SOFTWARE COMPONENTS FOR DYNAMIC BUILDING SIMULATION

Sana Gaaloul¹, Benoît Delinchant¹, Frédéric Wurtz¹ and Franck Verdière¹

¹G2ELAB: Grenoble Electrical Engineering LABoratory, Grenoble, France

ABSTRACT
This paper describes a standard of software components for building energy and control systems modeling. This technology is well positioned to overcome the limits of existing tools and to allow better collaboration between various engineering domains coexisting in building field. A standard named ICAr is proposed in addition to the Modelica approach to ensure models interoperability. So, the strengths and lacks of the “white box” modeling based on Modelica language will be firstly discussed in order to prove the necessity to introduce the “black box” modeling based on software components and to highlight the complementarity of these two approaches via “plug’out” and “plug’ in” notions.

As the white modeling was standardized by defining universal languages, this work proposes a norm of software component named ICAr. So, this paper defines the specifications of the ICAr standard: inputs, outputs and interfaces for external communication. Moreover, to demonstrate the importance of this approach in building performance simulation, a thermal envelope will be coupled to a simple heater model in different simulation environments using a mixed approach.

INTRODUCTION
In the challenge to reduce buildings consumption and CO₂ emissions, it’s essential to ameliorate the envelope isolation and equipments performances. This needs several simulations allowing virtual prototyping and avoiding expensive experimentation. That is why several simulation tools and many equipment models are developed nowadays. These tools are characterized by different capabilities of modeling and calculation for different requirement such as fine or system level modeling. But, the use of one model in different tools requires an important development effort and model reuse became an hard task. Therefore, it is important to improve tools interoperability to have easier model exchanging.

The IEEE glossary defines the interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged”. Two principles interoperability approaches are identified: the white and the black boxes approaches.

A white box model is a model defined by various typed quantities, and its content is transparent and accessible (Allain & al., 2002). Developing interoperability through a white box approach means that there is a common format for all models in which they will be written. Diverse models must be built and interpreted using a common template: content standardization.

The common format can be an independent modeling language like the standard VHDL-AMS (Hervé, 2002) or Modelica (Fritzson, 2004) that was widely used in building domain (Wetter 2009, Felgner & al. 2002…).

In contrast, a black box model is only described by its interfaces, masking its content. The interoperability based on this approach requires the specification of a common format, but only at interfaces level. This unified and simplified structure presents a way for semantic equivalence allowing the data mapping between models and applications: interfaces standardization.

The standard of software component presents an emerging technology for such a type of interoperability.

Although several building simulators are using these technology for model import and export (TRNSys’s type, Simulink’s S-function…), no efforts have been made to establish a standard for model exchanging which guaranties the compatibility between these tools.

In this paper, we will show the necessity of software components standard using (black-box approach) in the building community, in complement to other interoperability approaches (white-box approach). For this, ICAr (Interface for Component Architecture) norm is proposed, that could be standard of a black box approach. This norm will be able to take into account the specificities of building simulation. Such a standard will reduce the effort, time and cost in implementation, and promote the exchange of models between community’s members.

NEEDS OF INTEROPERABILITY AND USES OF WHITE AND BLACK BOX APPROACHES
Building simulation is becoming more and more various and heterogeneous involving components
from different engineering domains. Many BPS\(^1\) tools have been developed in order to help architects and researchers to evaluate energy efficiency.

The U.S. Department of Energy has published a report (Crawly \& al., 2005) comparing features and capabilities of 20 of the most used tools. A quick tour in this report lets conclude about strengths and lacks of each one. There is no tool allowing a complete simulation of different building components and the most of these tools seem to be complementary to each other.

Co-simulation should be a possible solution to couple these tools together. It can be a direct coupling between two simulators, with a master and a slave. This is the case of Fluent and ESP-r coupling (Djunaedy \& al., 2003) or TRNSYS and MATLAB coupling. Another technology based on a middleware managing the communication between tools using sockets (a stream between two computer codes), was implemented in the BCVTB (Wetter \& al., 2008).

An alternative way to ensure complementarity and collaborative modeling between tools consists on exchanging models. This means the extraction of existing models previously developed in one specialized tool for externally reuse in other tool in order to benefit from the expertise behind.

The model exchanging needs software modularity as defined by Mazzarella, 2009. Tools have to allow easy ways to introduce and use new models. In return, they must ensure safe model extraction.

Using tools based on object oriented programming presents more facilities for model developing and reuse. Modelica as an emergent OOM\(^2\) language is an interesting solution to ensure interoperability.

**White Box Interoperability approach based on Modelica: strengths and limitations**

The white box interoperability based on Modelica consists on redeveloping of all subsystems driven from various engineering domains in the same language (Kossel \& al., 2006). This multi-physical language is well positioned to become a standard for modeling in different fields like automotive industry, energy systems, defense industry...

Modelica is a language developed by the Modelica association since 1996 (https://www.modelica.org/) that established the free standard Modelica Library gathering basic models from several domains.

Beside the standard open source library, several efforts have been made to build various libraries for building modeling. As an example, we note: the HumanComfort library (Michaelsen \& al., 2009) which elaborates basic models for occupants’ thermal comfort prediction, the Buildings library (Wetter, 2009) for building energy and control systems, ATplus library (Felgner \& al., 2002) for thermal building behavior simulation.

It is an object-oriented programming language offering faster prototyping; natural modeling of complex systems (Cellier, 1996) and easy code reuse. A large system can be decomposed on multiple subsystems treated separately and reassembled after that by connections, and then resolved with appropriate numerical solvers. Moreover, existing models can be reused by import from the toolbox or by possible extension.

Modelica specifies the possibility to describe hybrid DAE\(^3\) complex systems behavior by formulating the corresponding equations which allows natural modeling in contrast with causal (block) description. All points previously detailed show that using Modelica is a powerful solution for models interoperability. However, it can not be a complete and unique language since we know that it is theoretically impossible to build a complete consistent formal language for all type of physical simulation as Gödel's incompleteness theorems (1931) proved it for all formal systems.

Besides this fundamental theoretical limitation of any language for a white box approach, technical problems can be also affronted as showed by Mazzarella, 2009 in the case of Modelica. He evoked poor run time efficiency, difficulties to resolve PDEs\(^4\) and to understand some simulation errors message. Moreover convergence problems have been pointed by Wetter \& Haugstetter, 2006 for simulating multi-zone building modeled in Dymola.

Moreover, there are other practical reasons showing that a white box language cannot be, and should not be, the only way for inter-operability like:

- Translating models to another language presents some risks of robustness loss when a faithful translation cannot be warranted and some errors cannot be avoided. Whereas conserving the model in his initial tool, where it was tested and validated, is more insurant.
- The recoding of models from other tools requires that they are open sources, so that their variables, parameters and equations can be identified and reused. This is not the case of all BPS tools. For example, in TRNSys, if some models are documented, many others are not, and only a DLL\(^5\) (Riederer \& al., 2009) is provided (they are provided in a black box form).
- Many models have been already developed in several tools and redeveloping all of them in

---

1 Building Performance Simulation
2 Object Oriented Modeling
3 Differential Algebraic Equation
4 Partial Differential Equation
5 DLL file (dynamically linked library) for Microsoft Windows and OS/2 environments and SO file (shared object) for Unix and Unix-like platforms.
another unified environment will take a huge amount of time and financial wise.

- This solution did not answer the need to improve existing tools that is essential to profit from their modeling and resolution capabilities. In fact, BPS tools must be able to include new innovative and complicated systems governed by control laws (plant system, PV, new HVAC units...).

The solution proposed to overcome these difficulties is to use, in complement of a standard modeling language (like modelica), the software component approach.

**Black box Interoperability Approach based on software components to complete the white box approach:**

Software component is defined by Szyperski, 1996 as “a unit of composition with contractually specified interfaces and explicit context dependencies only”. It is an autonomous deployment entity, encapsulating a software code. This binary code unit masks its content and only shows its interfaces to the receiver. This computer paradigm was introduced after the oriented object programming to overcome its limits and its incapacity to build decomposable and autonomous entity from existing tools.

A standard of software components specifies interfaces that define the kind and the format of information that will be exchanged by the component. This standardization approach allows easier models sharing because models will have known and normalized interfaces. A tool that has already implemented these interfaces will receive and use the component (plugged in) without any difficulties.

A software component must also specify its:

- Packaging and encapsulation methodology: how it can be generated and how it is composed.
- Deployment and introspection mechanism: how to use the component, to interrogate it, and to know its available services.
- Offered interoperability process: via http or web services for example...

Many efforts have been made to improve BPS tools modularity in order to generate and receive black box models (TRNSys, Matlab/Simulink...). TRNSys for example, allows the development of new models (static or dynamic) via fortran or C templates. It is then compiled on a library (.dll or .so) and used after that as a TYPE that can be connected with other Types from its library. As said, each type is defined by a corresponding DLL that can be externalized (Riederer & al., 2009).

But there is a lack of syntax standardization (standard programming interfaces) and semantic differences between models due to the approach of resolution (differential equations, time discretizing) and variables nature (continuous, events...).

A standard of software component is established when there are a reasonable number of clients that share the same requirements (Szyperski, 96). They are sharing the same type of components with the same implementation and interfaces. To reach this goal, those components must especially provide the maximum of services that can be expected to a wide use.

The computer industry and computer science community have specified technical standards like Microsoft's COM, OMG's CORBA and Sun's JavaBeans. Some communities, with their own needs for special trades, have specified other specific components like RTSC (Real Time Software Component, http://www.eclipse.org/rtsc/) or AUTOSAR (AUTomotive Open System Architecture, http://www.autosar.org/). This last was born as a result of a strong partnership between many automotive manufactures (BMW, TOYOTA, FORD...).

Recently, under the European project MODELISAR, new interface standardization FMI (Functional Mock-up Interface, http://www.functional-mockup-interface.org/) has been specified. It is based on ODE formalization and C language, and allows model exchange between Modelica and non-Modelica, partners' software tools (Dymola, AMESIM, Matlab/Simulink...).

Building simulation developers and users have also expressed such a need for collaboration and standard establishment. The use of a standard of software components in building community will offers many advantages for model sharing:

- Possible automation of component generation from various tools supporting the standard. The component will be available for external reuse in other tools.
- Portability, as the software component is an antonymous an executable entity.
- Robustness, when models are not redeveloped but extracted from their parent tools with their implementation.
- Confidentiality preserving: model content and equations, in the component approach, can be compiled in a binary code ensuring the hiding of the known-how which is behind.
- Compatibility with many tools: It is generally developed with the aim of interoperability, using language that can be widely adapted to various interfaces offered by simulation programs.
- Composition: software components are made to ease their composition in order to build a composite system.
Plug’in, Plug’out and software component bus:
Two concepts derive from these two approaches, allowing bridges from a concept to another: the “plug-out” and the “plug-in” (Figure 1). A plug-out is defined as a computer code allowing exporting features of modeling/simulation from an existing tool. In our context, it means the generation of a software component from a white box model, or from another black box provided by a simulator. A plug-in consists on importing a software component into a white box model, or into a simulation program in order to add more modeling features.

![Figure 1 Plug’in and plug’out](image)

Thanks to these notions, a software component can be, for example, generated from a Modelica prototype of new equipment and integrated after that in another tool to be connected with its models. In the other hand a Modelica environment can integrate a software component which was extracted from other existing tools as a black box.

Software component bus can be created to ensure available components sharing between various tools. As its hardware equivalent, this platform allows the loading, introspection, configuration, unloading, replacement of software components and communication channelling between components.

The standard of software component is a powerful candidate to complete the interoperability approach based on Modelica. Our standard for building models exchange was initially developed for electrical system optimization needs (Delinchant & al., 2004) and was after that improved for mechatronic systems due to the development of RELUCTOOL (Du Peloux & al., 2006).

In the next part, we will present the specification of the normalization established for the ICAr as a candidate for building community standard.

ICAr: A SOFTWARE COMPONENT NORM FOR BLACK BOX STANDARDIZATION IN BUILDING MODELING AND SIMULATION:

ICAr specifics
ICAr is a specific pattern of standardization of software component, it is an executable Java code defined by its inputs/outputs variables and the services that it can bring. One of the specificity of ICAr compared to other physical domain standard (especially Autosar) is its evolutionary architecture. It is possible to extend component capabilities by adding new functionalities such as several dynamic semantics, optimization, documentation, etc. A second specificity is the use of Java as a base language of development. It can also contain other data types (txt files, C / fortran, dlls...)

This last characteristic offers the ICAr more portability because Java is an independent platform language. In fact, when compiled, a byte code is generated that didn’t contain native code in contrast with C language. This will be done by the JVM (Java Virtual Machine) when running the java program.

Besides, the ICAr is compatible with the OSGI (Open Services Gateway Initiative, Marples and Kriens, 2001) service platform. This platform allows the sharing of services from over wide area to local networks and devices. This technology was used for example; to build a service oriented smart home architecture (Wu & al., 2007). So that, an ICAr service can be exchanged via web services and used to address homes devices embedded simulation.

As already said, ICAr is characterized by an evolutionary architecture and can be seen as a multi-services component. Services are available via different facets that are accessible via the interface “component” (Figure 2). This interface presents the way to instantiate the component, defines its inputs and outputs, and how to interact with all its features. ICAr is so characterized by a generic way for services introspection.

![Figure 2 ICAr component](image)
supported). A special focus is given in this paper to hybrid ODE simulation facet.

**ICAr interfaces specifications for dynamic simulation (ODE semantic)**

An hybrid ODE model deals with various variables natures: continuous dynamics, discrete state variables and events. It is a combination of continuous models and discrete ones with event handling.

Therefore, four mainly interfaces were developed in ICAr: **CommonStateSystem** interface for common functionalities like initialization, **ContinuousExplicitStateSystem**, **DiscretStateSystemAndEvents** and **HybridExplicitStateSyst**, that deals with deferent phenomena. After that, proper methods were implemented in each interface in order to be compatible with many API offered by various tools for new models introduction.

As a result of these methods implementation, the ICAr must have as inputs: the state vector \( X \), external inputs \( U \) and parameters vectors \( P \) (Figure 3). It provides as outputs: state vector derivatives \( \dot{X} \), algebraic outputs \( Y \), initial state vector \( X_0 \) and state variables names \( X_{names} \).

**ICAr Use: plug’in and plug’out**

An ICAr component can be constructed directly in Java using the development kit (ICArDevKit) and can be generated automatically (plug-out) from various sources having a white box description (Modelica, COMFIE) or a black box one (FMI components…).

After that, it can be plugged in Modelica solvers (Dymola, AMESim...), Matlab/Simulink, TRNSys, VHDL-AMS/SMASH (Rezgui & al., 2011)... Some of these plug-outs and plug-ins will be detailed when needed for applications achievement. They are used to illustrate use cases but once developed, they still relevant for each ICAr component.

**EXAMPLES OF ICAr COMPONENT GENERATION AND USE:**

To illustrate the importance of the ICAr approach, the different stages of his generation and use in some BS tools, two examples are described.

**First Example: generation of an ICAr from COMFIE and its use in Dymola**

This example deals with the generation of an ICAr component containing a building thermal envelope, built using state-system matrices issued from COMFIE. This component is projected in Dymola and coupled with an electrically controlled heater (Figure 4).

**COMFIE Plug’out**

COMFIE (Peuportier & al., 1990), is a software for temperature simulation and energy requirements evaluation during a fixed period for predefined scenarios. It offers a zonal modeling of the building taking into account several parameters (architectural structure, walls compositions and orientations), weather data and internal powers.

Envelope model is firstly established for each thermal area, using heat exchange equations at finite meshes level. To simplify the system describing each area, several mathematical manipulations are applied: base changing, order reducing... After that, all systems are coupled in a global one that will be solved iteratively at each fixed time step. It is expressed in a discret-state form (eq. 1) like many others envelope models (TRNSys, Energy Plus...).

\[
\begin{align*}
X(n+1) &= f(X(n), U(n), p) \\
Y(n) &= g(X(n), U(n), p)
\end{align*}
\]

(1)

However, this tool did not allow the introduction of new equipments and does not offer any possibility to communicate with other tools for co-simulation.

For these reasons, the building model established by COMFIE will be extracted in order to be externally resolved and coupled to other equipments.

The ICAr pattern allows a direct description of this model using the discrete facet, but we choose the adaptation to an ODE semantic (eq. 2), to have more step time choosing flexibility. So a variable step simulation can be adopted ensuring better simulation efficiency (Gaaloul & al., 2011).

\[
\begin{align*}
\dot{X} &= F(X, U, p, t) \\
Y &= G(X, U, p, t)
\end{align*}
\]

(2)

Once executed, COMFIE establish project’s state system matrices and save them in external text files. By exploiting these files, the state system can be easily reconstructed in the ICAr component using the development kit. The **continuousStateSystem** facet was implemented and the needed methods were defined from the state system description (Figure 5).
So, the white box COMFIE Model is encapsulated into a black box model to be autonomously simulated in many other tools.

**Dymola Plug’in**

Modelica offers the opportunity to extend his functionalities by calling external C functions (Modelica spec., 2010). An intermediary step is needed to define ICAr functions (written in Java) calling in a C project (Figure 6).

The C native application launches a Java virtual machine and use Java objects from its memory to ensure ICAr component loading and its methods calling using JNI. The JNI is a powerful interface allowing a Java application to invoke a native application and symmetrically by adapting data structures between the two applications.

A C/C++ dll project is built gathering different C functions for several manipulations: JVM launching, JNI adaptation routines, ICAr facets loading, ICAr interfaces introspection and methods calling.

Once the adaptation between the ICAr component and the C code is made, each C function can be called from Modelica using C external functions.

Finally, a Modelica-skeleton, corresponding to the architecture of the ICAr model, is generated automatically by a Java program. This program introspects the component in order to determine its facets, different inputs and outputs and generates the corresponding Modelica block description.

This block defines it inputs/outputs, state variables and parameters... It also describes relations between them, in the equation section that will be executed during the simulation. The figure below illustrates interactions between ICAr-COMFIE and its Modelica block during a simulation.

**Models coupling in Dymola**

Once imported in Dymola, the ICAr-COMFIE can be connected to other Modelica models: continuous, discrete or hybrid. In this example, it is coupled to a heater (second order model) that is commanded with an hysteretic thermostat (Figure 8): The heater power is injected in the stresses vector U at each time step for temperature calculation.

As shown in figure 9, the internal zonal temperature is maintained around the setting temperature (21°) due to the electrical controller (tolerance: 1°).

So, the generated ICAr component can be distributed and used in different tools like Dymola taking advantages from their capacities.
Second Example: ICAr-COMFIE used in Simulink

The second example will show that once generated, the ICAr component can be imported in different environments guaranteeing the same results for the same simulation conditions. The ICAr generated from COMFIE is imported in Matlab/Simulink and coupled with the same heater model (Figure 10).

![Image of Simulink import](image)

**Figure 10  ICAr component, generated from Comfie plug'out, simulated in Matlab/Simulink**

**MATLAB/Simulink Plug’in**

Simulink allows the introduction of time independent models due to S-function block that will be exploited in this plug’in to import the ICAr.

There are two main strategies to implement an S-function in Simulink. The first one consists on using the C language. It presents better performances because the model is already written in the native language and did not need to be compiled. The second one uses the matlab script (M file), an interpreted and more flexible Language. This second alternative was chosen when the Matlab language owns its proper JVM able to interpret the Java which avoids the C bridge as it was made for Dymola.

A 1-level M S-function template has been defined to call ICAr methods. So, when Simulink invokes the S-function, at each computation step, the M-functions calls the suitable ICAr methods and delegates the computation to the ICAr which return the request data.

**Models coupling in Simulink**

The ICAr-COMFIE is implemented in an S-function and can be connected to other models from Simulink libraries like it was shown in Figure 8.

Results obtained for these coupling in Simulink are similar to those obtained in Dymola (Figure 9) when using the same solver. This result proves that the ICAr component is executed independently of the simulation environment.

The ICAr-COMFIE embedded in an S-function can be also connected to Modelica derived models using the Modelica Physical Modeling Toolbox for MATLAB\(^6\). This is a white box solution that is possible only if model equations are available.

**Use cases conclusion and generalization**

ICAr approach is a solution proposed in order to improve models interoperability in building simulation domain in complement to the Modelica approach. The two detailed applications illustrate:

- The extraction and encapsulation of dedicated model like COMFIE. It consists on the encapsulation of PDE modeling capacities behind this final model, ensuring the robustness provided by a proven and widely used tool.
- The distribution of a generated ICAr component and its reuse in different simulation software like Dymola and Simulink, independently of its source. It is a separation between modeling skills and system simulation..
- The plug’in of the ICAr-COMFIE in other tools import supplementary capabilities of PDE’s resolution that are partially (Dymola) or not (Simulink) supported by targets.

In these examples, the ICAr component was generated from COMFIE, but many other standard simulators can also plug’ed out an ICAr. A plug’ out is possible when the tool offers a white box description of models like Comfie or Modelica or when it allows the export of a library corresponding to the model with specified API like FMU or TRNSYS dll’s.

In this perspective, other plug’ out has been developed allowing the automatic generation of ICAr components from a Modelica white box models or from FMU black box models.

In the other side, developing plug’ins to a specified tool is possible when it offers a structured interface allowing the addition of new components like TRNSYS’s “types” and Simulink’s S-function.

**CONCLUSION:**

This paper shows the role of the black box approach to overcome modeling languages lacks and to make easier models exchanging in building simulation context. ICAr component generation and projection was detailed for specific tools but it is also valid for many other environments offering interoperability possibilities.

The adoption of the ICAr as a standard for interoperability in building community will ensure: secure model exchanging between developers (robustness and privacy), overcoming existing tools limits by easier model introduction with richer modeling capacities, testing innovative systems in non specialized tools and various composition services...

ICAr norm is now used in many research projects and tries to reach a good adequacy with BPS requirements. Many other plug’ins and plug’outs are under development to allow a widely use of this pattern. Besides, multiple services can be also added to satisfy larger building simulation use cases.

---

REFERENCES:


