An optimizer using the software component paradigm for the optimization of engineering systems

Benoît Delinchant, Denis Duret, Laurence Estrabaut, Laurent Gerbaud, Huu Hieu Nguyen, Bertrand Du Peloux, Harijaona Lalao Rakotoarison, Franck Verdière, Frédéric Wurtz

An Optimizer using the Software Component Paradigm for the Optimization of Engineering Systems

B. Delinchant, D. Duret, L Estrabaut,

L. Gerbaud, H. Nguyen Huu, B. du Peloux, H.L. Rakotoarison, F. Verdiere, F. Wurtz,

(*)Laboratoire d'Electrotechnique de Grenoble (LEG)

UMR5529, CNRS/UJF/INPG

ENSIEG, Domaine Universitaire, BP 46,

F-38402 Saint Martin d'Hères Cedex, FRANCE

Contact : cades.support@leg.ensieg.inpg.fr

Abstract. This paper presents CADES, a framework dedicated to system design, based on optimization needs. The framework is defined with a standard implementing the software component paradigm and a pattern to use it. Indeed, this pattern details how to create and use a component (the model of the device to design). This paper highlights the component standard, a generator based on analytical expressions of the system, and an optimization service.

Keywords: Software Environments for Optimization, Optimizer, Software Components, Optimization Framework.

I. Introduction: the exploration of the software components for the sizing and optimization of systems

This paper is an investigation of a new kind of software architecture, using the software component paradigm for sizing, by using optimization approaches, engineering components and systems. In the field of software architecture, one of the previous important paradigms, widely used in the area of optimization software, is object oriented languages. If object oriented approach is indeed very powerful, one of its limit is the reusability of objects from one development project to another. The new software component paradigm is a new approach of computer science [1] that wants to be a solution to this problem of reusability. Based on

this approach, it will be shown in the following how it is possible to build global software architecture for optimization of components and systems with the goal of reaching the following needs:

- Need 1°: Need of an architecture offering solutions for the availability of models for components and systems. This means especially reaching two sub-needs:
    - Need 1° a: Need of a software architecture that allows managing methods and tools for designers to create the adapted model for each component and system, with multi physical modeling constraints (*magnetic, thermal, mechanical…*), and with many different levels of modeling (*analytic, semi-analytic, numerical*).
    - Need 1° b: Need of software architecture that allows capitalization and the reuse models from one project to another.
- Need 2°: Need of an architecture offering solutions for managing as much as possible optimization strategies and algorithms (*stochastic, deterministic, hybrid, Pareto…*).
- Need 3°: Need of an architecture offering solutions for connecting software components containing models with software components containing optimization algorithms by managing constraints of reactivity, modularity and interoperability.

II. Pattern Framework/Standard

A. Initial definitions

1. Pattern

A pattern is a model describing how to implement a general concept in different cases. It is largely used in software development with “design patterns” which defines “good ways” to program recurrent architectural concepts. The definition of a pattern is the result of an archetype appearing (*a primitive model*) from different applications, then it will be used in next applications.

In the following, two kind of pattern will be detailed. The first will define the way of using the CADES Framework, with its 3 distinct entities (*Generators, Components, and Services*). The second will define the way of extending CADES functionalities through the ICAr software component standard.

2. Black box / White box

Models are generally contained into tools (*like FEM simulation software, algorithms and equations*

implemented in a programming language), or standardized language (*like Modelica*[1] *or VHDL-AMS*[2]). First kind can be considered as black box and second as white box. Both produce outputs thanks to input values, corresponding to the direct problem solving. Black boxes and white boxes may then be described as follows:

- Black box [2]: brought by cybernetic, this concept enables manipulation of object without knowing its internal constitution. Only relations between inputs and outputs can be observed corresponding to its behavior.
- White box [3]: It is characterized by a set of structured and ordered data. It defines outputs values depending on inputs ones, but also how to obtain these outputs, corresponding to its nature.

Table 1: Black Box versus White Box.

	Black Box	White Box
capitalisation / diffusion	Parameterized computable capacities with knowledge protection	Customizable modelling with knowledge diffusion
composition	Weak coupling, multi-method allowed	Strong coupling only if same language

3. Software Component as Black box implementation

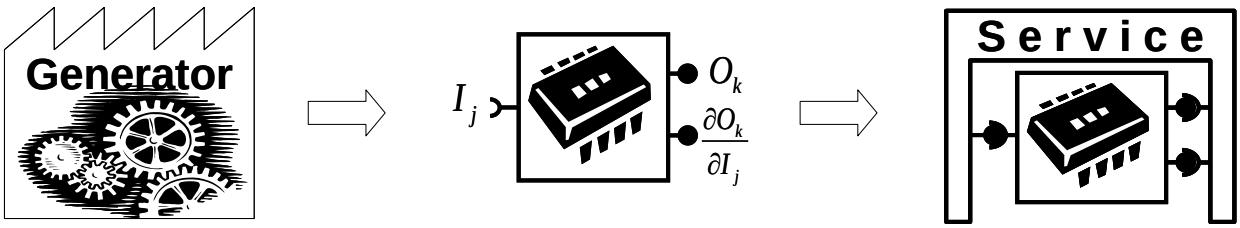
Software component are based on component paradigm which is largely used in electronic where electronic systems are realized by composition of elementary components. A software component is defined as an autonomous deployment entity, which encapsulate computer codes and describes by interfaces its interactions which are allowed with other components [1]. Software components are implementations of black box concept and are well suited to our engineering model encapsulation.

In the following, a black box component is specified which is based on Sun Microsystems's JavaBeans software component standard, especially for portability requirements.

B. Pattern of framework for environment using software component

CADES (*Component Architecture for Design of Engineering Systems*) environment has been developed around a software component dedicated to optimization. The following design pattern formalizes the creation and use of component during design process. First of all, **components** (*models*) are created by "**generators**" and then can be used into "**services**".

[1] http://www.modelica.org/
[2] http://www.vhdl.org/



(a) (b) (c)

Fig. 1 – (a) Generators create (b) components / (c) Services use (b) components.

As defined in Fig. 1 (b), component computes outputs from its inputs and computes also sensitivities (*partial derivatives of outputs depending on inputs*). This last capacity is important especially for optimization requirements as it will be shown in the following.

CADES framework owns several generation tools which create components from the following designer information:

- Multiphysical algebraic equations [4].
- Electromagnetic reluctance networks [5].
- Electromagnetic geometrical representations [6].

After creation, components can be used into CADES services like computation or optimization [7]. Optimization service offers different algorithms like SQP (*sequential quadratic programming*) [13], genetic, hybrid, global, or mixed discrete-continuous.

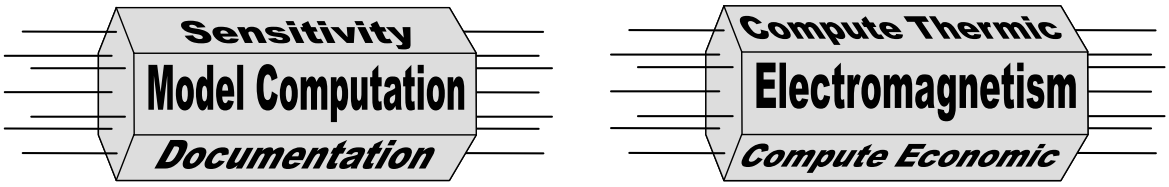
This pattern is completed by specific tools:

- Composer [8] (*use and generate components*): allowing black box component composition to create system model.
- Projectors: translate components to other standards or project them into commercial environments (*Matlab, etc*.).

This pattern, which answers a part of needs 1°, 2° and 3°, and has to be applied to extend the CADES framework with other generators and other services based on the actual standard of components. Moreover, as it will be shown in the next paragraph, the extension of the component's standard induces far more extensibility of the framework.

C. Pattern for the design of software components

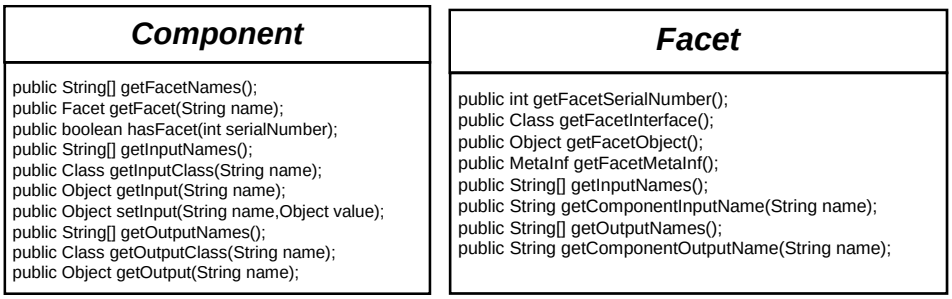
A pattern has also been defined for the standardization of components. ICAr (*Interface for Component ARchitecture*) represents the bases of our black box component and is issued from Vincent Fisher thesis works [9]. ICAr offers a unique interface to interact with all component capacities which depends on design requirements or needs. Such a component is called “multi-facets”, where each facet may represents behavioral model, sensitivity computation, temporal simulation, accurate model, model visualization… Facets may also represent a part of the system to be simulated (*electromagnetism* + *mechanic* + *thermic* + *economic* + *…*).



(a) (b)

Fig. 2 – Multi-facets component: (a) multi-information, (b) multi-physics.

This pattern, which answers part of needs 3° and 1°b, specifies two major objects which have to be used for the definition of ICAr components. Component interface is the anchor point of an ICAr component. It defines input parameters of the system as well as all available facets which are defined and useful for the device design.



(a) (b)

Fig. 3 – ICAr interfaces: (a) Component, (b) Facet.

III. CADES v1.0: Component Architecture for the design through optimization of engineering systems

A. Application in the area of sizing and feasibility studies

CADES v1.0 has been defined to be used as a design framework of engineering systems but especially for

the preliminary design stage. Indeed, the first available generator was an algebraic equation generator, since it answers multiphysic system modeling needs, and also offers a quick reactivity to test many solutions.

During early stage of design, the problem is generally not fully defined (*ill defined problem*). A solution to converge to a good problem definition is to start the design by finding potential solutions and applying constraints to see how these solutions react (see Fig. 4). Then the problem formulation converges during solution convergence that requires many tries and then a specific modeling and optimization framework.

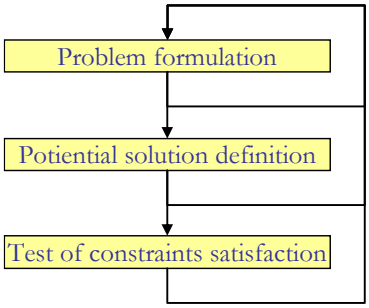


Fig. 4 – Early stages of a design process.

From an analytical description of potential solution, it is possible to extract sensitivity expressions from symbolic derivation of equations. Sensitivity information is helpful during constraint satisfaction or device optimization. Component standard defines in CADES framework 1.0 contains a facet to compute model but also a facet to compute sensitivity of the model (*partial derivatives of outputs depending on inputs*) that are both used inside of optimization services.

B. Description of the CADES framework

1. Component Generator: an extendable generator of software components

First of all, designer defines a SML file (*system modeling language*) which models the device with algebraic equations. After parsing this file, Component Generator performs an analysis to check the model validity (*loops, number of arguments in functions…*). Then, an automated process creates a code that computes the model. Lastly, an encapsulation creates the final software component, compliant with the ICAr specification.

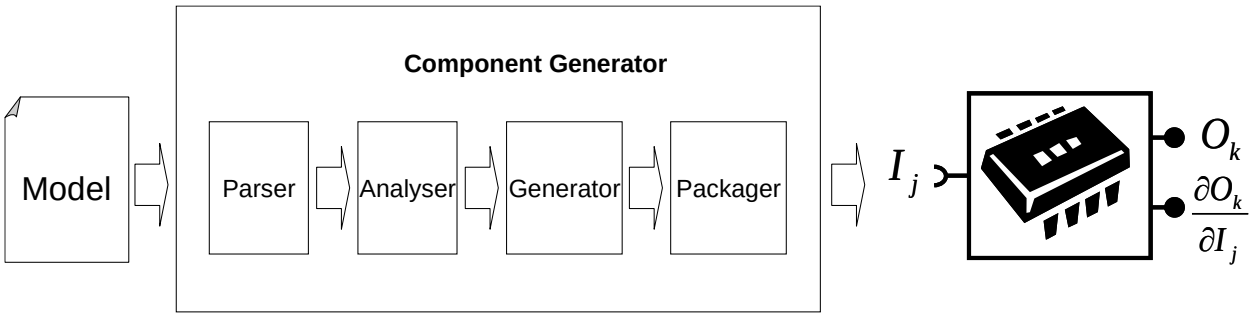


Fig. 5 – Detailed steps of generation process.

As previously said, Component Generator creates a software component from a description (*a model*) of the device. Then, the creation of the model file do not requires any computer science skills, because the designer gives its equations and functions modeling the device as if he writes it plainly in a sheet of paper.

Moreover, as generator produces computation code, it also derivates symbolically the equation and produces also the sensitivity computation code. This full automated process avoids human errors and lets designers concentrate on complex task like finding modelling equations.

Component Generator provides an advanced support for special behavior. When mathematical expressions are inefficient for an accurate modeling, the designer can describe this behavior with various executable codes (for example C, java).

Based on previous tools [10], our generator architecture allows modularity and adaptability. Firstly, thanks to XML technology, just one XML file gathers all instructions supported by the generator. So, a modification of the parsing step simply consists in modifying this XML file. Secondly, information built after the analyzing step does not depend on a specific programming language. As a consequence, projection of this model to several executable codes (Java, C, C++) shall be possible more easily. Lastly, some disconnection between generation and packaging step allows various packaging for the software component. Nowadays, it is compliant with the open standard developed in our lab and creation of other component such as S-function[3] is possible.

2. CADES component standard

CADES framework v1.0 defines a component which answers needs of engineering system sizing. Indeed it compute model but also sensitivities. It was implemented with the ICAr pattern and contains two facets:

- ModelSolver : which is able to compute output parameters depending on input ones.

[3] www. mathworks.com

- JacobianlSolver : which is able to compute partial derivative of output parameters depending on input ones.

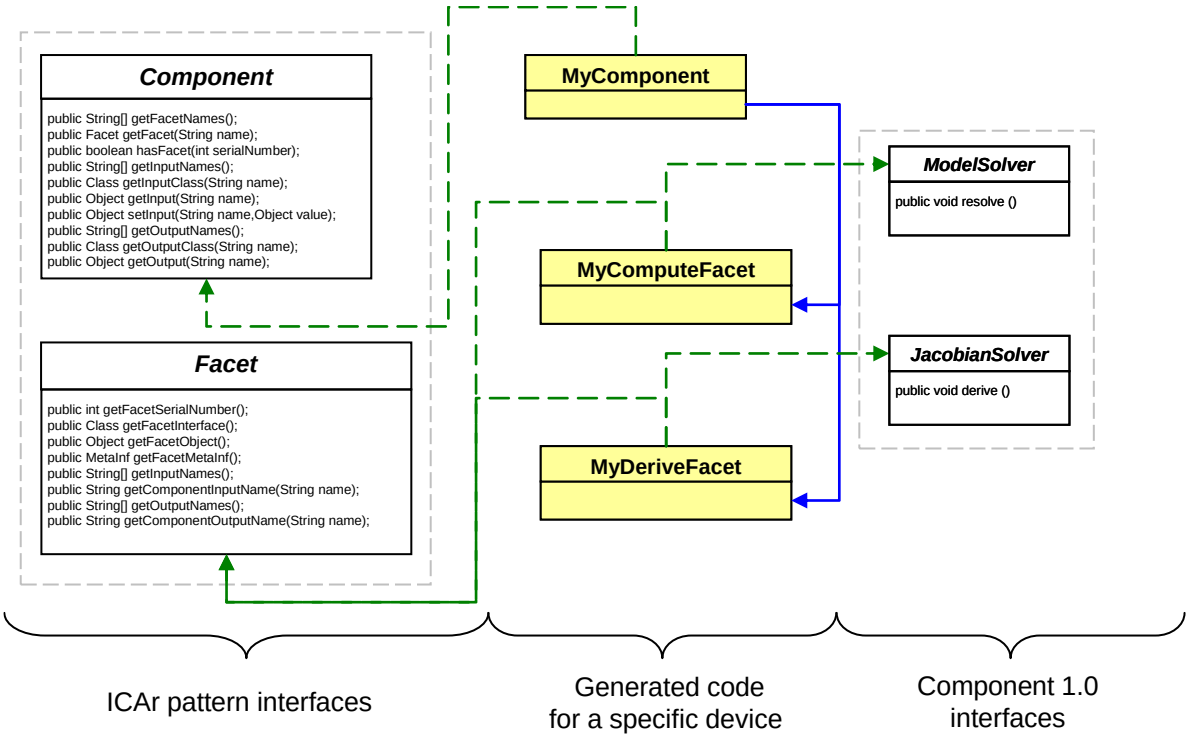


Fig. 6 – CADES Component v1.0 implementation example.

Based on this component standard, generators produce a Model Solver facet or both Model Solver and Jacobian Solver facets. Then services which know also Component standard v1.0 can use all these components.

3. *Component Optimizer: an optimizer using software components*

The optimizer service answers need 3°: connecting as easily and quickly as possible software components containing models to software components containing optimization algorithms [11].

That is why, the optimizer we have developed, has itself a software component architecture shown on fig.7 . It allows:

- to plug the software component containing the model, and its associated sensitivity, in the optimizer.
- to plug two other kinds of software components as defined in the following.

The first kind of software component contains the optimization strategy. This allows immediately

connecting the model to an optimization strategy. Until now we have developed optimizer components that contain algorithms like: gradient Sequential Quadratic Programming optimization (SQP), genetic optimization, global deterministic optimization, Pareto approaches, branch and bound algorithms [11]… It should also be noticed that those optimization components can be recursively plugged to each other in order to quickly create hybrid algorithms like a genetic combined with a SQP algorithm and this again without any programming skills for the final user of the optimizer.

The second kind of component is a graphical component, for the drawing of the parameterized picture of the structure.

It is not the aim of this paper to gives details about those components. What is interesting to notice is that again they are implemented by using patterns, concepts and technologies of software components previously described.

This architecture allows generically connecting new kinds of optimization algorithms to existing models without any need of programming, and so we reach need 3°.

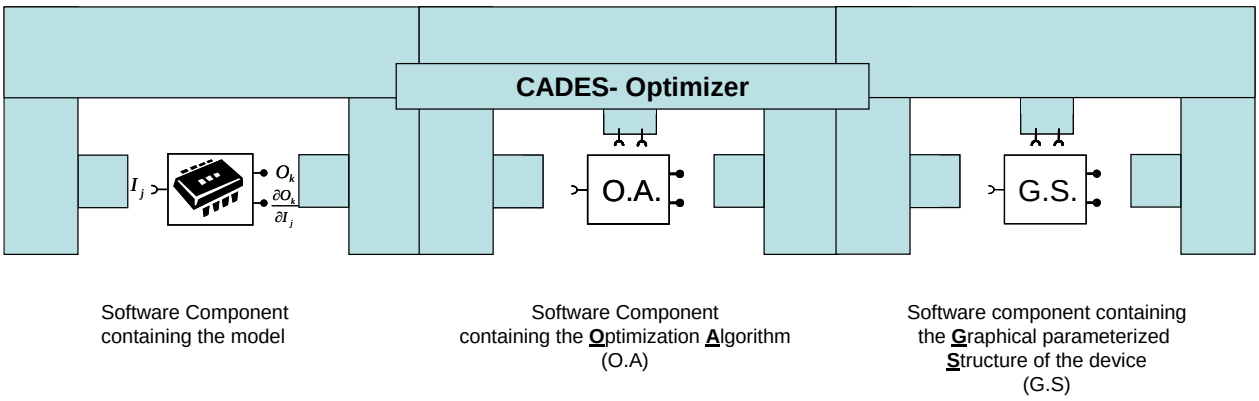


Fig. 7 – Structure of the optimizer: software components containing models are plugged together with software components containing optimization algorithms.

As shown on fig. 10, as soon as those components are connected in the optimizer, classical functionalities are available:

- Definition of constraints on input and output parameters (*constrained by interval, fixed…*).
- definition of the objective function (*minimize / maximize*).
- starting of the optimization.
- plots of the evolution of the parameters during the optimization.

- drawing of the evolution of the device structure during the optimization.

C. *Illustration of the CADES framework on an example*

The use of the framework is described here on the example of the sizing of a three phase power transformer. As shown on figure 8, the physical model of this transformer is described in the SML language in the Component Generator tool. This generator is then able to automatically generate the CADES component v1.0, which contains the model in an executable form. The component will also contain the formal right computation of the partial derivatives of the output of the component as a function of the inputs.

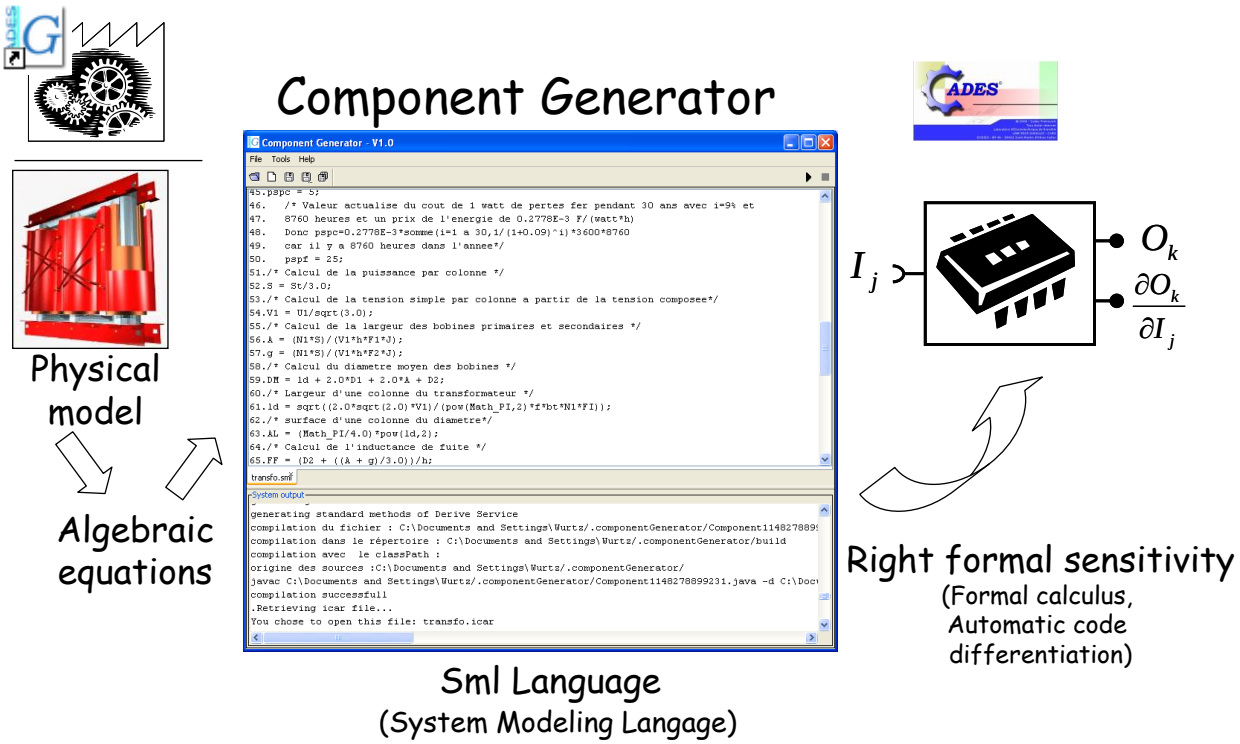


Fig. 8 – Use of the generator of the CADES framework: example of the optimization of a three phase power transformer.

After the software component has been generated, it can be plugged in a Component Calculator service which is the model simulation service. As shown on fig. 9, this service allows:

- to make computations, what means, give values to input parameters in order to compute output parameters.
- to draw plots of outputs values as functions of input values.
- to make sensitivity computation. You can especially see, on fig. 9, the computation of a jacobian

matrix.

This service has for goal to do as many as possible computations on the component, especially in order to verify its right behavior.

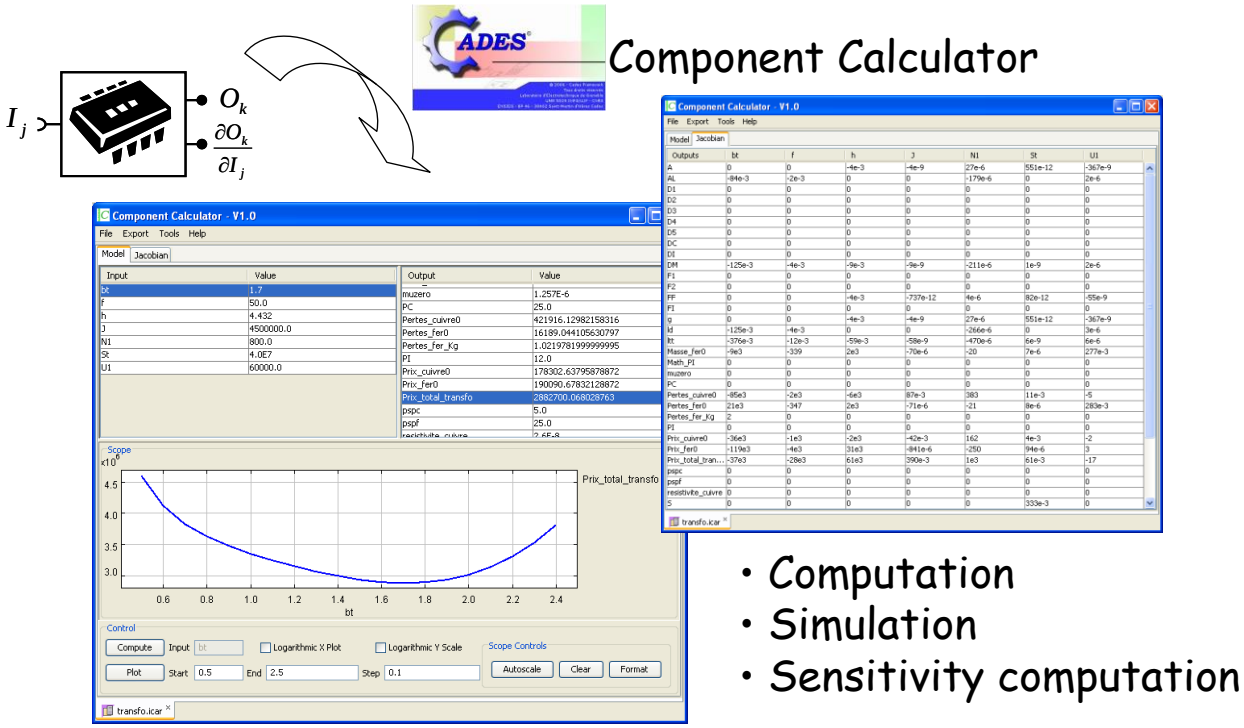


Fig. 9 –Use of the software component of the three phase transformer in the component Calculator.

After the software component generation, the most interesting is that it can be plugged into Component Optimizer. As shown on figure 10, this optimizer allows:

- to plug the software component containing the model, and its associated sensitivity.
- to plug also the two other kinds of software components that this optimizer is able to manage.

First kind of component is the software component containing the optimization strategy. As previously said, much kind of strategies is available. Here, on the example, a classical deterministic algorithm is used: Sequential Quadratic Programming approach (SQP). The software component encapsulates the SQP algorithm VF13, available in the Harwell Subroutine Library (See [12]), described in [13]. This algorithm is written in FORTRAN, but there were no technical difficulties to compile it and to integrate it in the optimization software component.

The second kind of component is the graphical component, for the drawing of the parameterized picture of the structure of the transformer.

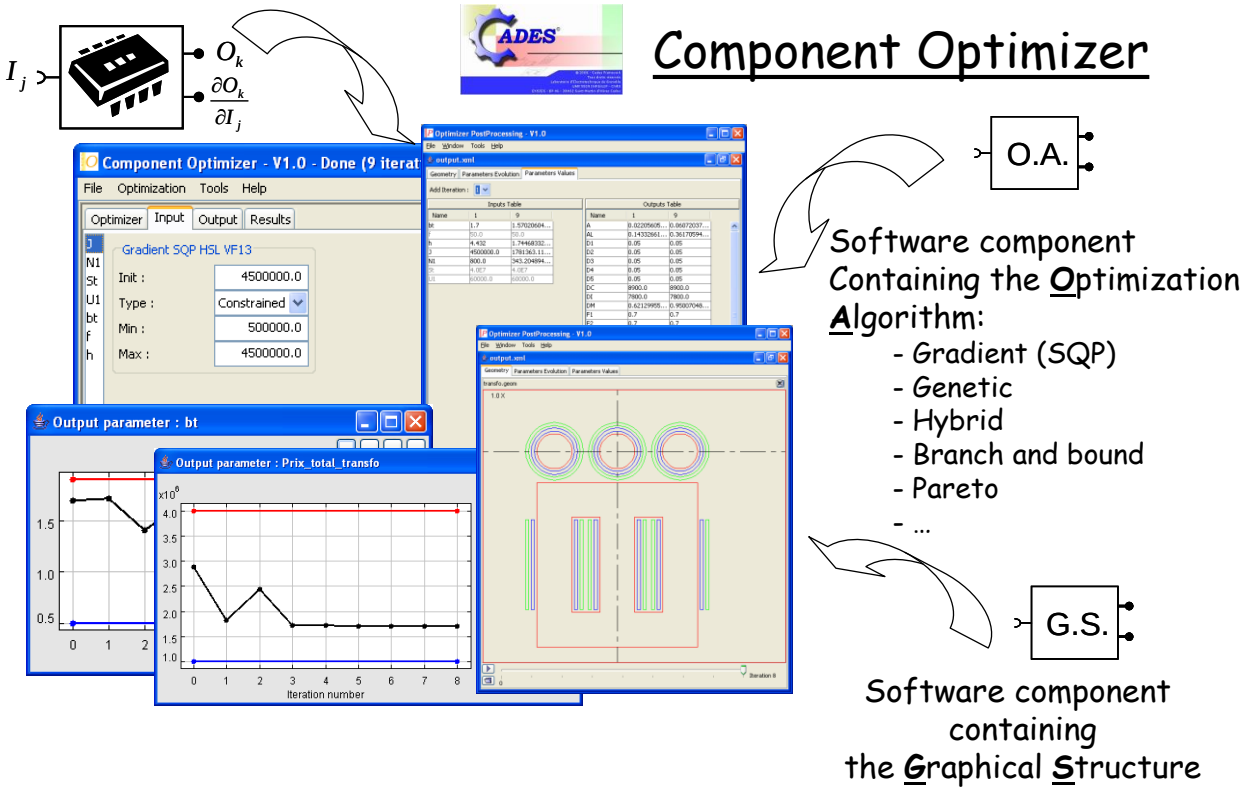


Fig.10 – Use of the software component of the three phase transformer in the component Optimizer.

IV. Perspectives of the framework

A. Community of users creating standardized software components of engineering components and systems

Since the software component can be standardized, especially those containing models of components and systems, they can be stored and reused in order to reach need 1° b. So our aim is this capitalization, but also the creation of community of users, that have in common to exchange the same standardized components. This should allow creating communities of users sharing knowledge contained in the software components, and their skills for creating those models. With these communities, models of the engineering components and systems for optimization can be found and used more quickly.

B. Implementation and extension of the software pattern

As defined in this paper, two kinds of pattern have been proposed. First was the CADES framework pattern which describes how generators, component and services interact. Second is ICAr component pattern which describes how Component and Facets interact.

Both are a way to extend the CADES framework for new needs. Thus, scientific or industrial people who

want to integrate modeling techniques can do it in a generator to produce components. Then they can take advantage of all available services (*simulation, optimization, composition …*). Moreover, services may also be developed like optimization services or tolerancing services to take advantage of existing generators but also to bring new design tools to designers with all existing and future community models.

If component v1.0 does not answer all requirements to develop specific pairs of generators/services, ICAr pattern can be used to extend the standard with new functionalities as it was already done with component visualization facet or temporal simulation facet. These extensions can be optional or required facets of component depending on which services are used. These extensions can be on facets functionalities (*computing, sensitivity, visualization, documentation, light computing / heavy computing…*) or on data types which are real number in Component v1.0. But a component may have for example two computation facets, one with input/output real number format and another with interval format to be used with specific global optimizer based on interval propagation [15].

V. Conclusions

Software component architecture is really a new paradigm in the implementation of optimization software. It offers new opportunities to quickly connect models to algorithms, through standardized interface. It allows easy combination and offers a way to capitalize and reuse pre-existing models as well as pre-existing algorithms. The CADES framework which has been presented in this paper is an opportunity for designers and scientists to improve design methodologies by developing generators, capitalizing model and modeling skills around a modeler community, and by developing new services.

VI. References

[1] Szypersky, C., “Component Software – Beyond Object-Oriented Programming”, Addison-Wesley, Reading, MA1998.

[2] Delinchant, B., Wurtz, F. and Fandino, J., "Mixing of FEM and Analytical Modeling for the Preliminary Design of a Transformer", in Rudnicki, M., and Wiak , S. (Eds), *Optimization and Inverse Problems in Electromagnetism*, OIPE'2002 (Optimization and Inverse Problems in Electromagnetism), Lotz, 12-14 September 2002, Kluwer Academic Publishers, Dordrecht, pp. 245-252.

[3] Allain, L., Gerbaud, L. and Van Der Schaeghe, C., “Capitalisation and treatment of models for the optimisation of electric drives”, in Rudnicki, M., and Wiak , S. (Eds), *Optimization and Inverse Problems in Electromagnetism*, OIPE'2002 (Optimization and Inverse Problems in Electromagnetism), Lotz, 12-14 September 2002, Kluwer Academic Publishers, Dordrecht, pp. 205-212.

[4] Duret, D., Gerbaud, L., du Peloux, B., Verdiere, F. and Rakotoarison, H.L., “A Generator of Software Components dedicated to the Optimization of Engineering Systems”, paper presented at OIPE'06 - 6th Workshop on Optimization and Inverse Problems in Electromagnetism, Sorrento, Italy, 13-15 September 2006.

[5] du Peloux, B., Gerbaud, L., Wurtz, F., Leconte, V. and Dorschner, F., "Automatic Generation of Sizing Static Models Based on Reluctance Networks for the Optimization of Electromagnetic Devices", IEEE Transactions on Magnetics, Volume 42, Issue 4, April 2006, pp. 715-718.

[6] Rakotoarison, H.L., Delinchant, B. and Cugat, O., “Methodology and tool for generating semi-analytical models used to pre-design electromagnetic MEMS (Mag-MEMS)”, paper presented at CEFC'06 - 12th Biennal IEEE Conference on electromagnetic Field Computation, Miami, USA, 30 April – 3 May, 2006.

[7] Wurtz, F., Delinchant, B., Nguyen Huu, H., Verdiere, F. and Bergeon, S., “An Optimizer using the Software Component Paradigm for the Optimization of Engineering Systems” , paper presented at OIPE'06 - 6th Workshop on Optimization and Inverse Problems in Electromagnetism, Sorrento, Italy, 13-15 September 2006.

[8] Delinchant, B., Wurtz, F., Magot, D. and Gerbaud, L., “A component-based framework for the composition of simulation software modeling electrical systems”, Journal of Simulation, Special Issue: Component-Based Modeling and Simulation. Jul 2004; vol. 80: pp 347 - 356.

[9] Fischer, V., Delinchant, B., Wurtz, F. and Gerbaud, L., “Using Software Components for Multi-disciplinary Optimization”, paper presented at CEFC'04 - 11th Biennal IEEE Conference on electromagnetic Field Computation, Seoul, South Korea, 06-09 June, 2004.

[10] Fischer, V. and Gerbaud, L., “CoreLab: a component-based integrated sizing environment”, COMPEL, The international journal for computation and mathematics in electrical and electronic engineering, vol 24, 2005, pp753-766.

[11] Magot, D., “Méthodes et outils logiciel d’aide au dimensionnement“, Application aux composants magnétiques et aux filtres passif“, thèse de l’Institut National Polytechnique de Grenoble, 28 septembre 2004.

[12] HSL (Harwell Subroutine Library), VF13 subroutine, available at http://www.cse.clrc.ac.uk/nag/hsl/hsl.shtml

[13] POWELL, M.J.D., "On the quadratic programming algorithm of Goldfarb and Idnani", Mathematical Programming Study 25 (1985), pp. 46-61.

[14] Duret, D., Gerbaud, L., du Peloux, B., Verdiere, F. and Rakotoarison , H.L., “A Generator of Software Components dedicated to the Optimization of Engineering Systems”, paper presented at OIPE'06 - 6th Workshop on Optimization and Inverse Problems in Electromagnetism, Sorrento, Italy, 13-15 September 2006.

[15] Delinchant, B., “Interval Arithmetic for the Design of Electrical Engineering Systems”, paper presented at OIPE'06 - 6th Workshop on Optimization and Inverse Problems in Electromagnetism, Sorrento, Italy, 13-15 September 2006.