An Overall Specification of a Meta-Model of Computation For Model-Driven Embedded System Modeling

Mokhoo Mbobi
Department of Computer Science
Ecole Supérieure d'Electricité
Gif-sur-Yvette, 91192, France
Mokhoo.Mbobi@supelec.fr

Frédéric Boulanger
Department of Computer Science
Ecole Supérieure d'Electricité
Gif-sur-Yvette, 91192, France
Frederic.Boulanger@supelec.fr

ABSTRACT

The prototyping of embedded systems being long and expensive, software models are often defined to be used as a behavioral reference for better exploration and evaluation of the systems properties. However, embedded systems being naturally heterogeneous, they require the coexistence of several models of computation. In software engineering, the great challenge is how to specify heterogeneous interfaces for ensuring well defined communications because various semantic properties are mixed. Therefore, some existing approaches use only a set of few models of computation to reduce this complexity. Some others can use an open set of models of computation but reduce the complexity by forbidding to use different models of computation in the same hierarchical level.

This paper proposes a modeling approach based on the meta-modeling of models of computation. It allows to mix different models of computation by using a meta-model of computation which provides a common semantic description at a higher abstraction level. This makes easier to specify the interactions between models of computation.

KEYWORDS: Modeling, Meta-Model, Model of Computation, Model-Driven, Embedded System.

1. INTRODUCTION

In [18] [3], the authors argue that the reasons for a growing embedded-system market are durable. So, new kinds of embedded systems will appear, existing systems will change, the services around them will develop, and the number of familiar objects containing embedded processors will increase in a continuous way in the future. To master this unceasingly growing market with a shorter and shorter time-to-market, the "design-development-production" cycle of components must be shortened. Therefore, researchers and equipment suppliers are required to deal respectively with technical and economic constraints. A detailed presentation of these constraints is given in [15] [16].

This kind of system being manufactured in large quantities and the risk of a prejudicial error being to avert, the realization "*a priori*" of a prototype is unavoidable. Furthermore, this prototyping being long and expensive, software models are often used as a behavioral reference for better exploration and evaluation of the systems properties. This is why a preliminary phase of modeling is necessary. From this phase, software and hardware, communication and computation, control and data are handled separately and are joined together only in the last phases of prototyping. This preliminary phase of modeling requires a multidisciplinary scientific knowledge, from which arises the need for several specialists in different scientific and technical domains. Each team brings its competencies through specific formalisms. Since the overall behavior of the system builds on communications and interactions between all components which make it up, the pinpoint description of different interfaces between these components is crucial. The challenge is to specify what happens at the boundaries since the components may use different physical laws and, consequently, have different internal semantic properties. Unfortunately, existing modeling approaches that can use an open set of MoCs cannot integrate them at the same hierarchical level. Others can do this integration but only by using a fixed set of MoCs which are generally the continuous and discrete signal models for electrical engineering, or state machines and differential equations for hybrid systems.

This paper comes up with a modeling approach based on the meta-modeling of models of computation. It allows the integration of various models of computation by using a meta-model of computation which provides a common semantic description to all models of computation. This description provides a way of specifying any model of computation at a higher abstraction level. Furthermore, since all models of computation are described in the same specification framework, the interactions between them can be easily specified like a heterogeneous element belonging to the meta-model.

2. HETEROGENEOUS MODELING

From an abstract point of view, a model of a system can be regarded as a set of components which have properties, and between which relations exist. The various components of a model can belong to different technical domains such as analogical or numerical electronics, mechanics, thermodynamics, optics, image and signal processing algorithms, etc. Therefore, these technical domains do not consider their respective components and the relations between them in the same way. They give different meanings to the relations between components in terms of scheduling, communication or dependency. In each domain, the interactions between the components are controlled by a set of "physical laws" or "axioms" which express constraints on the components properties according to their relationship. This set is called a "Model of Computation", or MoC. Continuous Time (CT), Discrete Events (DE), Discrete Time (DT), Communicating Sequential Processes (CSP), Finite State Machines (FSM), Synchronous Data Flow (SDF), Kahn Process Networks (PN) etc. are examples of models of computation that we give some features from [13]. In the Kahn Process Network MoC, processes interact through channels that can buffer messages. This MoC is suitable for loosely coupled distributed agents and data-centric algorithms. In the Continuous Time MoC, functional and storage components communicate continuous waveforms. This MoC is used to model the physical environment of a system, analog circuits or continuous control laws. In the Discrete Events MoC, components communicate via signals that carry events placed on a continuous and global time line. It is used for digital circuits, communication networks, querying systems, and embedded software at the input/output level. The Discrete Time MoC also has a global notion of time, but it is discrete and periodical. Every signal has a value at every clock tick. It is used in periodically sampled data systems and cycle-accurate modeling. In the Finite State Machines MoC, components are states and the relations between states are interpreted as transitions triggered by events. It is used to model operating modes and control sequences. A comparative and detailed study of various models of computation is presented in[9].

The organization of a system and the interactions between its different subsystems imply a connection of the subsystems that do not use the same model of computation. Such a system that uses various models of computation is called a "Heterogeneous System".

An embedded system is naturally heterogeneous since it uses several technical domains. Consequently its design calls for several models of computation. There are two main techniques for heterogeneous modeling: amorphous heterogeneity and hierarchical heterogeneity.

In the amorphous approach [16], modeling and design environments generally focus on a fixed set of models of computation. Since they use few models of computation that are known beforehand, they can easily define the union of them. Furthermore, the complete knowledge of the interactions between these MoCs allows to compute the whole behavior of a heterogeneous model. A digital to analogical signal converter with digital inputs and analog outputs is an example of such a system. SIMULINK [17] and VHDL-AMS [7] are examples of modeling and design environments that use amorphous heterogeneity.

The advantage of this approach of heterogeneity is the complete integration of the models of computation. However, it has some weaknesses because the communication protocols interact in an unforeseeable way with the control flow because of the lack of a clear separation between the control flow and communications. Moreover, the design may be less comprehensible and the set of MoCs is fixed.

The hierarchical approach [10] [11] [8] [15] [5], allows to support an open set of models of computation, and therefore forbids to build the union of these models of computation because they are not known beforehand. This approach of heterogeneity requires that each component obey only one model of computation. Since components that are connected obey the same model of computation, all the components that are interconnected must obey the same model of computation. However, the hierarchical abstraction makes it possible to use a model of computation to model the inside of a component that is different from the outer model of computation in which the component is used. Therefore changes of models of computation can only occur at the hierarchical boundary of a component; el Greco [6], PTOLEMY II [4] are examples of modeling and design environments that use the hierarchical heterogeneity approach.

Although having several advantages, such as the possibility to cope with the complexity of systems, by abstracting a network of components that obey the same model of computation in only one component, this approach presents some disadvantages such as the coupling between the hierarchical structure of the model and the changes of models of computation. This coupling can lead to models that have a structure that do reflect the effective structure of the system.

In [14] [15], we used the actor paradigm to introduce a new modeling approach called "Non-Hierarchical Heterogeneous Modeling". This approach builds on hierarchical heterogeneous modeling to provide a way of modeling Heterogeneous Systems and has been integrated and validated by simulation in the Ptolemy II platform. It uses "Heterogeneous Interface Components (HIC)" and a "Flat Heterogeneous Execution Model".

This new approach makes it possible to have a "flat" heterogeneous model which gives the possibility of changing models of computation without changing of hierarchical level. Moreover, the designer has the possibility to specify exactly what happens when data passes from one model of computation to another and to control the behavior of a heterogeneous system at the borders of the models of computation. The weakness of this approach is that it does not allow heterogeneous loops in the connectivity graph of the components.

3. TRIHEDRAL DECOMPOSITION OF MODELS OF COMPUTATION

Embedded software is a software which runs in devices that are not necessarily computers. This type of software is very widespread in automobiles, telephones, airplanes, etc. Its key characteristic is that it interacts with the physical world from which it must obtain certain properties. For example, it must take physical time and to consume energy to carry out its behavior. In embedded system hierarchical modeling, hierarchical block diagrams support abstraction and refinement. Abstraction allows a block diagram to be compressed into a single block while refinement allows a block to be expanded into a block diagram [1]. This abstraction depends on the models of computation which make it up. For example, in [2], the authors showed that there are two abstractions for hybrid systems: continuous and discrete. Moreover, this abstraction must contain temporal exactitude, concurrency, promptness, reactivity and heterogeneity. These properties are essential so that the resulting system has the expected behavior because it is not sufficient to ensure the perfect match between arguments and results of the function of the system. Consequently, the design flow of a system must take into account the three determining elements that are time, concurrency and communication. This is what we call a *"trihedral decomposition of a model of computation"* and we symbolize as shown in figure 1.

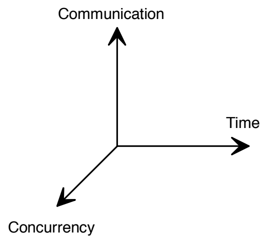

Figure 1: Trihedral abstraction of MoC

Generally, whatever the interpretation that we could have of this diagram, communication operations and concurrency will depend on time evolution.

3.1. Time

Some models of computation are very explicit and consider the time as a total order constraint, i.e., a real number which advances uniformly, and place events on this line of time or make continuous signals evolve with this time. Other models of computation consider the time as a partial order constraint imposed by causality. So, time is about the ordering of events [12] and there are two types of systems : *timed and untimed.*

A timed system is a system where the set of time T is totally ordered. That is, for any distinct t_1 and t_2 in T, either $t_1 < t_2$ or $t_2 < t_1$ Continuous Time(CT), Discrete Events (DE) and Discrete Time (DT) are examples of timed models of computation.

In an untimed system, the set of time T can be partially ordered. Communicating Sequential Processes (CSP), Kahn Process Networks (PN), Synchronous Data Flow (SDF) are examples of untimed models of computation.

Generally, time is considered as a shared value. At different abstraction levels, it must be modeled to reflect the temporal behavior of the system. It is important for the designer to clearly and accurately express the meaning of time used by the model of computation in each abstraction level, whether it is a partial or a total order constraint.

3.2. Concurrency

Concurrency or parallelism is the characteristic of a system to have actions, communications, or both, partially ordered, rather than completely ordered. Then, two systems are concurrent when they share the same set of time.

If an operation f starts at t_{f_s} and finishes at t_{f_e}, and an operation g starts and ends at t_{g_s} and t_{g_e} then

- if $t_{f_e} > t_{g_s}$ then f precedes g
- if $t_{g_e} > t_{f_s}$ then g precedes f
- in all other cases, f and g are concurrent.

The concurrency must be modeled at all abstraction levels to express the real system behavior.

Communicating Sequential Processes (CSP) is an example of a concurrent model of computation.

3.3. Communication

In order to have a maximum design flexibility of a heterogeneous system, the communication function must be separated from the whole function of the system and be modeled at different abstraction levels. This function is ensured by the communication protocol which provides a set of services needed to ensure the communication between components. This protocol depends on the model of computation, on its concurrent nature. It may be asynchronous message passing, rendezvous or synchronous message passing etc.

4. META-MODELING ARCHITECTURE

The embedded software meta-modeling architecture proposed in this paper contains three levels. The meta-model of computation level, the model of computation level, and the instance level, as shown in figure 2. Each of them contains a suitable specification of time, concurrency, and communication. In the following, we describe those specifications in the meta-model of computation level.

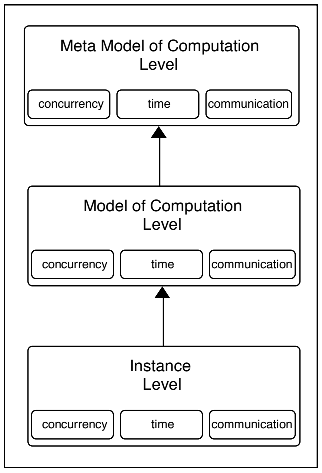

Figure 2: Meta-modeling levels

4.0.1 Meta MoC Level Specification

The meta-model of computation level specifies the model of computation in terms of time, concurrency, and communication. It provides a neutral and common description for the semantic specification of any model of computation. This description is common in the sense that it includes all determining semantic properties of all the models of computation. It is neutral because it provides only a template for models. These are templates because no mechanism is actually provided for them.

- *Time* : Any dynamic system operates in a certain notion of time, real, discrete or partially ordered. These notions are ordered by a refinement relation and the more refined time is the physical real time. This implies that it contains all others times. This is why, at this level the time specification corresponds to the physical real time: continuous. This time, should the need arise, can be refined in lower levels.

- *Concurrency :* Since a partial order relation can be transformed in a total order relation by applying a topological sort, on this level, any system is supposed to be concurrent. Likewise, in lower abstraction levels, should the need arise, this partial order relation will be easily transformed to obtain a total order relation.

- *Communication :* Since the message passing protocol can be serve as a basic type of communication protocol, in this level, each system is supposed to use the message passing protocol.

  The basic operations $read$ and $write$ which specify how to get and to send data are specified in this level. But these operations are empty because they contain no implementation.

The lower levels have to describe time, concurrency, and communication so that models of computation can interoperate. This is why on this level, a model of computation is considered concurrent, using a real physical time and a message passing protocol. This specification allows easy modeling of the semantic properties of time, concurrency and communication in a heterogeneous interaction between two or several models of computation in the sense of [19] and are represented as dotted blocks in figure 3:

1. to translate the common semantic properties between two MoCs

2. to ignore the semantic properties in the first MoC that are not present in the target MoC

3. to create the semantic properties in the target MoC that are not present in the first MoC

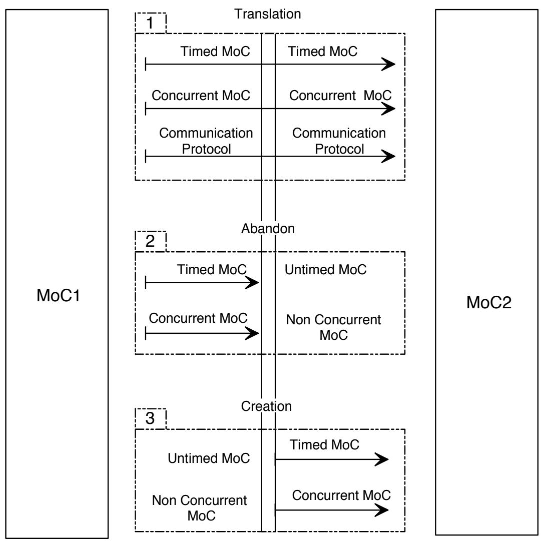

Figure 3: Heterogeneous interactions

4.0.2 MoC Level Specification

In this level, models of computation are implemented by instantiation of the meta-model of computation. The specification of each model of computation is implemented by taking into account its own semantic properties. It is here that the designer is required to implement the real form of time, concurrency and communication operations and protocol used by the model of computation. For example in the Discrete Time model of computation, it is here that the designer will implement a mechanism of time discretization.

4.0.3 Instance Level Specification

The instance level provides the particular instances of each model of computation by instantiation. It is the level where the model can be created, used, and killed.

5. TRANSFORMATION OF MOCS

The proposed architecture allows the models computation to maintain several types of relations such as the transformation relation where a model of computation M1, instance of a class C1, can be transformed in a model M2, instance of a class C2, as shown in the figure 4.

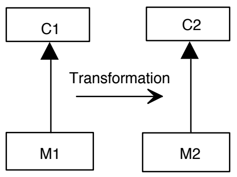

Figure 4: Transformation of MoCs

The success of this transformation is based on the transformation of the trihedral decomposition presented in 3.

- For untimed and timed systems: introduction or abandon of time in accordance with the target system.
- For concurrent or non concurrent systems: transformation of order relations.
- The basic communications protocol (message passing) can be enriched or used to build more complex protocols and conversely.

6. TECHNICAL INTEREST AND EXAMPLE

To support the heterogeneity that arises from the use of different technical domains, modeling tools nest different MoCs that characterize various technical domains. In Embedded systems, the control part sometimes changes more often than the signal processing part. Heterogeneous 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 is to hide this control inside the behavior of the data-processing operators. In figure 5, we present an example of a modem chip using four MoCs and some control in the same hierarchical level. It has digital inputs connected to the bus of a PC and analog outputs to the receiver/transmitter. The amplitude modulation part is easily modeled in a heterogeneous way using SDF, DT, DE, CT MoC. It is composed of a Generator G having a DT output, a Detector D with one SDF input and two DE outputs and a Amplifier A with two DE inputs and one CT output.

A digital signal S_d goes from the PC to the modem and obeys SDF semantics. It goes through D. In the modem, G provides a sinusoid signal S_s that obeys DT semantics and is injected in A. D computes the state (up or down) of signal S_d and sends an event S_{up} or S_{down} to A. S_{up} and S_{down} obey DE semantic and operate in mutual exclusion. A detects the presence of S_{up} or S_{down}. If S_{up} is present, it outputs a rectified signal $S_t = S_s * f_\alpha$ where $f_\alpha > 1$. If S_{down} is present, it outputs a rectified signal $S_t = S_s * f_\beta$ where $f_\beta < 1$.

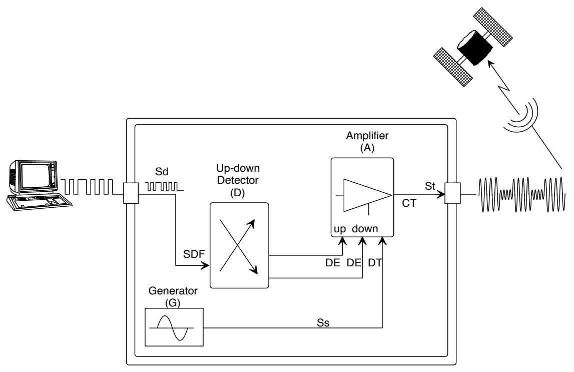

Figure 5: Example

7. CONCLUSION

Heterogeneous modeling is very useful for the modeling of many systems where software is becoming more and more important, 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 this paper, we have presented a high level model-based approach which provides the ability to explicitly specify the meaning of temporal properties, communication operations, and concurrency in several models of computation that communicate. Since all models of computation are specified using a common semantics description, heterogeneous interactions can be specified as a part of a heterogeneous system. Moreover, from the point of view of design, this approach allows the reuse of elementary models taken from a library of conceptual models of components to produce conceptual models of more complex components.

REFERENCES

[1] L. de Alfaro and T. A. Henzinger, *"Interface Theories for Component-based Design"*, in the Proceedings of the First International Workshop on Embedded Software, EMSOFT, 2001.

[2] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky, *"Hierarchical Hybrid Modeling of Embedded Systems"*, University of Pennsylvania, Availlable on line at http://www.seas.upenn.edu/hybrid/

[3] M. Auguin, *"Systèmes sur Puce : vers lexploration en milieu complexe"*, Ecole Thématique sur les Systèmes Enfouis, I3S, Université de Nice Sophia Antipolis, CNRS, INRIA, novembre 2003

[4] S.S. Bhattacharyya et al, *"heterogeneous concurrent Modeling and design in java, Volume I to III"*, Memorandum UCB/ERL M05/21 eecs, University of California at Berkeley, July, 2005

[5] F. Boulanger, M. Mbobi and M. Feredj, *"Flat Heterogeneous Modeling"*, Proceedings of the conference of Internet Processing Systems Interdisciplinaries, Venice, Italy, November 10 to 15, 2004

[6] J. Buck and R. Vaidyanathan. *"Heterogenous modeling and simulation of embedded systems in el Greco"*, Proceedings of the 8th international workshop on Hardware/software codesign, San Diego, California, USA, Pages: 142-146, 2000, ISBN:1-58113-268-9.

[7] E. Christen, K. Bakalar, E. Moser, *"Analog and Mixed-Signal Modeling using the VHDL-AMS Language"*, 36th Design Automation Conference, New Orleans, June 1999.

[8] A. Girault, B. Lee, and E. A. Lee, Fellow, IEEE, *"Hierarchical Finite State Machines with Multiple Concurrency Models"*, Proceedings of the DATE99 conference, pp.382-383, March99.

[9] E.A. Lee and A. Sangiovanni-Vincentelli, *"A Framework for Comparing Models of Computation"*, IEEE Transactions on computer-aided design of integrated circuits and systems, Vol. 17, no. 12, December 1998.

[10] B. Lee and E.A. Lee, *"Hierarchical Concurrent Finite State Machines in Ptolemy"*, University of California at Berkeley, Proceeding of International Conference on Application of Concurrency to System Design, p. 34-40, Fukushima, Japan, March 1998.

[11] B. Lee and E. A. Lee, *"Interaction of Finite State Machines and Concurrency Models"*, University of California at Berkeley, Proceeding of Thirty Second Annual Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, California, November 1998.

[12] B. Lee and S. Neuendorffer, *"Concurrent Models of Computation for Embedded Software"*, University of California at Berkeley, Proceeding IEEE, Computers and Digital Techniques, November 2004.

[13] J. Liu, *"Responsible Frameworks for Heterogeneous Modeling and Design of Embedded Systems"*, Ph.D. thesis, Technical Memorandum UCB/ERL M01/41 Electronics Research Laboratory, University of California, Berkeley, December 20th, 2001.

[14] M. Mbobi, F. Boulanger and M. Feredj, *"Execution Model for Non-Hierarchical Heterogeneous Modeling"*, Proceedings of The 2004 IEEE International Conference on Information Reuse and Integration, November 8-10, 2004, Las Vegas, USA, IEEE, ISBN 2004113902, pages 139 144

[15] M. Mbobi, *"Modélisation Hétérogène Non-Hiérarchique"*, Ph.D. Thesis, Supélec, Université Paris XI (Orsay), Décembre, 2004.

[16] M. Mbobi, F. Boulanger and M. Feredj, *"Issues of Hierarchical Heterogeneous Modeling in Component Reusability"*, Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration, 15-17 August, 2005, Las Vegas, Nevada, USA, IEEE Catalog Number 05EX1058, ISBN 0-7803-9093-8, pages 84 to 89

[17] M. Mokhtari and M. Marie. *"Engineering Applications of MATLAB5.3 and SIMULINK 3"*, Springer Verlag, 2000.

[18] RNTL, Rapport du groupe de travail "système embarqué et temps réel, co-développement", 2001, Availlale on line at http://www.telecom.gouv.fr/rtnl

[19] W-T. Chang, S. Ha, and E.A. Lee, *"Heterogenous simulation - mixing discrete-event models with dataflow"*, Journal of VLSI Signal Processing 15, 127-144, Kluwer Academic Publishers, 1997.