of the D2 actor, the D2 director activates the D2 actor which detects and reads the data on its input, and processes it.

## 5. IMPLEMENTATION IN PTOLEMY II

During the HIC processing part of the behavior of an heterogenous model, we have seen that when the HIC needs to produce some data in a domain for which it is not currently activated, it must postpone the production until it is activated by the director of this domain. The problem is that the HIC must inform the director that it needs to be activated in this domain. A simple solution would be to allow the HIC to request an activation from the director. However, the example we have chosen here is very simple since the HIC interacts with only two domains. When a HIC interacts with more domains, the order in which those domains must activate it must be determined according to the topology of the model, and even according to the semantics of the domains.

In our experiments with heterogenous models in Ptolemy II, we try to reuse as much as possible of what already exists. So, adding a method to the directors classes to allow a HIC to request an activation is something we try to avoid. A solution which involves no changes to existing classes in Ptolemy II is to introduce a new director which we call an *HDirector*. An HDirector does not implement a new computation model: it adds support for HICs. It is a kind of *meta-director* which schedules the activation of HICs by the regular directors (those who implement computation models). The role of the HDirector is also to let the regular directors see only the actors which belong to their domains. So, in our example, the D1 director would only see the D1 actor and the HIC as a D1 actor, with only its input. The D2 director would only see the D2 actor and the HIC as a D2 actor, with only its output. Since both regular director do not see the connection between the HIC and the actor in the other domain, the HDirector must maintain the causality chain between the consumption of data by the HIC on its D1 input and the production of data on its D2 output.

endfigure

Figure 5 shows the complete structure of an heterogenous model with an HDirector which coordinates the execution of several regular directors.

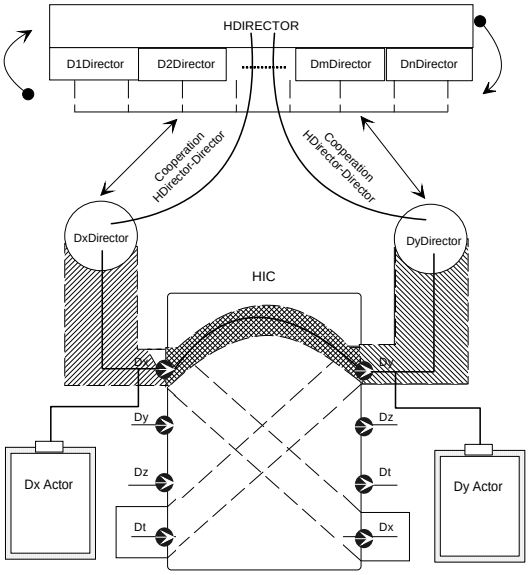



Figure 5. Structure of an heterogenous model

## 6. TECHNICAL INTEREST AND EXAMPLES

Non-hierarchical heterogenous modeling and heterogenous-interface components are very interesting for the modeling of many systems where software more and more importance such as telecommunication and navigation systems. These systems are evolving from almost pure signal-processing systems toward systems with the complex control needed to support several operating modes or communication protocols. In such systems, the control part sometimes changes more often than the signal processing part.

Heterogenous modeling and design has improved the design of such systems by allowing the explicit specification of both control and data-processing. Without heterogeneity, the only way to add control to a data-processing application way to hide this control inside the behavior of the data-processing operators.

With non-hierarchical heterogeneity, we get the ability to define explicitly how data is interpreted when it goes from one domain to another, and we are able to model heterogenous-interface components. Such physical components exist and are used in hardware. For instance, a modem chip has analog inputs and outputs to the phone line and has also digital inputs and outputs connected to a bus. They can easily be modeled using HICs.

Our first example is shown on figure 6. A signal S is observed, and each time its value crosses a given level, an event occurs. The events may be processed by a feedback subsystem to regulate some property (phase or frequency) of the signal. Here, we have chosen to model the signal by its Z-transform, so we use the Synchronous Data-Flow domain of Ptolemy II. The level-crossing detector monitors the signal and produces events, so it has an SDF input and a Discrete Events (DE domain of Ptolemy II) output. The feedback component is also an HIC since it processes events in the DE domain and produces data for the signal generator in SDF.

Our second example is also related to the DE and SDF domains of Ptolemy II. A packet-switching communication system transports both data and voice packets. The arrival of a packet is an event that triggers a process which discriminates voice and data packets. Voice packets are opened and their contents undergoes a filtering or compression process. The resulting samples are then put back into packets to travel on the network along with the unmodified data packets.

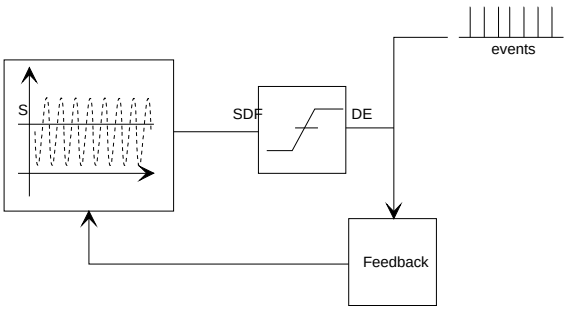



Figure 6. Simple exemple of an heterogenous model

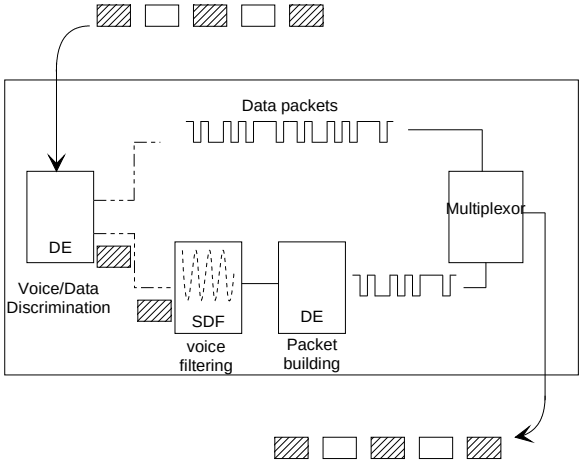



Figure 7. Packet switching and voice filtering

The modeling of such systems is more difficult and less intuitive when heterogeneity is coupled to hierarchy.

## 7. CONCLUSION

Hierarchical heterogeneity improves the quality of design tools by allowing the designers to use the most suitable model for each part of a system. However, we have shown that, when each change of computation model implies the creation of a new hierarchical level, obscure constructs appear in the structure of the system. Moreover, since each level of the hierarchy can contain only one computation model, it is impossible to model components with inputs or outputs that use different models of computation (for instance analog and digital input/output).

We propose a new approach which allows non-hierarchical heterogenous modeling by introducing Heterogenous-Interface Components. These components have inputs and outputs that communicate according to different computation models and can be used as bridges between other components, without the need for new level in the hierarchy of the system.

With this approach, the hierarchical structure of the model can match the hierarchy of the parts or of the functionalities of the system. However, the execution model is more complex since the different computation models used in the system must be coordinated. However, our preliminary implementation in Ptolemy II let us think that this coordination may be achieved with little impact on existing computation models.

## References

[1] http://www.specc.org/.

[2] http://www.superlog.org/.

[3] http://www.systemc.org/.

[4] M. Auguin. Co-conception de systèmes spécialisés sur composant. In *École thématique sur les systèmes enfouis, I3S, Université de Nice-Sophia Antipolis*, Novembre 2000.

[5] J. Buck and R. Vaidyanathan. Heterogenous modeling and simulation of embedded systems in el greco. Technical report, Synopsys Inc.

[6] J.-M. Daveau. *Spécification système et synthèse de la communication pour le co-design logiciel/matériel*. PhD thesis, Thèse de doctorat INPG, Laboratoire TIMA, Décembre 1997.

[7] J. Davis II, C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Tsay, B. Voge, and Y. Xiong. Heterogeneous concurrent modeling and design in java. Technical Report Memorandum UCB/ERL M01/12, Department of Electrical Engineering and Computer Science, University of California at Berkeley.

[8] B. Lee and E. A. Lee. Interaction of finite state machines and concurrency models. Technical report.

[9] G. Svarstad, A. Nicolescu, and A. Jerraya. A model for describing communications between aggregate objects in the specification and design of embedded systems. Technical report.