In *Proceedings of the 1998 EUROSIM Conference*, Helsinki, 1998, pp. 269-276.

SEMANTICS OF MODEL COMPOSITION IN HYBRID LANGUAGES

G. Fábián[1], D.A. van Beek[2], J.E. Rooda[3]

Eindhoven University of Technology, P.O.Box, 513, 5600 MB Eindhoven, {g.fabian[1], vanbeek[2], rooda[3]}@wtb.tue.nl

Abstract

The increasing complexity of hybrid models requires a mathematically sound language semantics that allows reasoning about model behaviour and supports development of robust models. Model composition in hybrid languages is realised by connecting continuous variables of sub-models. It is important to define the precise semantics of connections. Improperly defined connection semantics can *a)* introduce undesirable non-determinism and *b)* lead to undesirable semantics of conditional language constructs. In this article four different connection semantics are examined. Three of them impose strong restrictions on the modelling style or lead to less robust models. In the hybrid χ language the fourth possibility is chosen: a connection represents an algebraic equation. In combination with the initial state calculation mechanism, and the use of local variables, this results in robust and elegant composition semantics.

1. Introduction

Most hybrid languages support hierarchical models. Since many physical systems are composed of smaller components, models become more clear if they reflect the structure of the modelled system. Hierarchical modeling facilitates the process of model development in many ways. It supports both top-down and bottom-up development. Components can be tested individually and their behaviour can be studied in different environments, which promotes the development of libraries and standardization in different application areas [1].
Model composition in hybrid languages is realized by connecting (among others) continuous variables of sub-models. For this purpose, connection elements, so-called *streams* (gPROMS [2], ASCEND [3,4]), *terminals* (Dymola [5,6], Omola [7]), or *channels* (χ [8,9]) are introduced in the interface specification of the sub-models. The variables are connected by means of these elements. In most languages, a connection imposes an equality constraint on the variables. However, this is not the only meaning that can be associated with a connection. Depending on the physical quantity that the variables represent, continuous variables can be divided into two classes: *across* variables, that become equal, and *through* variables, that sum to zero, when connected [10]. In this article we concentrate on the semantics of connections that impose equality. However, the requirements of a connection semantics equally hold for connections of through variables. The only difference is that in that case, the connection equation is a summation to zero.
Specification of discontinuities in hybrid models is an important facility, provided by hybrid languages. One way to express discontinuities is by changing the values of the continuous variables, including the connected ones. Therefore, the language semantics has to define, what the values of the connected variables are after such a change, and how equality is (re-)established. When equality is (re-)established, the value of the variables of other sub-models may change as well. Particularly when sub-models are executed concurrently, the way equality is ensured among connected variables can alter the behaviour of the entire model. Therefore, it is essential to properly define the semantics of connections when one or both of the connected variables are changed discontinuously. In Section 2, situations are examined where model behaviour depends on the semantics of connections.
In Section 3, possible semantics of connections is treated. The different choices made by developers of hybrid languages are compared, and major advantages and disadvantages are given. In Section 4, the semantics of connections chosen in the χ language is discussed in more detail.

2. Discontinuous changes of connected variables

Hybrid languages provide discontinuity specification facilities in different forms. These facilities are used for several purposes. A discontinuity in the model can be used to describe an instantaneous action, the time scale

of which is much smaller than that of primary interest. Also, external actions that change the system state can be described in this way. An example could be the interaction of the plant operator, who switches a valve on or off. Discontinuities can also be used to model actions that are themselves of no interest; only the results of these actions are important. A possible example is the process of filling a tank with reactants in preparation for a batch process.
One way of introducing a discontinuity is to change the value of a continuous variable. It can be changed explicitly by an assignment. Another, more subtle way to change a continuous variable is by altering the equations or other variables, which determine its value. After such a change, the model has to be transformed into a consistent state. A model is in a consistent state if *a)* the variables satisfy the equations and *b)* the connected variables are equal. In Section 3 we show, that some connection semantics ensure that the second requirement is automatically satisfied in any state of the system (e.g. shared variables). In other connection semantics the second requirement is the special case of the first one (algebraic variables). When equality is re-established among the connected variables, the variables of other sub-models may change as well, which influences the behaviour of these sub-models. In Sections 2.1 and 2.2 we examine, how the behaviour of the connected sub-models depends on the way equality is ensured among connected variables.

In the examples, the χ syntax is used. The elementary building blocks of a χ model are *processes.* Processes are executed concurrently. A χ process consists of several optional sections, and is declared in the following way:

```
proc name (parameter declarations)=
|[ variable declarations ; initialization
| links
| DAEs
| discrete-event statements
]|
```

The process interface declares parameters, including discrete and continuous channels. A continuous channel is denoted by the $\multimap$ symbol. In the declaration section, the continuous variables are declared with the :: symbol, whereas discrete variables are declared with the : symbol. In the declaration of continuous types and continuous channels, the units of measurement are given between square brackets. The null dimension is denoted by [-]. In the link section the continuous variables are linked to channels. For this, the $\multimap$ symbol is used as well. Variables of different processes are connected if they are linked to the same channel. Equations are given in a symbolic, analytical form, where x' denotes the first time derivative of variable x. For assignments to continuous and discrete variables the ::= and the := symbols are used, respectively.

2.1 Scheduling dependency

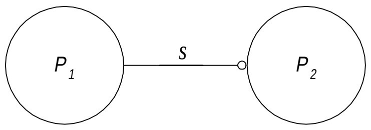

Figure 1. System of two processes.

This example illustrates how the value of a connected variable may depend on process scheduling. Consider the system consisting of two processes depicted in Figure 1. The two processes P_1 and P_2 are connected by channel s. This is specified in system S. *Systems* are used in χ to describe the fixed connection layout of the contained processes and subsystems.

$$\text{syst } S = |[\ s :: [-]\ |\ P_1(s) \| P_2(s)\ ]|$$

The local variables s_1 in process P_1 and s_2 in process P_2 (see below) are connected to channel s therefore, they are equal. Process P_1 determines the value of s_1. The value of s_1 changes periodically with a period of 4 seconds. In the first 3 seconds of each period s_1 is 0, and in the remaining 1 second it is 1. Process P_1 is as follows:

```
proc $P_1(s :: \multimap [-]) =$
|[ $s_1 :: [-]$; $s_1 ::= 0$
| $s \multimap s_1$
| $s_1' = 0$
| $*[\Delta 3;\ s_1 ::= 1;\ \Delta 1;\ s_1 ::= 0]$
]|
```

In the discrete-event part a repetition construct is used: statements that are enclosed between *[and] are repeated forever. The Δ symbol denotes time passing. Processes executing a statement Δc, where c is a numerical expression, remain blocked for c seconds.
Every second process P_2 stores the value of s_2 in a list. Lists are declared in χ by appending the * symbol to a type. The empty list is denoted by [], and ++ is the append operator.

```
proc $P_2(s :: \multimap [-]) =$
|[ $s_2 :: [-]$, $xs : \mathrm{real}*$
| $s \multimap s_2$
| $xs := [\,]$
; $*[\Delta 1;\ xs := xs \mathbin{++} [s_2]]$
]|
```

The value of s_2 after 3 seconds, when P_2 executes the statement $xs := xs \mathbin{++} [s_2]$, is determined by the connection semantics. If P_1 is executed before P_2, s_1 becomes 1. Depending on whether the value of s_2 is changed immediately or not, the value of s_2 in the assignment of xs is 0 or 1 respectively. One might find the desired output or even the correctness of this model questionable, since the value of s_2 is stored exactly at the time instant when s_1 changes in P_1. However, situations like this can not be prevented in complex models. The reason for this is that it is very difficult to oversee whether events in different sub-models happen at the same time. In fact, due to accumulating rounding errors even those events, which in principle should happen at distinct but very close time instants, may happen at the same time.

2.2 Conditional language constructs

In hybrid languages conditional constructs are used for several purposes. In many hybrid systems the describing set of equations depends on the state of the system. To model this, some form of a conditional equation is used. Such an equation specifies different sets of equations to be active, depending on the state of the system. In state-event specifications conditions are also used. The semantics of the connections affects the semantics of other language constructs as well. Suppose, for example, that in process P_2 the values of s_2 are sorted into two lists, according to whether they are smaller or equal, or bigger than 0.5.

```
proc $P_2(s :: \multimap [-]) =$
|[ $s_2 :: [-]$, $n : \mathrm{int}$, $xl, xh : \mathrm{real}*$
| $s \multimap s_2$
| $n := 0$; $xl := [\,]$; $xh := [\,]$
; $*[\Delta 1;$ [ $s_2 \le 0.5 \;\rightarrow\; n := n + 1;\ xl := xl \mathbin{++} [s_2]$
              [] $s_2 > 0.5 \;\rightarrow\; n := n + 1;\ xh := xh \mathbin{++} [s_2]$
              ]
   ]
]|
```

For selection, the guarded command of the form $[\, b_1 \rightarrow S_1 \,[]\; b_2 \rightarrow S_2 \,[]\ldots[]\, b_n \rightarrow S_n \,]$ is used. The boolean expression $b_i \; (1 \le i \le n)$ denotes a guard, which is open if the expression evaluates to true, and is otherwise closed. After evaluation of all the guards, one of the alternatives with an open guard is chosen for execution. In this example, a choice is made based on the value of s_2, therefore, execution of P_2 depends on the connection semantics. If after 3 seconds P_1 is scheduled first, and the assignment to s_1 results in the immediate change of s_2, then the second alternative ($s_2 > 0.5$) is chosen, otherwise the first one. Furthermore, if the concurrent assignment of s_1 ($s_1 ::= 1$) in P_1 changes the value of s_2 between the evaluation of the guards and the append action, then the first alternative ($s_2 \le 0.5$) is chosen in P_2, yet the value 1 is appended to list xl. Obviously, this would not be the desired behaviour of P_2. Normally, list xl should only contain values smaller or equal than 0.5, while values bigger than 0.5 should be stored in xh.

The example illustrates that when conditional language constructs use a connected continuous variable two problems may arise depending on the connection semantics. One problem is that the choice may depend on process scheduling. For the purpose of modelling industrial systems it is desirable that operations on a connected variable do not result in non-deterministic behaviour. The second problem is that after an alternative with a true condition is selected, the condition might immediately become false. This means that the condition can not be used as post-condition, when reasoning about the model behaviour.

3. Semantics of connections

In this section different connection semantics are surveyed. The purpose of this survey is to present different possibilities and to examine how different semantics determine, and in most cases restrict, the modelling style. Wherever possible, we mention some simulation languages that employ the given approach. However, it is not our aim to detail the consequences of choosing certain semantics in a given language. In order to do this, a language considered would need to be studied in full detail, and special language characteristics would need to be revealed. In many cases, however, there is not enough documentation available.
From the available documentation it appears that often the nature of the language guarantees that situations shown in the previous section do not occur. Furthermore, the way certain language constructs can be used and combined may be restricted in order to decrease the chance for erroneous or ambiguous models. In gPROMS, for example, two sub-models cannot manipulate the same connection equation simultaneously.

3.1 Shared variables

Connected variables become a single entity shared by the sub-models. It implies that changes to a variable in one sub-model are immediately visible in the other sub-model. This approach is implemented in ASCEND, if streams are connected with the aid of the ARE_THE_SAME operator. Streams can be merged in gPROMS as well, if the model's interface with its environment corresponds exactly to the interface of its sub-model. The disadvantage of this approach is that in parallel languages the simulation result may depend on the actual process scheduling. Furthermore, as the second example of the previous section illustrates, the semantics of the conditional language constructs may be affected as well. Solutions used in parallel programming languages to control access to shared variables (semaphores, access protocols) would eliminate the simplicity gained by merging the variables, and probably would be an unacceptable burden in most models of industrial systems.

3.2 Causal connections

Connections define a causal interaction between the subsystems, where in each connection one side acts as input and the other side as output. One problem with this approach is that the direction of the information flow is fixed for each component. It restricts re-use of a component in other environments. Another restriction is that the connections should not lead to algebraic loops. In order to satisfy this requirement, the model may need to be transformed.
General purpose simulation packages like ACSL [11] and SIMULINK [12] use this approach. These

languages require the Differential Algebraic Equations (DAEs) of each subsystem to be written in explicit state-space form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, $\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u})$, where $\mathbf{u}$ is used as input and $\mathbf{y}$ as output. Transformation into explicit state-space form (if possible) requires extra engineering skills and the resulting equations often do not reflect the natural system structure. With the advent of DAE solvers that can solve the implicit state space form $\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, t) = \mathbf{0}$, hybrid languages do not need to employ causal relations any more [13,14].

3.3 Master object

From the model's topology, first the sets of connected variables that are equal, are identified. For each set, a master object is introduced, which holds the value of the connection. This value is distributed among the variables at the end of each continuous phase. During the discrete-event phase, the sub-models have access to their (local) variables only. Changes of these variables may lead to a temporal inconsistency in the system. The system is brought back to consistency, by updating the master object, and distributing its new value among the variables. This idea is inspired by distributed systems, where several processes share a common memory. In order to allow concurrent accesses to the same data, duplicate copies are made of the shared block (i.e., block replication). For data coherence control, one copy is designated as a master copy, the value of which is distributed among the other copies.
The difficulty in hybrid systems is to find a suitable protocol that determines the value of the master object. The mechanism has to be simple and intuitive, such that developers of complex models can oversee the system behaviour. Also, the protocol may have to comply with the way non-connected variables are assigned. See the difference between assignments to differential and algebraic variables in Section 4. A possible choice could be to assign the master object each time one of the connected variables is assigned. In this way, the master object always holds the last change that has been made to the connection. Its value could be distributed only at the end of the discrete event. However, the value of the master object would depend on the process scheduling.
Another difficulty with using a master object is that it can hardly be generalized for connection of through variables.

3.4 Algebraic equations

Connections define an additional algebraic equation between the connected variables. This is the case in gPROMS, Dymola and in χ. However, the question remains what the semantics is of such an equation, i.e., when is it valid and how is equality ensured after a discontinuous change to one or both of the connected variables. Also, it is not always clear whether the semantics of such an equation is the same as that of other equations. Furthermore, if such an equation is replaced at a lower level by one shared entity due to optimization, the first alternative, (a shared variable) is implemented. Such a replacement is only allowed if the semantics of a connection equation between two variables is identical to that of referencing a single shared entity by both variables.

4. Semantics of model composition in the χ language

The χ language is a concurrent language. In order to avoid the disadvantages of using shared variables or a master object in parallel languages, algebraic equations have been chosen as the semantics of connections. In the sequel, the precise meaning of such a connection and the consequences of this semantics are explained.
The χ language is a hybrid specification language, that has been designed to be suitable for modelling discrete-event, continuous-time and hybrid systems. Simulation of a hybrid χ model can be described as a sequence of continuous phases alternated with discrete phases at certain time points. The continuous-time behaviour is given in the form of DAEs, while discrete actions are specified in a CSP-like [15] concurrent programming language. Variables representing the system dynamics are continuous variables; those that occur with primes in the DAEs are differential variables, whereas other continuous variables are algebraic. The values of continuous variables are calculated from the DAEs; assignments determine their value only for the current time point. The values of discrete variables are determined by assignments. They hold their value between two subsequent assignments. In a χ model all variables are local to their processes. This complies with the language philosophy: data exchange among processes in discrete sub-phases takes place only by

communication through discrete-channels. Furthermore, using only local variables turns out to be essential for a simple and robust connection semantics.

4.1 Connections

A connection between two continuous variables of different χ processes implies an additional algebraic equation. To keep the language small and robust, the semantics of a connection equation has been chosen to be identical to that of any other equations defined in the processes. The semantics is as follows. Equations are valid from the beginning until the end of a continuous-phase. The discrete-phases are divided into sub-phases. Each sub-phase ends when the discrete bodies of all the processes are blocked, waiting for events to happen. During discrete sub-phases the equation consistency may temporarily be lost due to discontinuous changes of continuous variables, or due to assignments to discrete variables appearing in the equations. At the end of each sub-phase a new consistent initial state is calculated, so that equality between connected variables is re-established.

4.2 Initial state calculation

The initial state is calculated by solving the system for the algebraic variables and the time derivatives of the differential variables. The differential variables are taken as known. Steady state calculation is requested, if time derivatives of the differential variables are assigned. In this special case, the value of the differential variable, the derivative of which is assigned, is calculated and its time derivative is taken as known. The language syntax supports new state calculation by allowing a special assignment symbol $::=$ to be used only for assignments to differential variables (e.g. $v ::= 0$) and, in case of steady-state calculation, for assignments to time derivatives of differential variables (e.g. $v' ::= 1$). Algebraic variables can only be given an "initial guess". For this purpose, the :~ symbol is used as in $a :\sim 10$. It is necessary to give algebraic variables an initial guess if more solutions are possible, or if the initial state solver requires a good starting value. The initial guess is used only to ensure that the solver converges to the required solution. Consider for example the equations describing a resistor and a capacitor connected in series with a voltage source.

$$\begin{array}{ccc} u_R & = & R \cdot i \\ C \cdot u_C' & = & i \\ u_0 & = & u_R + u_C \end{array}$$

The only differential variable is u_C, therefore its value can be freely chosen, e.g. $u_C ::= 10$. Algebraic variables (u_R, i) can take an initial guess, for example, $u_R :\sim 5$, $i :\sim 0.5$. As a result, the values of u_R and i change temporarily. However, when a consistent state is calculated, their values are determined by the equations.

4.3 Consequences

An advantage of treating connection equations in the same way as other equations becomes apparent in the situation where a differential variable is connected to an algebraic one. The value of the differential variable can only be changed by an assignment in the process where it appears differentiated in the equations. Consider, for example, the system of a mass m, connected by a linear spring with stiffness k as shown in Figure 2.

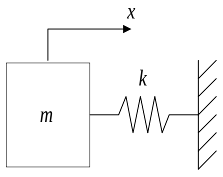

Figure 2. Mass-Spring system.

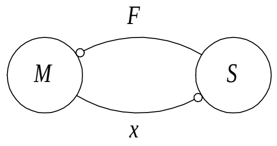

Figure 3. Mass-Spring model.

The χ model which describes this system consists of two processes: M and S, representing the mass and the spring. The processes are connected by channels F and x, which represent the force applied by the spring and the position of the mass, respectively. In process M the Newtonian equations describe the movements of the mass.

```
proc $M(F::\multimap[N], x::\multimap[\mathrm{m}], m:\mathrm{real}) =$
|[ $F_1::[N],\ x_1::[m],\ v::[m/s];\ x_1::=0.3$
| $F \multimap F_1,\ x \multimap x_1$
| $F_1 = m \cdot v'$
, $v = x_1'$
]|

proc $S(F::\multimap[N], x::\multimap[\mathrm{m}], k:\mathrm{real}) =$
|[ $F_2::[N],\ x_2::[m]$
| $F \multimap F_2,\ x \multimap x_2$
| $F_2 = k \cdot x_2$
]|

syst $MS =$
|[ $F::[N],\ x::[m]$
| $M(F, x, 0.1)\ ||\ S(F, x, 0.05)$
]|
```

In this model x_1 is a differential variable that is connected to an algebraic variable: x_2. This means that the connection equation $x_1 = x_2$ is added to the set of active equations. The value of x_1 can only be changed in process M, for example, by means of an assignment. Since x_2 is algebraic variable, it can only be given an initial guess. However, this does not have any effect on the value of x_1 because, when the initial state is calculated, x_1 is constant and x_2 is determined by the connection equation. Allowing only one process to change a differential variable discontinuously promotes more disciplined modelling.

Other situations, where the value of the connection should be changed in both processes are less common. When two algebraic variables are connected, like in the above example F_1 and F_2, the values of both of them are calculated from the equations. Therefore, it is unlikely that their value should be changed explicitly. The language reflects this: both can only be given an initial guess. Finally, when two differential variables are connected, one becomes dependent. This is the case, for example in multibody systems, where rigid bodies are connected to each other by joints that restrict the relative motion between pairs of bodies. The variables that represent the position of the bodies are differential, since they appear differentiated in the equations that describe the system motion. However, the position of a connected body can not be freely chosen; it is (partially) determined by the position of the body, to which it is connected. In general, connection of differential variables means, that one reduces the degree of freedom of choosing the values of differential variables. Furthermore, the index of the DAEs is increased, which leads to numerical difficulties.

Another advantage of the connection semantics chosen in the χ language is that discontinuous changes to connected variables do not alter the behaviour of connected processes within a discrete sub-phase. This is because all variables are local to the processes and equality is established only at the end of the sub-phase. This means that process behaviour in a discrete sub-phase can be fully analyzed based on the values of local variables, independently of whether a variable is connected or not.

5. Conclusions

The possibility of discontinuous changes of connected variables requires the semantics of connections to be properly defined. Improperly defined connection semantics can *a)* introduce undesirable non-determinism and *b)* lead to undesirable semantics of conditional language constructs.
The semantics of model composition can be defined in several ways. However, use of shared variables may lead to less robust models, whereas use of causal connections imposes restrictions on the modelling style.
By choosing the semantics of connections to be identical to that of other equations, the hybrid χ language is kept small and robust. In combination with keeping all variables local, this results in simple and elegant model composition semantics. Other beneficial consequences of this combination are that *a)* a differential variable can be changed in one process only and *b)* model behaviour in a discrete-event depends on the values of local variables only.

References

[1] S.E. Mattson. “On model structuring concepts.” In *Preprints of the 4th IFAC Symposium on Computer-Aided Design in Control Systems*, pp. 209-214, Beijing, 1988.

[2] P.I. Barton. *The modelling and simulation of combined discrete/continuous processes*, Ph.D. Thesis, University of London, 1992.

[3] *ASCEND IV Manual*, Carnegie Melon University, 1997.

[4] P. Piela, T. Epperly, K. Westerberg, and A. Westerberg. “ASCEND: An object-oriented computer environment for modeling and analysis: the modeling language.” In *Computers and Chemical Engineering,*15(1), pp. 53-72, 1991.

[5] H. Elmqvist, D. Brück, and M. Otter. *Dymola – User’s manual*. Dynasim AB, Lund, Sweden, 1996.

[6] H. Elmqvist, F. Cellier, and M. Otter. “Object-oriented modeling of hybrid systems.” In *Proceedings of European Simulation Symposium*, ESS’93, The Society of Computer Simulation, October, 1993.

[7] S.E. Mattson, M. Andersson, and. K.J. Åström. “Object-Oriented modelling and simulation.” In Linkens Ed., *CAD for Control Systems*, chapter 2, pp. 31-69. Marcel Dekker Inc, New York, 1993.

[8] J.M. van de Mortel-Fronczak, J.E. Rooda, and N.J.M. van den Nieuwelaar. “Specification of a flexible manufacturing system using concurrent programming.” In *Concurrent Engineering: Research and Applications,* 3(3) pp. 187-194, 1995.

[9] D.A. van Beek, S.H.F. Gordijn, and J.E. Rooda. “Integrating continuous-time and discrete-event concepts in modelling and simulation of manufacturing machines. ”In *Simulation Practice and Theory,* 5, pp. 653-669, 1997.

[10] H.E. Koenig, Y. Tokad, and H. Kesavan. *Analysis of Discrete Systems*. McGraw-Hill, 1967.

[11] E.E.L. Mitchell, and J.S. Gauthier. “Advanced continuous simulation Language(ACSL).” In *Simulation*, Vol. 26, No. 3, pp. 72-78, 1976.

[12] Simulink User's Guide, The MathWorks, Inc., 1993.

[13] H. Elmqvist, and S.E. Mattson. “MODELICA - The next generation modeling language - an international design effort.” In *Proceedings of the 1st World Congress on System Simulation , WCSS’97*, Singapore, 1997.

[14] F.E. Cellier, and H. Elmqvist. “Automated Formula Manipulation in Object-Oriented Continuous System Modeling.” In *IEEE Control Systems,no. 13(2)*, pp. 28-38, 1993.

[15] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood-Cliffs, 1985.