Available online at www.sciencedirect.com
SCIENCE DIRECT®
PERGAMON
Computers and Structures 81 (2003) 1689–1701
Computers & Structures
www.elsevier.com/locate/compstruc

An object-oriented simulation–optimization interface

R. Le Riche [a,*], J. Gaudin [b], J. Besson [c]

[a] Ecole des Mines de St. Etienne/CNRS URA 1884, 158 cours Fauriel, 42023 St. Etienne Cedex, France
[b] EADS CCR, 12 rue Pasteur, 92152 Suresnes Cedex, France
[c] Ecole des Mines de Paris/CNRS UMR 7633, Centre des Matériaux, BP87, 91003 Evry Cedex, France

Received 16 May 2002; accepted 26 March 2003

Abstract

Progress in the field of structural optimization naturally leads to an increasing number of structural models and optimization algorithms that need to be considered for design. Software architecture is of central importance in the ability to account for the complex links tying new structural models and optimizers. An object-oriented programming pattern for interfacing simulation and optimization codes is described in this article. The concepts of optimization variable, criteria, optimizers and simulation environment are the building blocks of the pattern. The resulting interface is logical, flexible and extensive. It encompasses constrained single or multiple objective formulations with continuous, discrete or mixed design variables. Applications are given for composite laminate design.

Keywords: Object-oriented programming; Optimization; Composite design; Software engineering; Process optimization; Multiple objective optimization

1. Introduction

As research in the field of structural modelling and optimization progresses, design needs to account for an increasing number of interdependent models and allow diverse problem formulations. The architecture of the computer program that implements structural models and optimization algorithms has therefore become crucial in dealing with such increasing complexity. Deliberate programming is a stepping stone for effectively capitalizing knowledge so that creativity in design is not limited by technical book-keeping.

This article describes a central part of any structural optimization program, the interface between simulation and optimization sub-programs. The interface is presented as an object-oriented programming pattern. Unlike sequential programming languages (FORTRAN, C, BASIC, PASCAL,...), object-oriented languages like C++ [1,2] provide ways to express the solutions to a problem directly and concisely, by creating new relevant types and their functionalities (objects). Object-oriented programming patterns [3,4] go one step further: they describe how different objects work together to fulfill a task.

In the field of mechanics, object-oriented patterns have mainly covered finite element programming [5–8]. The need for object-oriented analysis in advanced optimization strategies has been clearly stated and solutions proposed in [9,10]. The text starts with a review of simulation–optimization interfaces. Next, a pattern for an internal interface is proposed, which consists of assigning responsibilities to new programming types related to optimization variables, criteria, algorithm and simulation environment. The resulting program is versatile as it allows constrained single or multiple objective formulations with continuous, discrete or mixed design variables. Applications are given for composite laminate design. A coupled process-structure problem and a coupled material selection-structure problem are solved.

* Corresponding author.
E-mail address: leriche@emse.fr (R. Le Riche).

0045-7949/03/$ - see front matter
doi:10.1016/S0045-7949(03)00186-X

2. Interfacing models and optimizers

Simulation programs and optimizers can be interfaced externally or internally.

2.1. External interfaces

The external interface is composed of at least two executable programs, the optimizer and the simulation. The optimizer calls the simulation (system call). Both modules communicate through files. It is illustrated in Fig. 1, where x denotes the design variables, f the objective function(s), and g the constraint(s). Such implementation is typical when optimization is not planned beforehand and the simulation source files cannot be modified. It also occurs when resorting to an optimization package (such as [10,11]) separated from the simulation software. It usually requires writing a translator sub-program. The translator changes the vector of design variables into an input file that is read by the simulation. Reciprocally, the translator may have to interprete simulation output files in terms of objective function(s) and constraint(s).

The advantage of external interfaces is that, except for the translator, optimizer and simulation implementations are fully decoupled. The external interface has one important drawback: the simulation is entirely repeated at each evaluation of the optimization criteria, including loading of data, even if the design variables that are changed do not require it. Suppose for example that the optimization aims at choosing the material of a mechanical part. It is a waste of computer resources to read the geometry of the part at each evaluation, since it has not changed. Moreover, optimizer and simulation should be able to exchange information other than x, f and g: it is possible that certain combinations of design variables translate into an impossible simulation (consider rigid body motion of a mechanical component for example). The translator can of course read optional messages from the simulation, but this will typically involve tedious file parsing. It is worth mentioning that some software using external interfaces propose versatile ergonomic translators. The DAKOTA program [10] provides object-oriented utility classes, called `IOFilter`, to implement the translator. Another example is given by the "template mechanism" of [12], sketched in Fig. 2. In the template mechanism, input files are copied to template files (by adding `.tmpl` extension), and numerical values that are design variables are changed to identifiers (a `?` followed by a string). The translator, at each evaluation, copies the template files, remove the `.tmpl` extension, and replaces the variables identifiers by a numerical value.

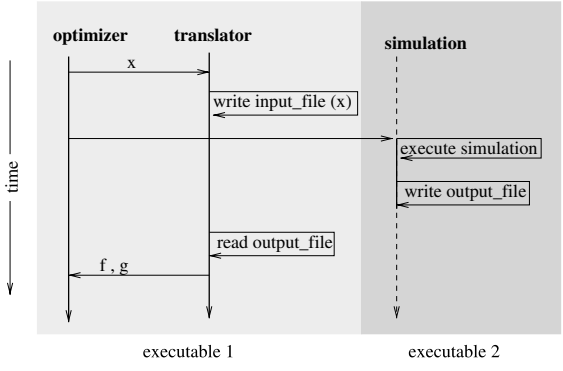

Fig. 1. External simulation–optimizer interface, one evaluation.

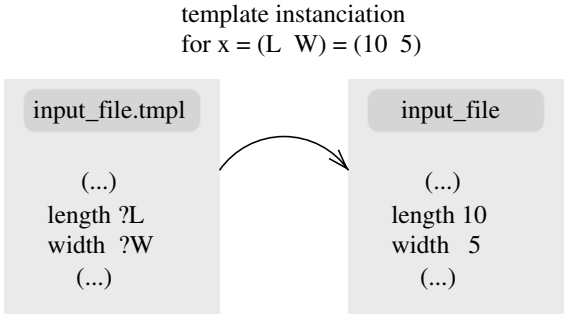

Fig. 2. Template instantiation as generic interfacing mechanism (from [12]).

2.2. Internal interfaces

Other implementations, where simulation and optimizer are embedded in the same program have internal interfaces. This is the case of the vast majority of analysis and optimization packages. With internal interfaces, design variables, optimization criteria (objective function and constraints) and optimization algorithms can typically be chosen from a pre-programmed list. For computer aided design software (e.g. [13,14]) geometrical parameters already exist and are used as design variables. Some software allows both continuous and discrete optimization variables [15]. Results of analyses, such as, in finite elements, displacements, stresses, eigenfrequencies, buckling loads,..., make up the list of possible optimization criteria. In some implementations [12,16], analytical functions of optimization criteria can define new optimization criteria. With internal interfaces, simulation data are put in memory once and for all and updated only when necessary. Great effort is often put into computing, analytically or semi-analytically, sensitivities of optimization criteria with respect to design variables for use with mathematical programming algorithms [17,16].