REPRODUCING THERMAL COUPLING BETWEEN COMPONENTS IN A GENERIC ENVIRONMENT LIKE MATLAB

Gilles LEFEBVRE & Elena PALOMO & Maria del Mar IZQUIERDO

GISE/ENPC - 6 av. Blaise Pascal - Cité Descartes - 77455 Marne la Vallée Cedex 2 France

(lefebvre@enpc.fr http://www.enpc.fr/gise)

ABSTRACT

We show here the actual state of a project based on an object-oriented philosophy, MotorLab. It is an interface to general and widely available interactive modeling environments (Matlab[2] and ΨlabΨ[3][1]), and relies on the use of the description structure during the whole modeling process. Furthermore, we tried from the beginning to base the modeling process in MotorLab on a technological description without need, when possible, to ask the user for lower-level indications. The presented prototype is running, but many works remains to be done to reach the fixed objectives.

INTRODUCTION

Many research teams have dedicated their effort for several years to the development of methods and tools which should contribute to increase the pertinence of models, and at the same time, should help the physicist, the architect, the engineer, to increase the modeling process efficiency [4, 5, 6, 7, 8, etc.]. It seems that for several years now, a large part of these teams have agreed on that it is better to split the modeling process in different stages which are chained together, every stage being designed in order to perform a specific task. This is particularly the case for the stage during which the thermal system is described, and the one during which the behavior is evaluated through dynamic simulations. This has been the result of a reaction against the monolithic simulation environments which have been developed during the seventies and the eighties, and which were difficult to maintain and to adapt to new problems or new methods. Splitting the modeling environments in small specialized pieces allows to connect dedicated model generators to different general or specific solvers without to rewrite the entire codes, and then permits a capitalization of the knowledge included in these environments.

But the initial structuration effort made by the user is lost, at least temporally, during the simulation, due to the fact that the solvers may rely only on numerical efficient data organization.

As other teams, we began several years ago to explore another way based on using more pertinent (from the physical point of view) algorithms and calculation methods, in order to be able to give more direct meaning to the elements and the events which may occur during the calculations. TRNSYS [18] which can be seen as the oldest and first object-based solver, the FET [10] based environment ZOOM, which is extensively used now in an other field than building modeling (climatic prediction) partly inspired the initiation of this work.

In this paper, we present a tool which is under development and is based on two main principles. The first one is that the systemic structure which is put in evidence when describing a system can be used at the time calculations are performed in order to simulate, to analyze, to reduce, etc. The research laboratory which provides modeling environments have to try to express the information related with a system under an organization which is as closed as possible to the structure which has been put in evidence by the user, and not try to impose the one coming from mathematical or numerical constraints. This first characteristic is shared with many other object-oriented environments. The second one is that the causality which must be introduce in order to calculate has generally no physical correspondence, and is not the expression of a physical causality. The physical relations between components of a system should have to be expressed without having to introduce explicitely any causality (the calculation method may require the expression of a numerical causality, but it is another problem, and both ones should not be mixed), and when possible using a technological/physical macro language. The SPARK solver [5] includes this facility at the mathematical level in which equations can be expressed as implicit (from the mathematical point of view) relations between variables.

The MotorLab prototype is, among other objectives, a practical solution to these two

[1] XLAB will be used in this paper to indicate that all related informations refer equally to both MatLab and ΨLab environments

problems. It is the actual state of a research axis which began several years ago and which led to the Motor environment, written in ADA, which has been presented for the first time at the BP'91 conference [11] and was presented as a PhD thesis by R. EBERT [18]. MotorLab reuses the experience acquired when developing Motor. The main principles are similar ones, but the practical implementation is completely different, and the objectives are much more ambitious. The present prototype, whose purposes are to demonstrate the feasibility of this approach and to be easily accessible and modifiable, is developed in Ψlab [3] and Matlab [2] which are general and very well known environments ; due to the numerous similarities between both environments, we will refer, in the following, to the "Xlab" environment.

PRINCIPLES

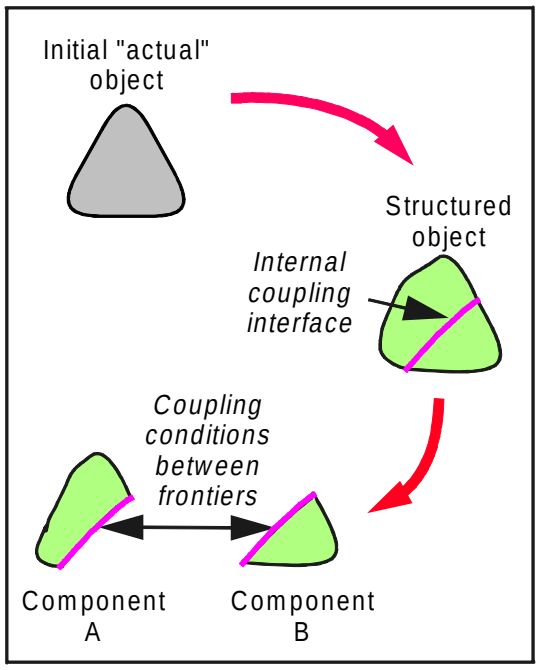

Figure 1 : An object seen as a system

The physical phenomena in a thermal system are assumed to be described by phenomenological equations, which are generally algebraic and/or differential equations (versus time and space). A well-posed problem must then be built by choosing a coherent and consistent set of equations which determines a unique solution in terms of state variable evolution versus time. A common method consists in translating directly the phenomenological equations into a set of calculation methods which are deduced from local or space distributed balance equations and boundary conditions; for example, this is the way which is used when the entire solution of a thermal problem is searched using a finite element methods on a meshing of the whole system.

But another method consists in expressing the initial complex system as a set of coupled elements. Every element is an identifiable part of the initial complex system, which can be modeled independently from the remaining part of the (global) system [12]. Every element state is determined by the phenomenological equations which are concerned with the particularities and characteristics of the element. In order for the set of elements to represent the whole system, the physical coupling which exists between these parts in the actual system must be translated in coupling relations which link the elements together. In order to have a clear image of what is a system, it is useful to imagine that the coupling relations are localized on interfaces, locations where are defined relations between frontiers of different elements. The observed system is then viewed as a set of two types of entities ; the elements and the interfaces.

Interfaces are mathematical artifacts that help to express physical relations that have been temporally broken when building the systemic model ; they should not contain any capacitive points. This lead to the fact that only component behavior can be represented by differential equations, when interfaces can only be represented by algebraic ones.

MATHEMATICAL EXPRESSION

Despite that this methodology can be applied to complex systems, we will focus this paper only on simple ones.

Let us consider a conductive wall made of two homogeneous layers in which occur heat conduction, only following the dimension perpendicular to the largest dimensions, noted x .

The heat balance equation for $M \in D$ is:

$$\frac{\partial T}{\partial t}(M,t) = \nabla K(M,T)\nabla T(M,t) + f(M,t) \quad \textbf{(1)}$$

The behavior of the wall is known when the solution of the above equation in term of the whole temperature field $T(M,t)$ is known.

Boundary conditions (points L and R on Figure 2) must be imposed to the system in order 1) to isolate the *actual* object, 2) to allow the mathematical model to have a unique solution.

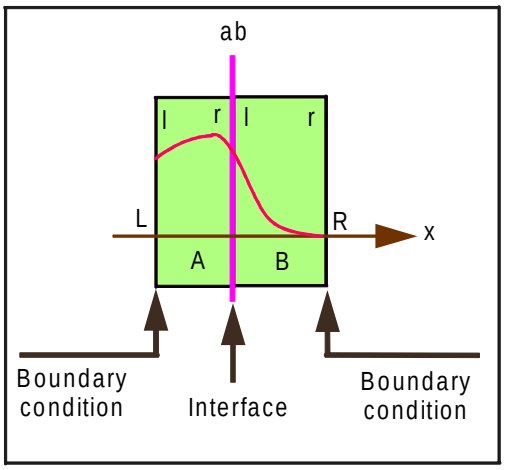

Figure 2 : A system example : a two-layers wall

If we apply the heat equation to a point located on the surface between the two layers (noted *ab* in Figure 2), the left term vanishes (as the surface is a set of non capacitive points) and the equation translates the compatibility of the heat fluxes on both sides of the surface. Another equation comes from that the field temperature must be continuous versus *x*.

$$\Phi_r^A + \Phi_l^B = 0$$
$$T_r^A = T_l^B \qquad \textbf{(2)}$$

The behavior of the two-layers wall excited by some boundary conditions applied on its frontiers L and G is described by a set of equations that determine the evolution of $T(M,t)$ versus M and t. The more usual way to find an exact or a numerical approximate of $T(M,t)$ is to *solve* directly the whole set of equations. But it is also possible to express $T(M,t)$ as a solution of a *local* phenomenological equation defined in each layer A and B (in this case this is already the heat equation **(1)**) respectively excited by boundary conditions on both sides (A.l and A.r for A, B.l and B.r for B)[2], WHEN the imposed variables that appear in the boundary conditions on the frontier respect the compatibility and continuity equations **(2).**

We consider thermal systems which physical state variables are only temperatures. The boundary conditions applied on an object are then an imposed temperature or flux. In every defined point of the object domain, it is possible to impose a temperature (intensive entity) or a flux (extensive entity), but not both. If one is imposed, the other one is determined by the physic, and can then been calculated by a model. An object model i can be formally written as follows :

$$f_i(\eta_i, T_i, \Phi_i) = 0 \qquad \textbf{(3)}$$

in which η_i is the state variable of the object i, T_i is the set of temperatures imposed or calculated on the frontiers of i, Φ_i is the set of fluxes temperatures imposed or calculated on the frontiers of i. If the object i is a capacitive one, f_i is an algebraic-differential operator, an algebraic one at the contrary. If we consider two elementary objects A and B coupled on an interface *ab* where their respective frontier *r* and *l* are submitted to a perfect thermal contact, the temperatures and fluxes of A and B will have to verify equations as **(2)** that we will write more generally :

$$g_{ab}(T_A^{ab}, T_B^{ab}, \Phi_A^{ab}, \Phi_B^{ab}) = 0 \qquad \textbf{(4)}$$

where α_β^v indicates the α variable of the component β restricted to the frontier *v*. Finally, a system is modeled under a hierarchical way, by a tree in which leaves are elementary component, and ramifications are coupling interfaces, as shown in Figure 3.

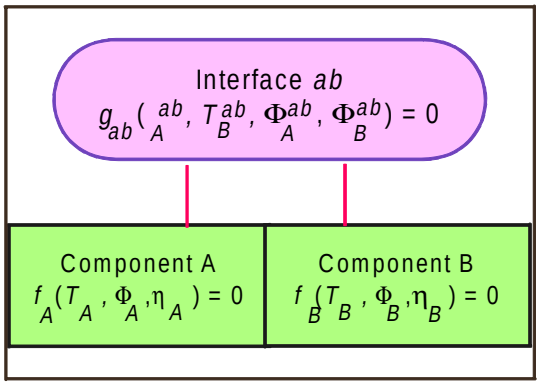

Figure 3 : Hierarchical decomposition tree

Components

In order to express more precisely the component models, we must decide how the component has been isolated from the rest of the world ; we must have decided which boundary conditions are applied on its frontiers. For example, if the A layer of our two-layer wall is limited by a Diriclet condition on the left side, and a Neumann one on the right side, the model g_A has the following expression :

[2] The indices *l* and *r* used to indicate the «left»and «right»sides of a layer are local names. When we use it «outside» their scope, we add a doted prefix in order to distinguish the local variables having the same name in different local models.

$$\begin{bmatrix}\Phi_A^l\\ T_A^r\end{bmatrix} = g_A^1\left(\eta_A, \begin{bmatrix}T_A^l\\ \Phi_A^r\end{bmatrix}, t\right)$$
$$\frac{d\eta_A}{dt} = g_A^2\left(\eta_A, \begin{bmatrix}T_A^l\\ \Phi_A^r\end{bmatrix}, t\right) \quad \textbf{(5)}$$

where g_A^1 and g_A^2 are algebraic operators. We must pay attention that the resulting model is formally equivalent to one which would be based on different boundary conditions. We will assume that some calculation methods (at least one) are available for every elementary component model. These methods must allow to calculate, for every incoming variable involved in the boundary conditions, the value of the corresponding outgoing variables ; the associated method will have to perform the following calculations :

$$\begin{bmatrix}\Phi_A^l\\ T_A^r\end{bmatrix} := \tilde{g}_A^1\left(\eta_A, \begin{bmatrix}T_A^l\\ \Phi_A^r\end{bmatrix}, t\right)$$
$$\eta_A(t_1) := \tilde{g}_A^2\left(\eta_A(t), \begin{bmatrix}T_A^l\\ \Phi_A^r\end{bmatrix}(t)\Big|_{[t_1,-\infty[}\right) \quad \textbf{(6)}$$

which determine new values of the the state function (or vector) η_A at time t_1 in function of the memory of the component and the past inputs evolution. The ":="symbol indicates that the value of the left term is calculated solving the right term, which is different from the significance of the "="symbol which has been used until now and which indicates a formal equality of the left and right terms.

Interfaces

An interface is a set of relations (equalities) which must be respected by variables of the coupled component defined on the respective frontiers. Let us take again our simple example. The equations of interface *ab* are given in **Table 1** where R_{ab} (a residue) is an intermediate quantity.

It must be observed that the different sets of equations obtained by gathering the components ones and the interface ones are formally rigorously equivalent ; they determine the same solution. Let us see now the practical implementation.

IMPLEMENTATION IN XLAB

Principles

The previous way of presenting the equations set to solve will be very helpful at the time we implement the proposed calculation methods. In Xlab, we can use two different functions. The first one is the `dassl`[17] function which is a general Differential Algebraic Equations (DAE) solver, based on backward-difference algorithms.

Boundary condition on frontier r of A	**Boundary condition on frontier l of B**	**Interface equations**
Diriclet	**Diriclet**	$\Phi_r^A + \Phi_l^B = R_{ab}$ $R_{ab} = 0$ $T_r^A = T_l^B(*)$
Neumann	**Neumann**	$T_r^A - T_l^B = R_{ab}$ $R_{ab} = 0$ $\Phi_r^A = -\Phi_l^B(*)$
Diriclet	**Neumann**	$T_r^A - T_l^B = R_{ab}$ $R_{ab} = 0$ $\Phi_r^A = -\Phi_l^B$ $T_A^r(*)$ or $\Phi_r^A + \Phi_l^B = R_{ab}$ $R_{ab} = 0$ $T_r^A = T_l^B$ $\Phi_B^l(*)$

Table 1: Interface equations versus boundary conditions

But, we do not have to care the way `dassl` is internally functioning, we only have to build a function containg equations **(5)**, and to pass it as argument to the `dassl`. This function will return new values of the state variable $\eta_A(t_1)$ at time t_1. Of course, if a component has no capacitive elements, the equation system reduces to an algebraic one and using `dassl` is no longer necessary ; by the way, if the component model is a very simple one (as can be a linear modal model), the time integration can be directly coded using a common integration scheme without using `dassl`. The second useful function available in Xlab is `fsolve` which is a general non linear algebraic equations system solver actuating by looking for roots which minimizes the residue of a set of implicit equations. `fsolve` is used to solve the interface equations set, by giving it the residue expression as argument under the form of a function

of the searched variables. The different possible expressions of the interface equations gathered in **Table 1** will be useful because the expression of residue function which must be provided to `fsolve` is evident ; the variable of which the value is searched for is indicated by an asterisk.

For example, if the right side of the left layer and the left side of the right layer are both limited by a Diriclet condition, MotorLab will proceed to the following calculations. Every layer will offer a Matlab function which calculates a new flux value on the correct side in function of the temperature of the same side and an old state value (object level methods, included in respective object models) :

Express the differential term of the state equations as a Matlab function	$g_A^2(\eta_A, T_A^r, T_A^l) :=$ $F\eta_A + B_A^l T_A^l + B_A^r T_A^r$
Build a function which returns new values of the flux on both sides in function of the temperature values on the same sides	$\eta_A(t+\Delta t) := dassl($ $g_A^2, \eta_A(t), T_A^r(t+\Delta t),$ $T_A^l(t+\Delta t), \ldots)$ $\Phi_A^r(t+\Delta t) :=$ $f_A^r(T_A^r(t+\Delta t), \ldots)$

At the interface level, the following methods, perform the coupling calculations :

Build a function which expresses the residue value on the interface in function of the dual variable, using the functions offered by the components to couple	$R_{ab}(T) :=$ $f_A^r(T, \ldots) -$ $f_B^l(T, \ldots)$
Find the value of the dual variable (interface temperature) which minimizes the residue (flux accumulation)	$T_{ab}(t+\Delta t) := fsolve($ $R_{ab}, t+\Delta t, \ldots)$

Please take into account that the actual implementation is more complex than the one indicated in the previous table. In a recursive implementation, the new value of T_{ab} can be used in other coupling relations. When running Motorlab, a first call is made to the highest "fsolve" which recursively will call the component and interface functions, which themselves use dassl and fsolve, which etc...

Practical items

Every component or interface is viewed as an object, with its own data and methods. The actual components are modeled as *instanciations* of model *classes*. The model classes contain only one method which in reality is a function. The instanciations are obtained by sending to and receiving from the method data which belongs to one particular object.

As every object must be able to actuates under different ways depending on the simulation algorithm stage, calls to the class function lead to different actions depending on a flag given as argument, technique which is rather similar to the Simulink [13] philosophy. Table 2 shows the different possible methods provided by a model depending on the flag value.

Flag	Function role
0	Initializes and pre-calculates intermediates parameters
1	Calculates the future state and output values without bringing the object up to date
3	Brings up to date the object state and outputs (requires first a function call with Flag = 1)
2	Performs calculations and actions required before stopping a simulation

Table 2 : Model flags

EXAMPLES

1st example : a component

The example shows the model of a thermally conductive, monodimensional, homogeneous layer. The file "Layer1D.sci"contains the actual Xlab code which could be automatically generated. Let us note that, due to the limited available space, not all the code has been copied. It can be seen in the source that the presented model is very simple. Due to the powerful of Xlab, writing complex operations is very clear and synthetic.

```
function [VO,S,Sdot,S0,Sdot0,GlobalR] =
Layer1D(t,t0,P,VI,Index,Flag,S,Sdot,S0,Sdot
0,GlobalR)
//------------- Misc stuff --------------
(...)
//
// VERIFYING ARGUMENTS SIZE
// (...)
//------------ <Flag = 0> ---------------
if (Flag == 0) then
XnodeNb = P(1);
Xlength = P(2);
(...)
// INITIAL PRE-CALCULATION OF INTERMEDIATE
```

```
// PARAMETERS (LOCPAR)
//---------------------------------------
(...)
elseif (Flag == 1)
//--------------<Flag = 1>-----------------
(...)
DeltaX = R(1);
DeltaY = R(2);
DeltaZ = R(3);
M = R(4);
(...)
// CALCULATION OF A NEW STATE "X"
// ----------------------------
// implicit sheme
S(Index) = M*S0(Index)+N*VI;
Sdot(Index)=(S(Index)-S0(Index))/(t-t0);
// CALCULATION OF OUTPUTS "VO"
// -------------------------
VO = J * S(Index) + G * VI;
elseif (Flag == 2)
//--------------<Flag = 2>-----------------
// nothing
elseif (Flag == 3)
//--------------<Flag = 3>-----------------
(...)
S0(Index) = S(Index);
Sdot0(Index) = Sdot(Index);
//
// CALCULATION OF OUTPUTS "VO"
// -------------------------
VO = J * S(Index) + G * VI;
end;
return[VO,S,Sdot,S0,Sdot0,GlobalR];
```

Component model class

Deriving an object from a model class is straightforward ; after memory areas are reserved for the object which will be referred to through an index, a parameter list is sent to the class model with "Flag = 0 ". The different methods of the model are then available from the object by using the function, giving it the object index, the flag value, the object parameters list, the input values as arguments.

```
getf('Layer1D.sci','c')
//
// PARAMETERS
//
XnodeNb = 3;   Xlength = 0.5;
Ylength = 1.0; Zlength = 1.0;
Lambda  = 1.7; Roc     = 1e6;
BounCond = [1,1];
//
Left_P=list(XnodeNb,Xlength,Ylength,Zlength
,Lambda,Roc,BoundCond);
//
// Creating a memory space for state and
// dotstate variables S, S0, Sdot and Sdot0
(...)
[S,Index] = AddElemInList(S,[]);
Left_Index = Index;
 (...)
```

2nd example : a system

This example of system model corresponds to the coupling of two homogeneous layers as in **Figure 2**. The actual Xlab code which implements the system model class is a little bit more complex, due to the need to write a specific internal function which calculates the residue value versus the coupling variable value ("deff...").

```
function [VO,S,Sdot,S0,Sdot0,GlobalR] =
PerfCont(t,t0,P,VI,Index,Flag,S,Sdot,S0
         Sdot0,GlobalR)
(...)
//
if (Flag == 0) then
// -------------<Flag = 0>-----------------
NSubModel = P(1);
for i = 1:NSubModel
 (...)
elseif (Flag == 1)
// ------------<(Flag = 1)>---------------
(..)
// Input translation
// Left layer
Left_VI(1) = VI(1);
// Right layer
Right_VI(2) = VI(2);
//
//
// COUPLING WITH SIMULATION FROM t0 to t
//
// I couple faces Left.2 with Right.1
// I define for that a function Flux =
```

```
// F = fct(T)

deff('[F] =
Residue(T,Left_VI,Right_VI,Flag)',[

'Left_VI(2) = T(1);

'Right_VI(1) = T(1);

'[Left_VO,S,Sdot,S0,Sdot0,GlobalR]  =
Layer1D(t,t0,R,Left_VI,Left_Index,Flag,...

S,Sdot,S0,Sdot0,GlobalR);'

'[Right_VO,S,Sdot,S0,Sdot0,GlobalR]  = ...

Layer1D(t,t0,R,Right_VI,Right_Index,Flag,S,
Sdot,S0,Sdot0,GlobalR);'

'F(1) = Left_VO(2) + Right_VO(1);'

]);


// INITIALIZATION

T0 = S0(Index);


// RESOLUTION

[T,F] =
fsolve(T0,list(Coupling,Left_VI,Right_VI,1)
);

//

// STATES

S(Index) = T;

Sdot(Index) = Sdot0(Index);

//

// OUTPUTS

//

// Fluid

VO(1) = Left_VO(1);

VO(2) = Right_VO(2);

elseif (Flag == 2)

//------------<(Flag = 2)>---------------

elseif (Flag == 3)

//------------<(Flag = 3)>---------------

(...)

// Input translation

// Left layer

Left_VI(1) = VI(1);

// Right layer

Right_VI(2) = VI(2);

//

[Left_VO,S,Sdot,S0,Sdot0,GlobalR]  =
Layer1D(t,t0,R,Left_VI,Left_Index,Flag,...

S,Sdot,S0,Sdot0,GlobalR);

[Right_VO,S,Sdot,S0,Sdot0,GlobalR]  =
Layer1D(t,t0,R,Right_VI,Right_Index,...

Flag,S,Sdot,S0,Sdot0,GlobalR);

end;
```

```
return[VO,S,Sdot,S0,Sdot0,GlobalR];
```

Defining a system object needs only to define the actual values of the parameters which are the names and types of the subsystems.

```
getf('PerfCont.sci','c')

//

//

// PARAMETERS

//

Wall_P=list(2,"Layer1D","Left","Layer1D","R
ight");

//

// STATES

//

// Creating a memory space for states var

(...)

// OUTPUTS, simple déclaration

Wall_VO = [];

//

// INPUTS

// I get input values which come from an

// higher system model or are fixed here

//

// Creating memory space for local par

(...)
```

Remaining problems

The translator between a more general modeling language (which could be similar to NMF) and the model class implementation (*.sci files) is not already written. The `fsolve` function which is used in Ψlab to solve the algebraic equations on the coupling interfaces is an external one, written in fortran, and is based on a static memory allocation which does not allow recursive use (if `fsolve` is called inside a function which is itself used by `fsolve`, the required memory is already in use). A dynamic memory allocation based `fsolve` function should be implemented. The limited performances of Motorlab in terms of model size and simulation duration could be partly overtaken by using a translator as MatCom [14] to produce C code based runtimes. But Motorlab, which in fact tends to increase the number of equations and variables, will certainly never be as efficient (in terms of simulation duration) as a one which will have been optimized for a specific class of problems. The (long) simulation time is partly due to the Motorlab modeling philosophy, and partly due to the fact that Matlab is an interpreted language, not a compiled one. We will address this

problem when a compiled version of Motorlab will be available.

It must be noticed also that the present implementation of Motorlab is not as nice as we would like and that the user has already to be aware of all the internal mechanisms of Motorlab ; furthermore, there is not already an automatic translation of the physical relations in the numerical implementation, fact that obliges the user to see the numerical causality.

CONCLUSION

The modeling methodology presented in this paper and its actual (but partial) implementation, MotorLab, have been proposed and are currently used in the framework of the European project ROOFSOL [16]. This allows the different teams involved in modeling activities to have the possibility to share their experiences, and warrants that the results of the project will then be available as executable codes in a simple and available environment, Xlab. One of the final ambitious objectives of this project is to develop a graphical interface, MotorLink, above Motorlab, in order to help to describe the coupling between subsystems and components. We could reuse a part of the Simulink (MatLab) or Scicos (ΨLab) [15] facilities. This would allow to offer a user-friendly modeling environment which would include all the *low-level* facilities of Xlab. From another point of view, we presented only how using MotorLab for simulation purpose. But the methodology, which relies in fact on an object based description of a system, allows easily to imagine how to perform other modeling activities than the only simulation one. Using the same technological/physical initial systemic description, other calculation methods or functionality's, as for example interval calculations, formal calculations, dimensional homogeneity verification could permit direct sensitivity studies, error propagation, etc. without having to perform a huge number of «classical » simulations, as it is generally required until now.

REFERENCES

[1] P. SAHLIN, A. BRING, K. KOLSAKER, "Future trends of the Neutral Model Format ", 4th Int. Conf. Proc. Building Simulation'95, IBPSA, Madison (WI), USA, aug. 14-16, 1995.

[2] Matlab, The MathWorks Inc., 24 Prime Park Way, Natik (MA), USA. *(www.mathworks.com)*

[3] Ψlab, INRIA - ENPC. *(www.inria.fr)*

[4] N.J. BLAIR, J.W. MITCHELL, W.A. BECKMAN, "Demo,stration of TRNSYS use in building simulations ", 4th Int. Conf. Proc. Building Simulation'95, IBPSA, Madison (WI), USA, aug. 14-16, 1995. (*http://sel.me.wisc.edu/trnsys*)

[5] W. BUHL et al., "Recent improvements in SPARK : strong component decomposition, multivalued objects, and graphical interface ", 3rd Int. Conf. Proc. Building Simulation'93, IBPSA, Adelaide, AUSTRALIA, aug. 16-18, 1993.

[6] A. JEANDEL et al., "ALLAN.Simulation : a general software tool for model description and simulation ", 3rd Int. Conf. Proc. Building Simulation'93, IBPSA, Adelaide, AUSTRALIA, aug. 16-18, 1993.

[7] D. BONNEAU et al., "CLIM2000 : modular software for energy simulation in buildings ", 3rd Int. Conf. Proc. Building Simulation'93, IBPSA, Adelaide, AUSTRALIA, aug. 16-18, 1993.

[8] A. BRING et al., "IDA - An environment for building and energy systems simulation ", 4th Int. Conf. Proc. Building Simulation'95, IBPSA, Madison (WI), USA, aug. 14-16, 1995.

[9] J.L. BONNIN et al., "The Almeth project ZOOM code : results and perspectives ", 2nd Int. Conf. Proc. Building Simulation'91, IBPSA, Sophia-Antipolis, FRANCE, aug. 20-22, 1991.

[10] J.L. BONNIN, J.Y. GRANDPEIX, J.L. JOLY, "Formalisme d'évolution par transfert : le projet ZOOM ", Séminaire spécialisé "Outils d'aide à la conception et à la gestion ", AFME, PARIS, oct. 26-27, 1989.

[11] R. EBERT, B.PEUPORTIER, G.LEFEBVRE, "Simulation of thermal building behavior based on an object-oriented ADA implementation ", Conf. Proc. Building Simulation'91, IBPSA, Sophia-Antipolis, France, aug.1991.

[12] J.L. LE MOIGNE, La théorie du système général, PUF, PARIS, 1984 and 1997.

[13] SIMULINK, The MathWorks Inc., 24 Prime Park Way, Natik (MA), USA. *(www.mathworks.com)*

[14] MATCOM, MathTools Ltd, 1996. *(www.mathworks.com)*

[15] SCICOS, INRIA - ENPC. *(www.inria.fr)*

[16] ROOFSOL, "Roof solution for natural cooling ", European project n°JOR3CT960074, DG XII, 1996-1998.*(http://roofsol.ciemat.es/roofsol)*

[17] L.R. PETZOLD, *A description of DASSL : a differential/algebraic system solver*, SAND82-8637, Sandia National Laboratories, sept. 1982.

[18] R. EBERT, *Développment d'un environnement de simulation de systèmes complexes. Application aux bâtiments*, PhD thesis, Ecole Nationale des Ponts et Chaussées, PARIS, 1993.