Concepts, Aims, and Problems which Modelling in OR and OOP Have in Common

Jörg R. Mühlbacher

Institute for Information Processing and Microprocessor Technology (FIM),
University of Linz, Austria

Preface

This paper is dedicated to Paul Stähly on the occasion of his 65th birthday. It reflects many private discussions between us, in particular, on modelling and the use of an appropriate programming language. Paul Stähly has always been a dedicated user of SIMULA 67, already in those days when object oriented programming (OOP) was not invented yet. He has early noticed the power of object orientation in his research area (OR). The authors of SIMULA 67 introduced classes as templates which define data structures and algorithms to be included in a particular instance (of a model). In addition, classes may be defined in terms of each other.

Paul Stähly has also been and still is aware of the problem to which extent the results of a calculation process can be interpreted semantically correct, if the model is used for a different application area as it was designed for originally. In component oriented software development this is referred to as "domain dependency", and there is a demand for a "contract" which may be understood as additional semantic information for the user of a model and its implementation.

1 Aims of modelling

Let us briefly recall a couple of major reasons why modelling is essential. In this context we have in mind applied disciplines as operations research and software
development. In order to focus on central aspects, we use examples which are well known and used in such areas.

*Abstraction to master complexity*

Talking about models we have to restrict to abstract ones in which physical situations or alike are described in symbolic form, and in this case we often make use of the language of mathematics (mathematical expressions, systems of equations, graphs, ...) or - with respect to computer science (CS) - we use notations and languages which have been developed for CS.

The most obvious purpose of models is: they are designed by an abstraction process in order to avoid too much complexity and to allow us to master the complexity which still remains after this process of abstraction. Using “complexity” we refer generally to both: time complexity as it is defined in the theory of algorithms, and complexity meaning interactions between components the model consists of.

A simple example is the calculation, whether a path from a source $s_0$ to destinations $s_i$ ($i = 1, 2, ..., n-1$) exists. The abstraction process, step by step, starts at a map of the region in question (of course, the map already is a model of the landscape), omits geographical orientation and finally ends up in a sparse matrix $A$, where

\[
\text{if a connection between } s_i, s_j \text{ exists then } A[i, j] = 1 \text{ else } A[i, j] = 0 \text{ end.}
\]

Whether, in terms of programming, the matrix $A$ as model of the resulting digraph is represented as an array or as a list of lists of neighbours (adjacency list) is a question of convenient representation and depends on the programming language used for the implementation. At this stage the representation has no influence on what the model represents and which kind of information we can gather from it.

*Reusability of models and their associated algorithms*

Another purpose of abstract models is the aim to reuse them. We can apply identical algorithms whenever we have brought down a task to a model which is “isomorphic” to a model having been derived from another problem. With “isomorphic” we refer to the fact that sometimes a transformation might be necessary in addition.

At this point the important aspect is: we have omitted details during the abstraction process.