What's Your Problem? The Problem of Problem Definition

J. Christopher Beck and Michael Gruninger

Department of Mechanical & Industrial Engineering
University of Toronto
{jcb,gruninger}@mie.utoronto.ca

Abstract. The first step in developing an application to solve a real world problem is to define the problem. Typically in applied mathematics, artificial intelligence, and operations research, the definition process generates a well-defined problem that is subsequently studied and, if the project is successful, solved (for some appropriate definition of "solved"). Our thesis is that problem definition is inherently a process of abstraction, reformulation, and approximation that has not been deeply studied in the literature.[1]

1 A Definition of Problem Definition

Consultants and applied researchers are often faced with real-world objects: a chocolate-bar factory or a set of operating theatres in a hospital to schedule, a classroom timetable at a university to develop. For all but the most trivial such problems, the consultant is faced with deciding what aspects of the situation should be included in a problem definition. This is a critical set of decisions because the problem definition has a great influence on the computational complexity of the eventual mathematical model of the problem as well as the extent to which a solution to the problem is actually useful in the real world.

The problem definition process is inherently one of abstraction, approximation, and, because it is likely to be iterative, reformulation. Can a production facility ignore the variance in activity durations? Can it account for machine breakdown by only scheduling machines at 80% capacity? Can we ignore tooling, the scheduling of human operators, upstream and downstream facilities? These issues are difficult to resolve because the impact of the decisions are inter-dependent and may become known much later in the development process (or never). Typically, it comes down to experience and, perhaps, small-scale experiments with prototype systems.

Problem Definition $\neq$ Mathematical Modeling. The problem of problem definition as sketched above is not the same as the "modeling problem" that has been extensively studied in constraint programming and operations research. Modeling begins with a complete problem definition and develops and compares different mathematical models to solve the problem so defined.

[1] This paper is less a research summary and more a description of what we think is an interesting research direction. References to relevant work are solicited.

I. Miguel and W. Ruml (Eds.): SARA 2007, LNAI 4612, pp. 393–394, 2007.

2 Three Orthogonal Research Directions

An Experimental System. We do not know very much about the impact of problem definition decisions on application success or failure. One, low-level direction, is to develop a detailed simulation model of the real object such as a factory and use this as the "real world" for experimentation.[2] Given the simulation model, we could then develop different problem definitions at different levels of abstraction and, crucially, evaluate the impact of the levels of abstraction by solving the optimization models and "executing" the solutions in the simulation model. The primary question to investigate is the trade-off between the quality of the executed solution vs. the computational complexity of finding it and how the problem definition affects this trade-off.

Ontologies, Domain Modeling, and Problem Definition. An ontology is a set of reusable, sharable logical definitions that can be used for knowledge representation. With an ontology, such as Process Specification Language (PSL) [1], it is possible to build a logical domain model of a given real-world problem. Nevertheless, the specification of a domain model is not easy; in particular, the problem definition must be addressed: at what level of abstraction should the various real-world entities be represented? The answer, of course, is: at whatever level is sufficient for the application. Sufficiency, however, must be judged by the outcome of the overall system. And so we are left, again, to rely on judgment and experience.

However, an empirical approach may be possible. Imagine a set of applications in a given area, such as production scheduling, that have both software systems and accompanying domain models. Based on differences (and changes during development) in the software systems, the corresponding models, and the corresponding outcomes, we may be able to begin to develop a predictive or advisory meta-system. Given a domain model of a new production scheduling problem, the meta-system, by reasoning about existing models and outcomes, may be able to provide predictions about performance outcomes, advice on abstractions or refinements that might be useful, recommendations for optimization technology, and "what-if" analysis.

A Methodology for System Engineering. Finally, given that the problem definition problem appears AI-complete, it is likely to be a human activity for the foreseeable future. The abstraction, approximation, and reformulation that is inherent in defining and building software systems in general would seem to apply equally to optimization systems.[3] Building on software engineering, perhaps a methodology for system engineering can be developed, incorporating ontologies and simulation prototypes.

References

1. Gruninger, M.: Ontology of the process specification language. In: Staab, S., Studer, R. (ed.) Handbook of Ontologies, pp. 575–592 (2003)

[2] The simulation model also requires a problem definition process. The hope is that it will be more detailed and refined than any optimization model we would investigate.

[3] Is building an optimization system different than building any other software system?