A Structured Modeling Technology*

Marek Makowski

International Institute for Applied Systems Analysis,
Laxenburg, A-2361 Austria
`http://www.iiasa.ac.at/~marek`
`e-mail:marek@iiasa.ac.at`

Abstract

This paper presents the methodological background and implementation of a structured modeling environment developed to meet the requirements of modeling activities undertaken to support intergovernmental negotiations aimed at improving European air quality. Although the motivation for the reported work came from the actual complex application presented in the paper, the actual scope of the paper covers a wide range of issues related to model-based decision-making support. The paper starts with a summary of the context of modeling composed of: the role of models in decision-making support; modeling paradigms; and state-of-the-art aspects of modeling complex problems. The modeling process is then characterized, and the requirement analysis for implementation of structured modeling is specified. The main part of the paper presents the structured modeling technology which was developed to support the implementation of the structured modeling principles for modeling complex problems.

Keywords: structured modeling, decision support systems, modeling systems and languages, model management, simulation, large-scale optimization, multiple-criterion optimization, database management systems, object-oriented programming, Internet, environment.

Contents

1   Introduction

Success comes from making the right decision at the right time. Rational decisions need to be based on a combination of knowledge, preferences, and intuition. Some of the knowledge pertinent to making correct decisions can be represented by mathematical models; these not only help to analyze the relations between the decisions and the resulting outcomes, but also facilitate identification of those decisions that correspond best to the preferences of the *Decision Maker* (DM). Desire for success is one of the oldest driving forces; thus modeling has a guaranteed future, provided it contributes more to success than to failure. There are countless publications documenting successful modeling stories, and many too, that present various modeling paradigms, techniques, and tools. Much less attention is paid to modeling failures and limitations. Clearly, a failure may be just a step on the road to success,[1] and limitations, when recognized, drive the development of better methods and tools.

    This article presents a new modeling technology which had to be developed to adequately address the needs of modeling a complex problem. In other words, the requirements of an actual application have driven the developments reported here. In fact, many of these requirements have already been formulated by Geoffrion (1987), who also proposed a rigid methodology called *Structured Modeling* (SM). Over the last two decades several hundred papers presenting various elements of SM methodology, prototypes of software tools, and SM applications have documented the developments in this methodology. Unfortunately, there have never been enough resources to unify the results of many diversified (not only SM-related) modeling activities to permit the development of a modeling environment that meets the requirements formulated by Geoffrion (1987) and accepted by most modeling practitioners. The results of

---

[1]In fact every successful modeler can confirm by own experience that *Success is the ability to go from one failure to another with no loss of enthusiasm* (W. Churchill).

various modeling activities have, however, provided a good basis for the development of the modeling technology presented here.

This paper outlines a complex modeling application developed to support international negotiations, a process that requires various types of analyses of a complex model, as well as interaction with its users. Both these aspects implicitly determine a number of modeling requirements pertinent not only to this case. The first set of requirements is in the domain of the developers, who design and implement a mathematical model, the relevant databases, and the software tools used for various types of model analyses. The second involves making the model specification acceptable to the users as well as making the results of the various analyses comprehensible to them. Both these domains are closely related. However, because of space limitations and the complexity of the problems related to the second domain, this paper focuses only on the issues pertaining to the first domain.

The main purpose of this paper is to address the basic problems faced by modelers, which can be summarized by a single question: how can the modeling legacy be exploited to provide the best possible model-based decision-making support within time and budget constraints? The paper thus has a much wider scope than just presenting a new modeling technology and a complex application. Although the modeling issues discussed here have been driven by the needs of a particular application, the paper covers a wide range of issues related to model-based decision-making support. The modeling technology presented here does not cover all these issues, but it is nevertheless applicable to a wide class of complex problems that are adequately represented by algebraic models.

The paper aims to contribute to mutual understanding between three communities working in the areas of: (1) decision analysis and support; (2) modeling and mathematical programming; and (3) applications of modeling to support solutions of complex problems. The paper therefore includes an overview of diversified paradigms of model-based decision-making support, discusses in more detail requirements for modeling environments for this type of support, and outlines a complex application that illustrates the need for a qualitative improvement in modeling technology. Addressing, as it does, a multidisciplinary audience the paper includes information that will already be known by readers familiar with the corresponding literature and/or practice; it is hoped, however, that every reader will find elements that are new and interesting. The paper contains an extensive list of references that aim to provide pointers for further reading for those new to some of the presented concepts who may therefore find the presentation too sketchy.

The paper is organized in the following way. Section 2 summarizes the role and structure of models applicable to decision-making support, discusses the issue of modeling paradigms, provides an illustration of a complex model by presenting the relevant characteristics of the RAINS model, and summarizes the state-of-the-art of modeling complex problems. Section 3 presents the elements of structured modeling, and the requirement analysis for modeling environments supporting structured modeling. The main part of the paper, presented in Section 4, is composed of an overview of the implemented approach, followed by a presentation of the common basis for the components of a structured modeling process, all of which are then discussed in detail. Programmer-oriented details of the implementation presented in Section 4 are discussed in Appendix A.

2   Context

The term *modeling* is used in various contexts and different types of modeling methods are widely used; for example, in simulation models (e.g., Modelica, Ptolemy II), in modeling ap-

proaches at the macro-level (such as CIMOSA, PERA, GRAI), in languages for specifying and documenting object-oriented software (e.g., UML), for a quick and easy way of defining predictive models (PMML), for supporting continuous spatial modeling (SMML), and in many other recently developed markup languages based on XML.

Many commonly used models can be classified as *Algebraic Models* (AM). Following the Oxford Dictionary and a common understanding, we use the term Algebraic Model (AM) for a set of relations (such as equations or inequalities) between quantitative inputs (decisions) and outputs (performance indices) that measure consequences of implementation of decisions; further on the term *model* usually stands for *AM* unless specified otherwise. AMs are used for model-based *Decision Support Systems* (DSS) that make it possible to find solutions to real problems that are better than those that could be found without model-based problem analysis. The term *decision support* is typically associated with management and policy-making but in practice similar model-based activities are also being performed in almost all fields of industry and research. Thus, AMs are used in a wide range of application domains including (but not restricted to) planning problems in environmental systems analysis, telecommunications, logistics, transportation, finance, marketing, production, distribution, as well as in science, research and education, whenever decisions require various analyses of large amounts of data and/or complex relations. AMs have many common analytical features; thus modeling methods and tools developed for AMs can be useful in a wide range of application domains.

2.1   Role of models

Mathematical modeling for decision-making support is the process of creating, analyzing, and documenting a model, which is an abstract representation of a problem developed for finding a possibly best solution to a decision problem. The problem implicitly leads the modeling process because aiming for an all-purpose generic model would inevitably result in a model that was far too complex to be justified resource-wise. Consider, for example, modeling a cup of coffee. Very different models are suitable for studying various aspects, e.g., how something (sugar, cream) is dissolved in the cup's content, or under what conditions the cup might break from thermal stresses, or what shape of cup is most suitable for use in aircraft, or how a cup of coffee enhances different people's productivity. An attempt to develop a model to cover all these aspects, and represent all the accumulated knowledge on even such a simple topic would not be rational.

To define a purpose for modeling one needs to analyze if and how a model can contribute to finding a better solution that can be found without a model. Not only have many of the problems related to a cup of coffee been satisfactorily solved without modeling but, in everyday life, everybody solves many complex problems without even thinking of modeling them. Consider driving a car, for example. Each driver controls a car subconsciously, applying quite complex principles of adaptive control, often without even understanding the dynamics of the car. Control engineers could solve differential equations to optimize the way they drive a car, but they do not need to do so. Moreover, in congested traffic each driver almost constantly monitors the behavior of the other drivers and every few seconds subconsciously predicts their behavior, assessing the risk related to various combinations of the predicted behavior. Moreover, experienced drivers can simultaneously converse, listen to the radio, or even consider problems not related to driving. Given the complexity of this everyday activity, it is amazing how well (measured e.g., by the frequency of mistakes that leads to accidents) the problem of controlling cars is solved by drivers with very diversified backgrounds and experience. If every driver can do this, then one should ask why models may be better at solving problems that seem to be simpler.

The simplest answer to this question comes from the observation that it is possible to accu-

mulate enough knowledge and experience to solve complex problems, sometimes even without understanding all the underlying mechanisms. However, in other decision-making situations models are necessary: not only to support a decision-making process but also to enhance public understanding of problems and the proposed solutions. As this paper focuses on supporting decision making, we only briefly comment on the role of models in public information. By now it is commonly agreed that the provision of information is critical to public acceptance, and that in reality some commonly discussed problems are actually incorrectly understood. Selected issues of modeling for knowledge exchange are discussed by Wierzbicki and Makowski (2000). The relevance of this publication for policy making is illustrated e.g., by Sterman (2002), who points out that although the Kyoto Protocol is one of the most widely discussed topics, most people believe that stabilizing emissions at near current rates will stabilize the climate. Current debates (some accompanied by strikes) on pension system reforms in several European countries also clearly show a wide misunderstanding of the consequences of population structure dynamics on economies in general and on pension systems in particular. These, and many other problems, can also be explained to the public by adapting relevant models for use in presentations that the public can understand. Unfortunately, various models developed for policy-making problems use different assumptions, and often different sets of data; therefore a comparative analysis of their results can at best be done and understood by a small community of modelers. The need for public access to knowledge pertinent to policy-making will certainly grow, see e.g., (Haklay, 2003), who discusses access to environmental information; thus the role of models in public life will also grow accordingly. Multidisciplinary and interdisciplinary modeling will grow in importance for the next generation society, see e.g., (Drucker, 2001), for which a knowledge-based economy will become a major driving force for development. Models can represent knowledge as both synthesized and structured information, which can be verified by various groups of model users, see e.g., (Borst, 1997; Funke and Sebastian, 1996; Mannino, Greenberg and Hong, 1990; Makowski and Wierzbicki, 2003a).

Modeling for supporting decision making is one of the basic activities of the Operations Research (OR) community; thus there are numberless applications in this field. A good illustration of the scope of applications for environmental quality control is given by Greenberg (1995), who not only surveyed problems and approaches but provided an annotated bibliography containing 355 citations. During the last decade the number of applications has certainly grown considerably.

Despite the countless number of successful applications, there is also well-justified criticism of various critical aspects of modeling, e.g., by Ackoff (1967), Ackoff (1979), Chapman (1988), Makowski (2001), and Sterman (2002). The role of models in modern decision making that is shared by the author of this paper is discussed in detail by Wierzbicki, Makowski and Wessels (2000), who also present the methodology and tools for model-based decision-making support, and illustrate them with several applications to complex environmental policy-making problems. A more focused discussion of selected elements of modeling for decision support is provided by Makowski and Wierzbicki (2003b), which also includes an updated bibliography on modeling for decision support.

We summarize this discussion by pointing out that models represent those aspects of knowledge relevant to a decision-making process that can be used more efficiently in computerized form than in any other way. In many cases, however, the modeling process itself (which starts with structuring the modeled problem) is even more important than the final version of a model (and the end product of this process, which is a DSS into which the developed model is included). This issue is discussed in more detail in Section 3.1.

2.2 Models for decision-making support

A mathematical model describes the modeled problem by means of variables that are abstract representations of those elements of the problem that need to be considered in order to evaluate the consequences of implementation of a decision (typically represented by a vector composed of many variables). More precisely, such a model is typically developed using the following concepts:

- Decisions (inputs) x, which are controlled by the user;
- External decisions (inputs) z, which are not controlled by the user;
- Outcomes (outputs) y, used for measuring the consequences of implementation of inputs; and
- Relations between decisions x and z, and outcomes y; such relations are typically presented in the form: $y = F(x, z)$, where $F(\cdot)$ is a vector of functions.
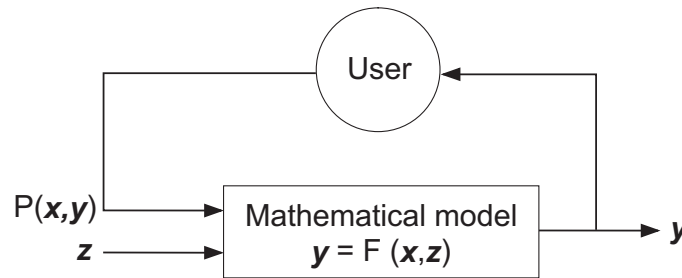

Figure 1: Structure of the use of a mathematical model for decision-making support.

A structure of the use of a model for decision-making support is illustrated in Figure 1. The basic function of a DSS is to support the user in finding values for his/her decision variables x that will result in a solution of the problem that best fits the preferences of the user.

A typical decision problem has an infinite number of solutions, and users are interested in those that correspond best to their preferences represented here by a *preferential structure* $P(x, y)$ of the user. A preferential structure takes different forms for different methods of model analysis, e.g., for:

- Classical simulation, it is composed of given values of input variables;
- Soft simulation, it is defined by desired values of decisions and by a measure of the distance between the actual and desired values of decisions;
- Single criterion optimization, it is defined by a selected goal function and by optional additional constraints for the other (than that selected as the goal function) outcome variables;
- Multicriteria model analysis, it is defined by an achievement scalarizing function, which represents the trade-offs between the criteria used for the evaluation of solutions.

A preferential structure typically induces partial ordering of solutions obtained for different combinations of values of inputs; in a well-organized modeling process it is not included in the model but is defined during the model analysis phase, when typically users substantially modify their preferences provided this is made easily available. In fact, a well-organized model analysis phase is composed of several stages, see e.g., (Makowski and Wierzbicki, 2003b), each serving different needs; thus, typically, not only are different forms of $P(\cdot)$ used for the same problem but also different instances of each of these forms are defined upon analysis of previously obtained solutions.

Such an approach to use models for supporting decision making differs substantially from the (traditional) OR routine of representing a decision problem as a mathematical programming problem in the form:

$$\hat{x} = \arg\min_{x \in X_0} \mathcal{F}(x), \tag{1}$$

which provides optimal decisions $\hat{\boldsymbol{x}}$. A concise formulation of (1) may be misleading for those who are unaware that solving a mathematical programming problem is frequently a challenging task. One should be aware of both the scientific values and the resources required to find and implement an algorithm able to provide a correct solution $\hat{\boldsymbol{x}}$ from a set X_0 that minimizes the objective $\mathcal{F}(\boldsymbol{x})$, and uses possibly small computer resources. In fact, various types of mathematical programming problems are typically defined during the analysis phase; thus, optimization continues to play a crucial role in model-based decision support. However, optimization in supporting decision making for solving complex problems has quite a different role from its function in some engineering applications (especially real-time control problems) or in very early implementations of OR for solving well-structured military or production planning problems. This point has already been made clearly e.g., by Ackoff (1979), and by Chapman (1992), who characterized the traditional way of using OR methods for solving problems as composed of the following five stages: describe the problem; formulate a model of the problem; solve the model; test the solution; and implement the solution. The shortcomings of such an approach are discussed in many other publications, some of which are overviewed by Wierzbicki et al. (2000). Here, we outline only one issue of more general interest related to widely used single-criterion optimization.

Large optimization problems have practically non-unique solutions. More exactly, they have many (typically, an infinite number of) very different solutions with almost the same value of the original goal function. Let us illustrate this issue by considering two solutions $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ such that:

$$|c(\boldsymbol{x_1}) - c(\boldsymbol{x_2})| < \epsilon \qquad ||\boldsymbol{x_1} - \boldsymbol{x_2}|| > \delta \qquad (2)$$

where $c(\cdot)$ is a goal function, $|| \cdot ||$ is a norm used for defining the distance between solutions, and ϵ, δ are two positive numbers, small and large, respectively. Although most large optimization problems have this property, its implications do not attract enough attention because analysts often look only at an optimal solution without analyzing other solutions that have practically the same value of the goal function. Typically, a problem gets noticed when various instances of the mathematical programming problem that differ very little have very different optimal solutions (while the value of the goal function remains practically the same). There is a simple and practical technique called regularization that provides a suboptimal solution with additional properties specified by a user. The methodological background of regularization is presented, e.g., by Makowski (1994), and its implementation in the RAINS model is discussed by Makowski (2001).

2.3 Modeling paradigms

A scientific paradigm, as defined by Kuhn (1970) and by Hloyningen-Huene (1993), embodies the consensus of a scientific community on an approach to a problem (or to a class of problems); a scientific paradigm consists of the theories, laws, rules, models, concepts, and definitions that are generally accepted in science. Because of the unquestionable success of modeling in problem solving, various modeling paradigms have been intensively developed over the last few decades. In this, to a great extent case-study-driven process, a growing tendency to focus on specific methodologies and tools was observed. As a result, different types of models (characterized by types of variables and relations between them) were developed (e.g., static, dynamic, continuous, discrete, deterministic, stochastic, set-membership, fuzzy, soft constraints) with a view to best representing different problems by a selected type of model. Moreover, different methods of model analysis (e.g., simulation, optimization, soft simulation, multicriteria model analysis) have been developed as the best-possible support for various types of model

analyses for different purposes and/or users. Finally, because of the growing complexity of various computational tasks, solvers have become more and more specialized, even for what was originally the same type of mathematical programming problem. The following publications provide a small sample of various successful attempts to develop modeling methods and apply them to various DSSs that, over the years, have resulted in recognized modeling paradigms: (Zadeh, 1965; Charnes and Cooper, 1967; Keeney and Raiffa, 1976; Wierzbicki, 1977; Zimmermann, 1978; Maclean, 1985; Sawaragi, Nakayama and Tanino, 1985; Tversky and Kahneman, 1985; Yu, 1985; Geoffrion, 1987; Rapoport, 1989; Sawaragi and Nakamori, 1991; Stewart, 1992; Wierzbicki, 1997; van Waveren, Groot, Scholten, van Geer, Wösten, Koeze and Noort, 1999; Wierzbicki et al., 2000; Nakamori and Sawaragi, 2000; Ruan, Kacprzyk and Fedrizzi, 2001; Carlsson and Fullér, 2002; Fink, 2002; Liu, 2002; Makowski and Wierzbicki, 2003b). Because of space considerations we cannot comment more specifically on particular modeling paradigms. Such comments, and a more extensive bibliography, can be found in e.g., Wierzbicki et al. (2000), and Makowski and Wierzbicki (2003b).

Each modeling paradigm embodies a great deal of accumulated knowledge, expertise, methodology, and modeling tools specialized for solving various problems peculiar to each modeling paradigm. These resources, however, are fragmented, and using more than one paradigm for the problem at hand is too expensive and time-consuming in practice.

Multiparadigm modeling, defined as an efficient application of all pertinent modeling paradigms, is one of the key issues involved in modeling complex problems. In some situations it is possible to use a more general paradigm, which "includes" a simpler paradigm (e.g. building a nonlinear model, which contains a linear part. However, even in such a case it is typically rational to use nonlinear and linear paradigms to respective parts of such a model). Thus, instead of using a more general paradigm (e.g., one can formally treat a linear model as a nonlinear one), it is rational to use a *unifying paradigm*, e.g., one that supports nonlinear models with a (possibly large) linear part. In other situations, however, it is not practicable to unify different paradigms. In such situations one needs to *switch paradigms*. The main difference between switching paradigms (within a properly organized modeling process) and applying different paradigms "independently", lies in the appropriate handling of those modeling process elements that are common to different paradigms and in supporting a comparative analysis of the results obtained with the help of applied paradigms.

Geoffrion (1987) formulated the principles of structured modeling thus providing methodological framework for the integration of various paradigms. Moreover, Geoffrion (1989) discussed integration of various elements of modeling more explicitly. Unfortunately, the proposed integrating framework has been to a large extent ignored, and most modeling paradigms have been developed somewhat separately. All these developments have been rational from the point of view of providing more efficient solutions for specific types of models or elements of a modeling process. Nevertheless, efficiency requirements in developing specific methodologies and tools for each modeling paradigm have resulted in paradigm-specific problem representations as well as corresponding modeling tools for model specification, analysis, documentation, and maintenance.

Before discussing the modeling state-of-the-art, we need to focus the discussion on what type of problems we are dealing with. The complexity of problems, and the role of corresponding models in decision support are the two main factors that determine requirements for the type of modeling technology that differs substantially from the technologies successfully applied for modeling well-structured and relatively simple problems. In most publications that deal with modeling, small problems are used as an illustration of the modeling methods and tools presented. Often, these can also be applied to large problems. However, the complexity is characterized not primarily by the size, but rather by the structure of the problem and by the

requirements for the corresponding modeling process. Thus, in order to justify the technology presented in this paper, we need to present the real-world problem which cannot be modeled appropriately by commonly used modeling tools. The model presented implicitly defines the class of models dealt with by this paper, and it is also used for the illustration of modeling issues discussed here.

2.4   RAINS models

With a view to discussing various modeling problems later in this paper, we now present an outline of the RAINS model, which is used for supporting international negotiations aimed at improving European air quality, and is described in more detail by Amann and Makowski (2000).

RAINS provides a consistent framework for the analysis of cost-effective emission reduction strategies. The quality of air is assessed by several indicators computed at a few hundred grids, their values depending on the locations and the amounts of emissions of various pollutants. Hence, the decision (input) variables are emissions, and output variables (used for assessing consequences of decisions) are composed of the costs for reducing emissions and of a set of various air-quality indicators; each indicator is composed of vectors of the values of the indicator at each of the grids into which Europe is divided for the purpose of air-quality assessment. The original RAINS model, described by Alcamo, Shaw and Hordijk (1990), which was a small Linear Programming model (that dealt only with acidification), can be considered as a small pilot prototype of the current version of RAINS. The development of several versions of RAINS over more than ten years was driven by the needs of the negotiators. The first version of RAINS was used for negotiating the sulphur protocol; it therefore dealt only with a single pollutant. However, it has become clear that a multi-pollutant, multi-effect approach offers substantial environmental and financial advantages. To respond to these needs, RAINS has been extended and gradually modified to the version described by Schöpp, Amann, Cofała and Klimont (1999). In mathematical programming terms, this version of RAINS is a large (about 30,000 variables and over 30,000 constraints) nonlinear model which requires advanced techniques for model generation, preprocessing, and optimization-based analysis. The model specification and a more detailed discussion of applied modeling paradigms is provided by Makowski (2000).

The ever-growing needs of RAINS users require more detailed analyses of emission control options, and of more types of pollution. To address these needs, a new version of RAINS is being developed. Its structure is outlined in Figure 2. RAINS comprises now modules for emission generation (with databases on current and future economic activities, energy consumption levels, fuel characteristics, etc.), for emission control options and costs, for atmospheric dispersion of pollutants, and for environmental sensitivities (i.e., databases on critical loads).

The decision variables are composed of the levels of activities (consisting of technologies considered for relevant combinations of economic sectors and fuels); those, in turn, determine the levels of emissions of various types of air pollution by various sectors in each country; the activity levels of each technology imply the corresponding emission control policies and the related costs of implementation of these polices. Cost-effective measures can therefore be calculated by a minimization of the sum of costs related to reductions of all types of considered emissions in all activities in each country. To determine the corresponding environmental impact, emission levels are used as inputs to the several dispersion submodels and to the ozone formation submodel. Studies of the environmental impact of air pollution have resulted in the establishment of critical levels for various air-quality indicators. Consequently, exposure indices have been defined for each of the approximately 600 (or 5400 for more detailed analyses)
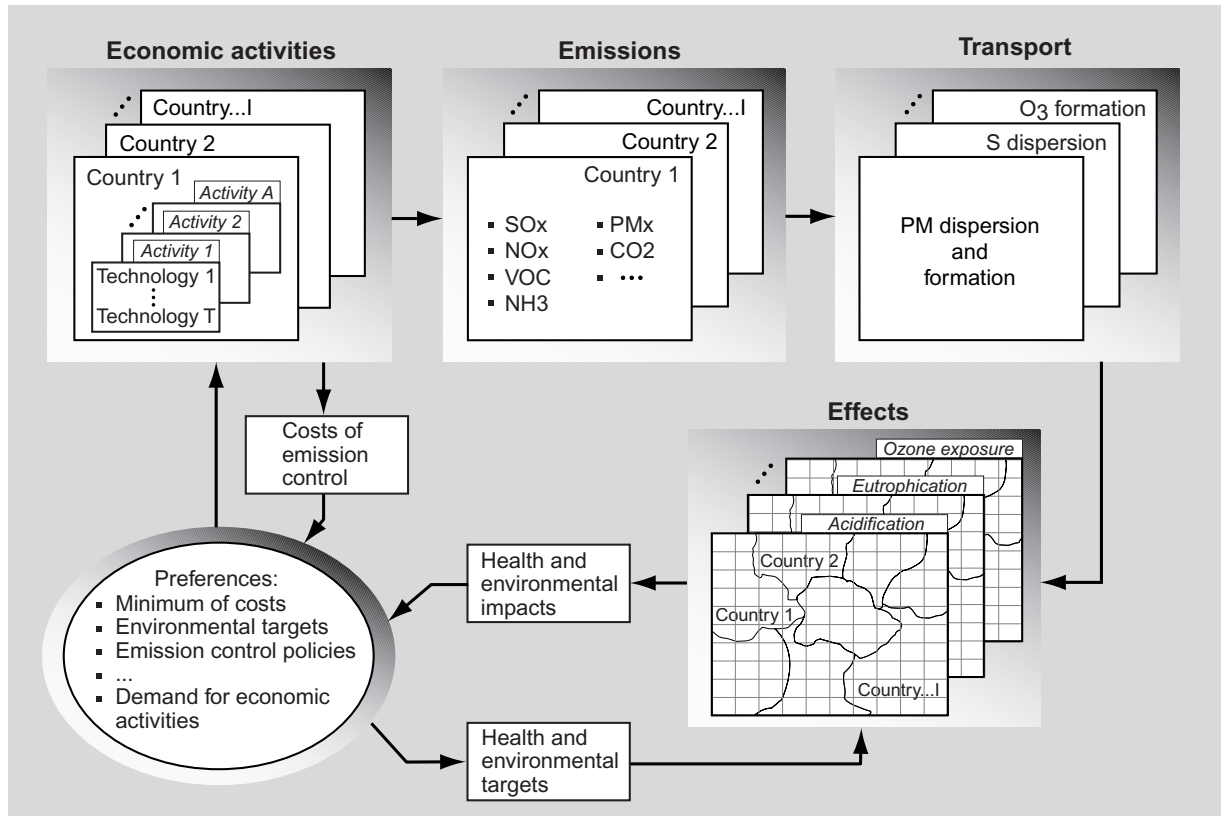
Figure 2: RAINS model structure.

grids in Europe (called also *receptors*).

We do not present here a specification of the currently developed version of RAINS because it would require a considerable amount of space. Although the model specification is indeed more complex than in the older version, the actual challenge is caused by the growing complexity of the definitions of compound variables, i.e., collections of primitive variables which are indexed so that they can be organized into structures (the issue of compound variables is discussed in detail in Section 4). Thus we characterize here only the indices used in the new version of the RAINS model.

The model uses the following indices which are organized in the corresponding sets:

- Index $i \in I$ corresponds to a country. The number of elements in I is about 50.
- Index $s \in S_i$ corresponds to a sector. The number of elements in S is about 30.
- Index $f \in F_i$ corresponds to a fuel. The number of elements in F is about 30.
- Index $a \in A_i$ corresponds to an activity (a *sector-fuel* combination). Therefore, elements of A_i are defined by a selection of pairs $\{s, f\}$ (all combinations are not necessarily allowed). The number of elements in A_i is up to (depending on the economy of i-th country) 400.
- Index $t \in T_a$ corresponds to a technology for a-th activity. The number of elements in T_a is between 6-8.
- Index $p \in P$ corresponds to a type of pollution. The number of elements in P is up to 7 (SO_x, NO_x, VOC, NH_3, PM_2, PM_x, CO_x).
- Index $j \in J$ corresponds to a receptor (which represents one of the grids into which Europe is divided for the assessment of the environmental impacts of air pollution). The number of elements in J is about 600, but can be increased to about 5400.
- Index $m \in M$ corresponds to a set of receptors, for which modified requirements on environment quality are analyzed. The number of elements in M is typically a fraction of the number

of elements in J.

- Index $l \in L$ corresponds to an indicator of environment quality. The set L may have up to 10 elements.

   The definitions of these sets are done implicitly during definitions of the various instances of the model (see Sections 3.1 and 4.5) by selections of corresponding data sets. Additionally, there are subsets of activities (and corresponding indices) that are defined by testing conditions corresponding to the fulfillment of binding agreements. Such conditions are defined as constraints in a symbolic model specification; they are instantiated from the data selected for each instance of the model and then tested during the instance definition, the results of the tests defining the corresponding subsets of activities.

   The RAINS model is used not only by analysts supporting information needed for negotiations, but also by many other users in industry and various organizations. The model is therefore tested not only by the developers, but also by a large community of users performing different types of analyses. Moreover, various users apply various sets of data for their analysis, and they also typically modify the data provided by the developers of RAINS. Such a wide and diversified way of exploiting the model requires transparency of the model specification and of the data used. There is also a demand for analysis of submodels (parts of the model adapted for analysis of more specific issues). Finally, we should point out that the data used for computations of the parameters of the model come from different sources, including the results of analysis of outputs from other models (see Section 4.4 for details). The diversified groups of model users and their needs, combined with the complexity of data handling, have imposed much more demanding requirements (outlined in Section 3.2) on the applied modeling technology.

   The new version of RAINS has been the main driving force for the development of the modeling technology presented in this paper. Developers of other complex models, however, face similar problems. Thus, the approach presented is likely to be interesting for the OR community.

2.5   The modeling state-of-the-art

The modeling needs of AMs are supported by general-purpose modeling environments such as GAMS, AMPL, AIMMS, MPL, and object-oriented modeling systems (e.g., ASCEND). Expertise and tools have also been developed with a focus on various modeling paradigms, either specific for a preferred type of analysis (e.g., optimization- or simulation-based) or specialized for a type of problem, e.g., PDI (Production, Distribution and Inventory Planning). These tools have been developed over the years and will continue to be developed and used for applications that can be adequately supported by a corresponding modeling paradigm. However, there are problems, and the corresponding models (such as RAINS) that demand modeling technology that cannot be provided by general-purpose tools. Typically, the development of complex models can be afforded only if it is based on modular, reusable tools. A comprehensive and integrated analysis of such models requires the application of various modeling paradigms. As a consequence of the long-term development outlined in Section 2.3, however, it has become increasingly difficult to apply all the pertinent paradigms to a problem at hand because corresponding resources are fragmented and using more than one paradigm for a problem is expensive and time-consuming in practice. Models have become complex and/or large; therefore the development and use of a model with even one specific paradigm is a costly and time-consuming process. Moreover, incompatible model representations used by different paradigms imply that resources used for modeling with one paradigm can hardly be reused when another paradigm is applied to the same problem. Multi-paradigm modeling support is thus currently one of the most challenging problems faced not only by the OR community but by a

broader audience of computerized decision- and design-support practitioners or even generally by the scientific community. Various modeling paradigms could be used much more broadly if scientific communities could get a better understanding of methodologies developed by other communities, and if modeling resources (model specifications, data, modeling tools) would be available in a uniform representation.

Low productivity of model-based work compared with high productivity of data-based work has already been discussed by Geoffrion (1987). In the case of databases, DBMSs are mature and well-established, and there is a broad agreement on the definitions of the abstract data models, as well as on the operations (e.g., those featured in SQL) to be supported for working with these data. This broad agreement has made it possible to efficiently use data from different sources because DBMS products of high quality are available and widely used. Despite the difference in the stage of development of DBMS compared with modeling, DBMS technology can be profitably used in modeling. It is therefore strange that professional-quality DBMS techniques are not routinely used (some of the most widely used algebraic modeling systems support specialized database links; however, none of them manages all persistent elements of the whole modeling process through a DBMS) in most modeling systems although it is generally agreed that dealing properly with models of a realistic size requires the use of modern DBMS technology, which has advanced immensely and is now well integrated with the Web.

Continuous progress in the foundations of modeling, and in database management, and new opportunities emerging from the network-based, platform-independent technologies offer a solid background for providing the desired modeling support needed for management, policy makers, research, and education. Arguments supporting this statement are summarized e.g., in (Dolk, 1988; ETAN Expert Working Group, 1999; Dolk, 2000; Cohen, Kelly and Medaglia, 2001; Geoffrion and Krishnan, 2001; Tsai, 2001; Dolan, Fourer, More and Manson, 2002). However, modeling technology is still at the stage where data-processing technology was before the development of DBMS. The data-management revolution occurred in response to severe problems with data reusability associated with file-processing approaches to application development. DBMSs make it possible to efficiently share not only databases but also tools and services for data analysis that are developed and supplied by various providers and made available on computer networks. Data processing was revolutionized by the transition from file processing (when data was stored in various forms and software for data processing had to be developed for each application) to DBMS. The need to share data resources resulted in the development of DBMSs that separate the data from the applications that use the data. The modeling world has not yet learned this lesson: almost every modeling paradigm still uses a specific format of model specification and data handling.

Most new modeling practitioners dealing with complex problems are often surprised by the amount of work and the length of time required to obtain truly useful results from model-based studies. The experienced modelers are familiar with principles of good modeling practice and aware of the fact that modeling skills are composed of knowledge, experience, art, and craft. Here, we only mention a sample of publications: Geoffrion (1989), Pidd (1999), van Waveren et al. (1999), Wierzbicki et al. (2000), Nakamori and Sawaragi (2000), Paczyński, Makowski and Wierzbicki (2000), Makowski and Wierzbicki (2003b), which cover selected issues of modeling practice. Most experienced practitioners are aware that in their modeling activities only a small fraction of available data, models, and modeling tools can be used because of, typically, limited resources (even if financial resources are not binding then time available for providing results is the limiting resource). At least seven factors – some of them summarized already by Geoffrion (1987) – are responsible for this situation:

1. Available modeling software typically caters to just one or two of the many phases of the total modeling cycle associated with model-based analysis and systems. Such phases can

be listed as follows: determination of requirements, design, building, testing, use, revision, maintenance, documentation, explanation, model analysis (which can again be subdivided into many types depending on the paradigm of the analysis used), reporting on findings, and the evolution of the model. Most practitioners are forced to work with a poorly integrated collection of tools to deal with these various phases which makes it difficult to follow principles of good modeling practice.

2. Support for model development, maintenance, documentation and analysis is very poor. Data handling issues (automatic documentation of data source, modifications, updates, using results from one model as parameters in another model, ownership, access rights) are especially critical for large-scale or complex models. Also poorly supported are: a version control of a model's symbolic specification, handling data needed for definitions of a model's parameters, model instantiation (defined by a selected specification and a corresponding set of data), and analysis of results (with type-specific views on various data). Finally, a comparative analysis of results (both for an instance of a model, and for a set of instances) are typically resource-demanding activities that need to be reimplemented for each model.

3. More than five distinct representations are used for each model: (i) a representation suitable for end users; (ii) a symbolic representation used by modelers; (iii) computer-executable representation of the model specification; (iv) a representation of the model instance; and (v) various representations of computational tasks in formats that are suitable for corresponding solvers. These representations (some of them also called *formats*) are typically incompatible (even for widely used and precisely defined types of mathematical programming problems) and are often inefficient because of their redundancy, possible inconsistency, and the demand for widely different requirements for handling them.

4. Modeling software uses different representations of model specification, and its diagnostic is typically limited to the syntax of a specification (semantics is therefore not diagnosed; thus a consistency in the units used for various entities of the model is not assured). Hence, even linking models (i.e. using output from one model as input to another model) is difficult and requires many resources. Integration of models (incorporating one into another), especially those developed in different application areas, is even more difficult. For these reasons, the rapidly growing modeling resources (models and data) available within an organization or as public resources on the Web cannot be used efficiently.

5. Interfacing models with advanced solvers (software for various computational tasks) used for model analysis is a laborious task requiring specialized skills, and the burden often falls on the end user. As solvers are specialized for a particular type of analysis, carrying out a comprehensive model analysis (which requires other solvers supporting different types of analysis) is even more difficult. This typically results in analysis being limited to only one type.

6. Support for multiparadigm modeling (as defined in Section 2.3) does not exist. Modeling software typically addresses just one of the many types of models or just one of the paradigms of model analysis. Therefore, changing the type of the model (e.g., replacing deterministic parameters by fuzzy or stochastic parameters or relations) or the type of model analysis typically requires changing modeling software. Thus, for a comprehensive analysis of results obtained from different modeling paradigms, additional resources are needed to convert both the assumptions and the results into a form that allows for comparisons.

7. Experience in and knowledge of the modeling process is to a large extent tacit knowledge. As modeling complexity increases, such support is becoming more and more essential, and even experienced modelers are facing problems with efficient use of available modeling resources. Knowledge-engineering methods, see e.g., (Borst, 1997; Carlsson and Walden, 1995; Funke and Sebastian, 1996; Liang, 1988), have not been applied yet to support efficient organization

of the modeling process.

The above summary of the state-of-the-art is widely recognized and shows that the available modeling technology cannot satisfy the rapidly growing needs for advanced modeling support that efficiently exploits the shareable knowledge contained in data, models and modeling tools. One needs to look no further than the optimization-focused modeling landscape, where a proliferation of tools exists, e.g., (Brooke, Kendrick and Meeraus, 1992; Greenberg, 1992; Drud, 1992; Fourer, Gay and Kernighan, 1993; Piela, McKelvey and Westerberg, 1993; Bhargava, Krishnan and Mueller, 1997; Wright, Worobetz, Kang, Mookerjee and Chandrasekharan, 1997; Kang, Wright, Chandrasekharan, Mookerjee and Worobetz, 1997; Bhargava and Krishnan, 1998; Wierzbicki et al., 2000; Fourer and Goux, 2001; Gondzio and Sarkissian, 2003; Fourer, Gay and Kernighan, 2003), nearly all of which are designed to operate in "stand-alone" mode. The ability of such tools to communicate with the broad range of programs and utilities needed to support the full modeling cycle is often marginal and typically requires either the use of broadly accepted (but not efficient) formats (e.g., MPS format for LP problems) or the development of interfaces to efficient but tool-specific formats (e.g., those used by solvers adapted for modeling languages). This artificially restricts the reusability of model-based work and, ultimately, the utility of the models themselves.

This summary by no means implies that there is no progress in modeling technology. On the contrary, a number of modeling languages and tools are constantly being improved to better address modeling needs. For example, many modeling systems do a good job in handling different model representations with little attention from human users; AIMMS (Bisschop and Roelofs, 2004) implements unit consistency; the structure of huge optimization problems is exploited by Gondzio and Sarkissian (2003) in distributed computing on a cluster of PCs. Moreover, Fourer, Lopes and Martin (2004) proposed a W3C schema for representing LP problem instances in XML, and provided an open-source C++ library to facilitate the exchange of information between modeling languages and solvers (this, however, is a very new development and it has not therefore been possible to evaluate if and how it can be exploited for the approach reported in this paper). All these developments, however, focus on needed but incremental improvements in what is rather traditional modeling technology. This is unlikely to result in the qualitative jump that is needed to meet the requirements summarized in Section 3.2.

For many applications one of the constantly improved modeling tools will continue to be a rational choice. For other, especially complex problems (such as RAINS), however, a new modeling technology is needed. Moreover, the two driving forces will most likely create enough incentive to force a breakthrough in modeling technology. First, modeling complex problems requires more and more resources, which can only be provided by collaborative work among large interdisciplinary teams. Second, adequate software technology that supports modular and extensible implementation of structured modeling is now in place. This paper shows how this technology can provide building blocks for the implementation of elements of a modeling environment for structured modeling.

3   Structured modeling

3.1   Modeling process

Modeling is a network of activities, often referred to as a *modeling cycle*, or a *modeling process*, or a *modeling lifecycle*. Geoffrion (1989) provides a detailed specification of a modeling cycle, together with references to earlier works on this topic. Here, we discuss the modeling cycle composed of more aggregated elements which correspond to the elements of the modeling

technology presented in Section 4.

Typically, such a process starts with an *analysis of the problem*, including the role of a model in the corresponding decision-making process. Subsequently, a conceptual (qualitative) version of a model is set up to support further discussions between modelers and users. In this phase a type (or alternative types) of mathematical representation(s) of the problem is/are decided; such a representation is composed of types of variables and of the mathematical relations between them. Next, such a conceptual model, together with an understanding of the problem, directs modelers to define a *model specification*. The latter is of a generic nature. It is composed of mathematical (symbolic) relations, and implemented using either a general-purpose modeling tool or by developing a problem-specific model generator. Different types of variables and relations are used depending not only on the kind of problem modeled but also on the choice of a model type that is relevant to its future use, available data, and resources for model development, analysis, and maintenance. For any non-trivial problem, model specification is an iterative process that involves a series of discussions between developers (typically OR specialists) and users until a common understanding of the problem and its model representation is agreed. Substantial changes of model specification are usually made during this process.

The most time-consuming element of a modeling process is *data collection and verification*. The data typically come from different sources (it is also often the result of analyses of other models); therefore, assembling the data and making it complete and consistent (e.g., defined in units consistent with the specification of model relations) is a resource-consuming process. Especially for large models, data management and documentation require a much more sophisticated approach than is commonly perceived. This issue is discussed in more detail in Section 4.4.

A *model instance* is defined by the model specification and a selection of data that define the parameters of its relations. During the model implementation many model instances are created and tested to verify that the symbolic model specification is properly implemented. Model instances differ by the various selections of data used for *instantiations* of the model specification, which typically correspond to various assumptions about the modeled problem. Typically, many instances of a model are used for different sets of data corresponding to various assumptions that the user wants to examine in order to check to what extent the model adequately represents the problem. An instance of the model is also called a *substantive model* because it represents relations between variables but does not include any preferential structure (arguments for this are presented in Section 2.2).

The next phase of the modeling process is *model analysis*. A typical decision problem has an infinite number of solutions, and users are interested in identifying and examining more closely a subset of solutions that correspond best to their preferences (including trade-offs between conflicting objectives), and to various assumptions that typically result in the selection of different sets of data defining model parameters. Therefore, a properly organized analysis of a model is the essence of any model-based problem support. Properly organized means that the user is supported in using all relevant methods of analysis, comparing the results, documenting the modeling process, and also in moving back to the first stage, whenever he/she wants to change the type of model (i.e., using a different type of variables and/or relations e.g., for handling uncertainty, or imprecision of model parameters). During the analysis of each instance of the model, different *computational tasks* are generated; each task is solved by a *solver* (a software tool specialized for specific types of mathematical programming problems). Thus, the model analysis is made up of two stages: first, various instances of the model are defined and analyzed; second, a comparative analysis of the results of various analyses of instances is performed.

3.2   Requirement analysis

A *modeling environment* (understood as networked modeling resources composed of models, data, software tools, and computing hardware) for support of structured modeling should have the following features:

1. Support the whole modeling cycle through knowledge-engineering methods adapted for managing and use of modeling resources (models, data, solvers), and for guiding a user through the whole modeling cycle while supporting a number of diversified modeling paradigms. Communicating multiple agents should be adapted to enable users' requests for the desired resources to be handled. Thus, agents that communicate with each other directly should provide access to modeling resources, in particular to applications dynamically composed of shared and geographically distributed modeling resources.

2. Integrate the whole modeling process with modern DBMS technology, thus providing a natural way to use proven technology (that is also efficient and familiar to many practitioners) for various modeling tasks.

3. Provide automatic generation of human-readable documentation of the whole modeling process, including different model representations, history of changes, data used, various views on results, and other functions desired for a comprehensive and well-documented model analysis that allows auditing.

4. Be based on an efficient, robust and structured representation of AMs that supports several advanced modeling paradigms. It needs to support the checking not only of the syntax of the model specification but also its semantics (including the correctness of dimensions and units) which is a prerequisite for coupling models developed in various application domains. It should require only a single-source model specification, thus assuring consistent specification, documentation, and actual implementation of the model. In particular, it should provide structured representation for the several types of AMs most commonly used, and be open for adding other types of models, which will inherit a large part of the already-implemented functionality that can be shared by several types of models.

5. Combine a de facto standardization of unifying model representation (around a rigorous, principled representational formalism of great generality) with an advanced system (which includes tools for model specification, editing, coupling and merging several models, and tools to setup, execute, evaluate, and monitor the modeling process), thus providing the functionality of an advanced modeling environment that also encourages good modeling practice.

6. Provide an open technology framework to allow access to already-developed models as well as to a wide range of modeling tools (supporting several leading modeling paradigms) to be available at distributed locations, and various hardware and software platforms. Moreover, it should provide an open environment to which new resources (models, data, and modeling tools) can be easily added, and should adopt the open-source principles (see `http://www. opensource.org`) that have commonly known advantages.

Thus, by pooling diversified paradigms, models, data, and modeling tools from various disciplines, by enabling work with heterogeneous hardware and software, such an environment should provide a common and efficient modeling platform for users from various disciplines and, in particular, users needing interdisciplinary modeling. While modeling in most disciplines is rather well developed, combining models between disciplines is much more difficult. Thus, an important aspect of the structured modeling environment should be its ease of combining interdisciplinary modeling activities.

A modeling environment that implements the features summarized above will certainly have an impact on the world of modeling similar to that of relational databases on the world of data management. However, the implementation of such an environment requires resources that

are not available for the work reported in this paper. Therefore, the choice has been made to implement only selected features of the proposed environment which does however make it possible to develop and exploit the new generation of the RAINS model by using the developed elements of the proposed modeling environment.

4   Modeling technology

Modeling technology is a craft of a systematic treatment of modeling tasks using a combination of pertinent elements of applied science, experience, intuition, and modeling resources, the latter being composed of knowledge encoded in models, data, and modeling tools. Thus the key to a successful modeling undertaking is defined by the appropriate choice of *"a combination of pertinent elements"*. The search for such keys has resulted in the development of various modeling paradigms (see Section 2.3) and underlying modeling technologies.

The *Structured Modeling Technology* (SMT) presented here is based on two successful paradigms: the Structured Modeling paradigm, which provides a proven methodological background, and the Object-Oriented Programming paradigm which, combined with DBMS, XML, and the Web technologies, provides an efficient and robust implementation framework. Moreover, the development of SMT has not only been driven by the actual needs of modeling a complex problem, but it has also been tested by complex applications. With this foundation, SMT is applicable to a wide range of complex algebraic models.

SMT is presented by discussing the following issues in sequence: overview of elements, common basis, details of each element, and integration of the elements into a modeling environment.

4.1   Divide, conquer, integrate

Divide and conquer is a proven tactic of generations of politicians and military. Nowadays, it is also widely used for constructive purposes in various areas, including computer sciences, e.g., as a basic principle of systems analysis, as one of the algorithm design paradigms, in object-oriented programming paradigms, to manage large collections of documents, and for designing computing architectures. This tactic is also a natural way of dealing with the modeling of complex problems, provided that *divide and conquer* is combined with a reintegration of the conquered pieces.

In other words, a structured approach for dealing with a complex problem is composed of three mutually linked elements: divide the problem into pieces, deal with each piece, and then integrate them. This advice is not always easy to follow, because the divided and conquered pieces may be difficult to integrate effectively (as is well known to those who are familiar e.g., with the decomposition algorithms, or aggregation of models). However, as is shown in this paper, this tactic can be successfully applied to modeling.

To succeed in any complex modeling undertaking, one needs to divide the modeling process into elements, and to provide a common basis that makes it possible to integrate these elements. We will first summarize the elements, then present the common basis, which will be followed by a more detailed discussion of the technology applied to each of the elements of the modeling process.

SMT supports modeling activities composed of the following elements:
1. Model (symbolic) specification consisting of:
   - A specification of compound variables and underlying sets; and

- A specification of algebraic relations between variables; such relations (also called functions), together with corresponding ranges of their values, are conventionally called constraints.

In the terms used in programming, the model specification is equivalent to a model declaration, i.e., it specifies types of variables, and the relations between them, but it does not define the values of parameters in these relations.

2. Collection of the data to be used for defining values of parameters specified in the symbolic relations.

3. A model instance defined by a selection of two objects: a model specification, and a set of data to be used for defining all sets of the compound variables, and all parameters in the constraints. A model instance is composed of a collection of objects, each corresponding either to a variable or to a constraint. Model instantiation is the process of definition of these objects, which are defined by two sets of data defining:

- Sets that in turn define compound variables and constraints; and
- Parameters of the constraints.

Note that the persistent representation of an instance is composed of declarations of the corresponding objects, i.e., there is no need to store copies of the data (because the data is stored in a Data Warehouse as described in Section 4.4).

4. Comprehensive analyses of selected model instances (discussed in Section 4.6).

5. Comparative analysis of the results of analysis of various model instances.

6. Documentation of the modeling process.

Before discussing each of these elements in more detail, we outline their common basis.

4.2 Common basis

SMT is built on the SM methodology (summarized in Section 3), and on standard paradigms and tools (referred to as *enabling key technologies*), which will now be characterized. Although these technologies are commonly used, their applicability to modeling is not commonly recognized. We will therefore characterize the key features of each technology that are exploited in SMT. One of these technologies is *Object-Oriented Programming* (OOP), which is widely used for software development, including some implementations of modeling systems. As OOP is especially useful for implementations of SM methodology, we will here outline basic structures (i.e., the data structures and the functions operating on them).

Thus, the common basis for all elements of SMT is composed of:

- A combination of standard computing technologies applied to all elements; and
- Basic structures (in the sense of OOP) used by more than one element of SMT.

We will now discuss these elements in sequence.

4.2.1 Enabling key technologies

SMT is built on commonly used paradigms and tools, including the Web, DBMS, XML, and OOP. Key features of these paradigms may not be known to all readers, however, therefore the role that each of them plays in SMT is outlined here.

The Web is by far the most commonly used platform for sharing computerized resources; it supports the pooling of diversified modeling resources (composed of models, data, modeling tools) available on heterogeneous hardware and software at distant locations, thus also enabling collaborative work across physical and disciplinary boundaries. Moreover, the fast development of the Web calls for its more advanced use, i.e., for jumping from passive access to distributed

information to interactive sharing of knowledge represented in models. The need for fast development in the advanced use of the Web is discussed e.g., by ETAN Expert Working Group (1999). Although today most existing modeling resources are fragmented and incompatible, there is no better alternative networking platform for modeling than the Web. The Web is used extensively for problem-specific decision-support systems, e.g., (Cohen et al., 2001), the RAINS model described in Section 2.4, and for many general-purpose computing tools, e.g., optimization servers reviewed by Fourer and Goux (2001). Finally, Web technology has become a standard, therefore the number of both experienced professionals and tools supporting their work is growing fast.

The role DBMSs play in modeling is outlined in Section 2.5. It is by now generally agreed that a proper dealing with models of a realistic size requires the use of modern DBMS technology, which also saves a great deal of resources needed for developing models with traditional techniques that typically do not use DBMSs. A novel feature of SMT is that it uses DBMSs not only for data but also for the whole modeling process outlined in Section 3.1. Integration of all key modeling tasks with elements of modern DBMS technology and with the Web provides a natural way of using these technologies that are proven and familiar to many practitioners, and thus integrates modeling activities with the management of data used for models while facilitating separation of model development and use, and of model specification and data. Moreover, the use of modern DBMS technologies provides fast response times and efficient data management through the Web.

SMT is being implemented for two widely used DBMSs, namely Oracle (which was used for the first version of SMT), and PostgreSQL (to which SMT will be ported in the near future). There are three reasons for this decision. First, an implementation for more than one DBMS provides a good basis for porting SMT to other DBMSs that conform to the SQL standard. Second, PostgreSQL is perhaps the most robust and capable of all the open-source databases, and is available on all commonly used platforms. Third, Oracle is the commercial DBMS used by the team developing the RAINS model.

While the Web and DBMS are already well-established technologies, XML is one of the recent developments aimed at enabling further expansion of these technologies into new domains of content management. Thus, XML opens new possibilities for modeling.

XML is a data format for storing structured and semi-structured text, originally designed for publications on a variety of media. However, it can be also used for self-documenting various types of information that is exchanged between applications. Therefore, the same XML-format content can be used e.g., for various formats of documentation (printed or displayed), and for various computer-readable representations of objects (model specification, data, model instances, results from the analysis) used in modeling activities. XML is actually a metalanguage that has no predefined list of elements; it is therefore possible to define elements as needed. Using an optional DTD (*Document Type Definition*) and XML Schema one can define the elements allowed in a particular type of document (e.g., containing a symbolic model specification, a collection of data, or a model instance). Many implementations of XML are available (browsers and authoring tools), many of which are open-source software. Moreover, XML documents can be managed by DBMSs, which is the final argument justifying the role of XML in SMT.

A number of XML-based modeling approaches has been developed recently, e.g., (Fourer et al., 2004; Kim, 2001; Lee and Neuendorffer, 2000; Srinivasan and Sundaram, 2000; Thorsteinsson, 1999). For the efficient handling of large and complex AMs, however, a new approach to using XML capabilities for handling data is necessary. Such an approach is summarized in Section 4.4. Here, we outline the applicability of XML to a model specification.

An XML document type can be defined for describing mathematics as a basis for computer-

to-computer communication. XML thus enables a single-source model symbolic specification that can be used for:

- High-quality documentation of the model specification that conforms also to the strict rules regarding the mathematical notation and even the layout of equations or formulas;
- Definition of the model instances;
- Definition of various forms of preferential structures;
- Generation of computational tasks representing a mathematical programming problem the solution of which fits best a selected representation of user preferences for a chosen model instance; and
- Processing solutions of computational tasks.

Using one source of a model specification for the whole modeling process is obviously critically important to assure consistency between a model specification, its documentation, and implementation.

The role and impact of OOP on SMT is essential, but difficult to present explicitly. The impact on the design and implementation of various elements of SMT is often implicit (e.g., on the definition of DTD for various elements of the modeling process). The most visible role OOP plays in the definition of the structures is presented in the next section.

4.2.2   Legacy of enabling technologies

Structure is probably the most fundamental concept for all key paradigms used for SMT. We summarize, in chronological order, the legacy of earlier concepts that is exploited in SMT, and then present the basic structures used in SMT.

Codd (1970) defined the so-called relational model, the first set of abstract principles for database management which provided a basis for the entire field of DBMSs. In the early and mid-1970s a variety of relational languages were developed to realize at least some of the features of the abstract relational model. One of them was SEQUEL (Structured English Query Language) developed at IBM in 1974 that was subsequently modified and renamed SQL (Structured Query Language) in 1977; this became the ANSI standard in 1986 and has been used by all widely used products for database management up to the present. While, in the DBMS world, the word *structure* is used explicitly only in SQL, structure is nevertheless probably the most commonly used concept for the basic elements of any DBMS, such as records, views, joints, etc. This is yet another argument for exploiting DBMS technology in modeling.

Geoffrion (1987) introduced *Structured Modeling* (SM), which is a way of dealing with analytic models and the computing environments that support modeling processes. The core concepts of SM defined by Geoffrion (1989) include sets, ordered pairs, relations, classification, grouping, hierarchy, primitive and compound entities, various types of elements (such as attribute, function, test), collections, generic and modular structures, as well as model instances and model structures. Most of these concepts are either directly used in or modified for SMT. In fact, a good understanding of the concepts defined for SM is essential for understanding any of the advanced modeling paradigms, even those that do not explicitly use these concepts. The *Structured Modeling Language* (SML) specified in (Geoffrion, 1992b; Geoffrion, 1992c) is not directly used for SMT; however, many of its elements have provided valuable guidance for SMT, something that will be clearly visible to readers familiar with SML.

The logical and physical structure of XML-based documents, and their representations used for document management through DBMSs and for their use through SMT, is of too technical a character to be discussed in this paper. Therefore, only descriptive summaries on the use of XML are provided with presentations of these elements of SMT, for which XML is used.

Object-oriented paradigms, such as object-oriented design and object-oriented system de-

velopment (see e.g., Booch (1991), de Champeaux, Lea and Faure (1993), respectively) are often used routinely by modelers who are also experienced in OOP, and this is also the case for SMT. The advantages of object-oriented approaches are far too well-known by practitioners to need repeating here. We only comment below on the two concepts of OOP, namely inheritance and encapsulation, which are extensively used in SMT.

Before presenting the structures used in SMT, in order to make the rest of this paper comprehensible to readers not familiar with C++, we recall here basic concepts of this language. From now on we will use the C++ semantic when discussing the issues directly related to SMT implementation.

Both structured modeling and OOP are based on the encapsulation paradigm, which clearly distinguishes between the specification and the implementation of an operation, and implicitly enforces modularity, which is necessary to structure complex applications. Encapsulation in OOP is based on abstract data types, in C++ called *classes*. A declaration of a class is composed of two parts:
- Data (sometimes called attributes); and
- Methods (functions, procedures) that operate on the data; these functions are of two types: *public*, which can be used from outside the object, and *private* which are known only within the class.

In addition to the classes provided by C++ (including the standard, and other libraries), users can define their own classes, thus creating new abstract data types.

There are two features of C++ which not only contribute greatly to its effectiveness in modeling, but also show similarities between OOP and structured modeling, namely *inheritance* and *templates*.

Inheritance is the mechanism that allows a class B (called *derived class*) to inherit the properties of a class A (called *base class*). Thus the objects of class B have access to the attributes and procedures of class A; thus there is no need to implement them in class B. However, if needed, then selected procedures can easily be redefined (specialized) for class B. For the illustration of inheritance, please see the discussion (in Appendix A) of the *entity* class, and classes derived from it.

Templates (formally called class templates) are generic data types. A class template looks like a "normal" class, except that some aspects are represented by place-holders, which are replaced by a class type when the template is used. Thus templates allow for design and implementation of a data structure and corresponding functions without specifying the details of the data. For example, the `mVector` class template (described in Appendix A) can be safely used for handling very different types of data (e.g., numbers, strings, and also complex data structures, such as model specifications, elements of documentation, etc). In other words, templates are parameterized classes that are implemented once but can be reused for various data types.

Inheritance and templates not only greatly improve the efficiency of the development of applications (because the amount of programming effort using C++ is dramatically smaller than needed for an implementation of the same application through any procedural language) but also their reliability (because the shared code is debugged much more carefully than is realistically possible for procedural programming). However, the most important argument is the fact that abstract data types can be defined in a way that naturally corresponds to the objects of structured modeling.

Objects of a given type (defined by a class) are created using special functions (called constructors), which take optional parameters that are used to provide other than initial default values of all data defining the object. In other words, objects are instances of abstract data (classes). The data part of a class is typically composed of objects of several types. Properly developed constructors create objects with a completely defined (by data items, each of which

is instantiated by its own constructor) state; and functions operating on data assure that data are kept in a correct state as long as the object is used.

Similarly, model instances are constructed from symbolic (abstract) model specification, and data used for definition of model parameters, and computational tasks are constructed from a model instance and a representation of a preferential structure.

This brief overview of C++ shows that the data structure of C++ applications can perfectly correspond to the elements of structured modeling. This is achieved by an appropriate design of hierarchies of classes, and by definitions of objects that correspond to elements of structured modeling. The rigid requirements of C++ assure that each object has to be completely created and that it is kept in an appropriate state for as long as it is needed. Computing resources allocated for objects that are no longer needed are automatically freed (using the mechanism of the destructors) and made available for other objects. Destructors of objects that need to be persistent (i.e., whose data need to be stored to be available for other applications) store the corresponding part of the object in a DBMS.

Summing up the overview of C++, we would point out that a formal presentation of C++ by Stroustrup (1997) contains excellent examples illustrating each element of C++, and the chapter on *Design and Programming* that is valuable reading also for modelers who avoid programming completely.

4.2.3 Basic SMT structures

After the summary of the legacy of paradigms that contribute to SMT, we present the core structures which are building blocks for SMT. These structures are embedded in applications and therefore are, of course, hidden from users, who use applications through the Web-based interface. However, the outline of the structures and of functions operating on them also illustrate modeling features that are not available in commonly used modeling tools.

In order to outline the role of OOP in the implementation of SMT we list here some of its basic classes, together with the data types, which each of them handles:
- `mSet` - various sets of indices, including functions defining a set by conditions;
- `mDict` - dictionaries (mappings between symbols [typically single letter] used in formulas, short names, and full names) used for indices of compound variables;
- `Entity` - primitive entities (parameters and variables);
- `Constr` - constraints (relations between variables);
- `mSpecs` - symbolic model specification;
- `mData` - meta data;
- `mInstance` - instance of the model;
- `mDoc` - documentation of the modeling process;
- `sModel` - management of all elements of a modeling process; and
- `mResources` - administration of resources (including access control).

More details about these classes are provided below together with a discussion of the corresponding elements of the structured modeling process. Moreover, a more technical description of other basic SMT classes is presented in Appendix A.

One of the characteristics of the RAINS model is its complex structure of indices outlined in Section 2.4. Therefore SMT has structures that handles compound variables and associated indices in an efficient and flexible way. The best (known to the author) summary of the role of indexing is given by Geoffrion (1992a): *"Indexing structures are of fundamental importance to modeling . . . as a device for mathematical abstraction, and because they facilitate conciseness, stability, and error-resistance"*. SMT exploits several of the ideas presented in this paper in implementations of various structured collections of entities, most of which require several

indices, sets of which are often mutually depended. As outlined in the illustrative definition of `R_MATRIX` in Appendix A, nested vectors can handle such collections. While a conventional matrix operates on two indices only, the presented structure can be used for handling a larger number of indices. Moreover, a conceptually similar approach is implemented to handle sparse collections, i.e., collections that have elements defined only for some combinations of indices. Sparse matrices are commonly used for large mathematical programming problems, therefore sparse collections of SMT can be interpreted as a generalization of sparse matrices. The design of such collections is conceptually simple; their efficient and robust implementation is more complex but as collections are commonly and heavily used in OOP, there is enough experience to do it right. However, an appropriate use of large multidimensional collections requires the support of classes that handle indices properly. In SMT this is based on the two classes outlined above: `mSet` and `mDict`. Most of the instances of `mSet` are implicitly defined by users (by selecting subsets of indices) during the instantiation of a model. Some of them, however, are defined by the results of testing specified conditions (please see Section 2.4 for an illustration). The `mDict` class provides the functionality needed to translate/link various representations of an index. In practice, at least three types (descriptors) are used: a one-letter symbol (typically used in formulas), and two collections of strings corresponding to each of the defined values of the index: the first composed of short descriptions (typically, abbreviations used by experienced users, and by the model developers); the second, composed of full names.

With the main building blocks outlined, we can now characterize the main elements of SMT that support the structured modeling process.

4.3   Model specification

H.W. v. Goethe summarized the issue of common understanding of mathematical representations of descriptively formulated problems in one sentence: *"Mathematicians are like Frenchmen: whenever you say something to them, they translate it into their own language, and forthwith it is something entirely different."* This is precisely the challenge of symbolic model specification: how to adequately represent the decision problem perceived and formulated within the habitual domain[2] of decision makers into the precise world of mathematics. SMT deals with algebraic models, therefore algebraic representation (in the form of equations or functions, using a commonly accepted notation) of the relations between specified variables is the natural choice.

Model specification defines a collection of compound variables (of type `Variable`, see Appendix A), and a collection of symbolic relations (of type `Constr`) between the variables. The sets of indices (of type `mSet`) needed for instantiation of collections are only declared (they are defined later during instantiations of a model).

In other words, model specification provides parametric definitions of all variables and constraints, and is equivalent to a commonly used symbolic definition of a problem by a specification of variables and constraints in which all distinct collections of variables and constraints are declared. To illustrate this let us consider a simple example of a state equation of a control problem, which is usually presented in a matrix form as:

$$Ax + Bu = c, \tag{3}$$

---

[2]A fairly stable set of ways for thinking, evaluating, judging and making decisions. Yu (1990) presents all aspects of habitual domains: their foundations, expansions, dynamics and applications to various important problems in people's lives, including effective decision making. It is based on an integration of psychology, system science, management and common sense and wisdom.

where A and B are matrices of parameters, x and u are vectors of state and control variables, respectively, and c is a vector of parameters. Equation (3) can also be presented in an equivalent, more detailed way as:

$$\sum_{i \in I} a_{ij} x_i + \sum_{k \in K} b_{kj} u_k = c_j, \qquad j \in J \tag{4}$$

where I, K, J are sets of indices (of the state variables, control variables, and state equations, respectively).

For this problem the model specification is composed of:
- Two vectors of variables x, and u, respectively. Each vector is composed of objects of the class `Variable`;
- Two matrices, A and B, and vector c, each composed of objects of the class `Param`;
- One object of type `Constr` that represents eq. (3) or eq. (4); and
- Three objects of type `mSet`, that will be defined during instantiations of the model for actual definitions of indices.

Thus a specification contains the full symbolic (i.e., except the values) characteristics of all the variables and parameters of the model, and the definitions of relations between them. This includes types (both of mathematical programming [e.g., real, integer, binary] and corresponding to the role of the variable in the model [e.g., decision, outcome, auxiliary]) of variables. For all variables and parameters their respective defaults (i.e., the values that will become actual, if not overwritten during the model instantiation) lower and upper bounds, and zero tolerance are defined.

One of the commonly used characteristics of a model specification is its Jacobian. More precisely, left-hand sides of constraints are formally treated as functions. If one denotes such functions by $g(x)$, where vector x is composed of all variables of the model, then the constraints of the model specification can be presented as:

$$l \le g(x) \le u, \tag{5}$$

where l and u are vectors composed of lower and upper bounds for the corresponding function. This general formulation covers all three commonly used types of constraints. Elements of the Jacobian matrix $J(x)$ are then defined in the usual way:

$$J_{ik} = \frac{\partial g_i(x)}{\partial x_k}, \tag{6}$$

where i and k, are indices of functions and variables, respectively. The nonzero elements of the Jacobian characterize the sparsity patterns, which is an important information for solvers that make use of it. Moreover, non-constant elements define nonlinear variables and constraints, which provide the information needed for selecting a solver, and for generating a computational task in the format accepted by the solver. Note that elements of the Jacobian can be defined using symbolic differentiation. For nonlinear problems, one typically also needs (e.g., for scaling the underlying optimization problem) the Hessian matrix (which is composed of partial derivatives of the Jacobian). With a symbolic differentiation utility in place, the Hessian can also be easily defined. Note that differentiation applied to the symbolic specification of a model requires just a tiny fraction of resources needed for the differentiation of an instance of a large model. Moreover, it needs to be done only once, because instances of the Jacobian and of the Hessian can be created from their symbolic specifications with the data that is used for the instantiation of the model.

Symbolic model specification has a number of advantages, including:
- It is complete, i.e., properties of all entities of the model are defined.

- It is compact, e.g., specification of the RAINS model presented in (Makowski, 2000) is composed of 21 variables and 25 constraints (both compound), while instances of this model have typically over 30,000 of each of the variables and the constraints. This is because only types of elements of each collection (of compound parameters or variables) need to be defined.
- It presents the relations between the variables in the form of algebraic expressions, which is a commonly used (by both modelers, and users with different backgrounds) representation of AMs.
- It makes it easy to verify (and also to use for various purposes, e.g., to provide relevant information to solvers) the structure of the model.
- Units of all entities are defined. This allows for a diagnostic not only of the syntax but also of the semantic correctness of the symbolic specification. Moreover, during the model instantiation, units of actual data are checked for consistency with the model specification.
- It makes it technically easy[3] to combine specifications of two or more models. This requires only uniqueness of names; given a small number of names used in symbolic specification, it is easy to resolve possible name conflicts.
- It makes it easy to extract part of a model, e.g., for either more detailed analysis or for use as a part of another model.
- It makes it easy to experiment with various modifications of the model specification.
- One source of model specification (represented in XML) is used for the whole modeling cycle.
- A version control system implemented in the `mSpecs` class and combined with the DBMS makes it easy to document the history of modification and to retrieve various versions of a model. This is an important feature for various analyses of any complex model for which, typically, a large number of instances is defined, each instance being analyzed using different preferential structures.

Model specification, although technically made easy with SMT, still remains a challenging task that requires team work by people who can contribute various elements of the needed knowledge. We would mention here just two basic issues. First, the underlying decision problem and the corresponding decision-making process must be understood. Second, the implications of various possible model specifications both on the resources needed for model development, and the scope of support the model will provide should be carefully examined. More comments on model specification are presented in Section 3.1.

4.4   Data

Data maintenance for a large complex model is by far the most risky element of any modeling process. The popular saying *"garbage in, garbage out"* for large amounts of data implies that even a tiny fraction of all data, when wrong, may lead to very misleading results from the model analysis. The problem may be difficult to trace because, for some analyses, even "very wrong" data elements may not have any practical impact on the corresponding solutions (even if a sensitivity analysis would indicate it should), while in other situations even a relatively small mistake may result in a dramatic difference between solutions (for wrong, and correct data, respectively). Collecting and verifying data needed for a small model is a relatively simple process. It is the size that makes data collection so different for large models. To illustrate this let us assume that one needs only one minute to collect and verify one data item (which is certainly an underestimation). A typical model used in text books has fewer than 20 elements of the Jacobian, therefore its data can be collected in less than an hour and can be presented in a fraction of a page (either printed or displayed) for relatively easy verification. However,

---

[3]However, such models need to be specified in such a way that their integration will be semantically correct. Generally, integration of separately developed models may be difficult, see e.g., (Kottemann and Dolk, 1992).

the Jacobian of the new version of the RAINS model will have over 10^{11} elements. Therefore assuming a working year composed of 1800 hours, collection and verification of 10^{11} data elements would require about 10^6 person-years. Fortunately, large models have sparse Jacobian, but human resources needed for collection and verification of nonzero elements still amounts to a large number of person-years.

Data for large models comes from different sources (also as results from analysis of various models), and larger subsets of data are maintained by teams. Fortunately, there is a natural division of data into subsets, which are maintained by individual persons or small teams. Persons working with well-defined subsets of data are experienced in collecting, cleansing, verifying, and maintaining the data they are responsible for. Therefore the "only" problem is how to structure the process of aggregating the subsets of data maintained by various teams (typically also using different hardware and software) into a data collection that can be used for model instantiation and analysis. To achieve this, a structured approach based on DBMSs is a must.

DBMSs have recently been used more often by various modeling systems for handling data. SMT, however, is most probably the first to use DBMS to handle all the persistent elements of the modeling process. Moreover, SMT exploits the concept of *Data Warehouse* (DW) for supporting persistency and efficiency of data handling. The latter is achieved by defining a base dataset, and supporting incremental modifications of this set (which allows for avoiding duplication of large amounts of data needed in more traditional approaches requiring the storage of complete datasets even when only a small fraction of the data is modified).

The data structures of a DW are generated automatically from the model specification. This not only assures consistency between the declarations of the parameters in the model specification and the data used for instantiations of these parameters but also saves substantial resources that would otherwise have been needed for preparing and maintaining data structures for any complex model. Moreover, SMT supports import of data from DBs, and from files having a simple, self-documenting format.

There are two issues of critical importance for a successful aggregation of verified data coming from different sources: consistency and completeness.

Consistency requires that the data is collected using compatible methodologies (which can be agreed upon relatively easily among those working with subsets of data) and is provided in (or converted to) the same units. The latter problem is commonly known by practitioners which is the reason for including the attribute handling units in the base class `Entity`. Thus, the units of all entities used in SMT are checked at two stages: first, during the model specification (to make sure that the parameters and variables of the model are in compatible units); second, during the model instantiation (to make sure that the data actually used is consistent with the model specification).

Completeness means that all compound data objects needed for instantiation of the model are available, and that each of them contains elements of data corresponding to all indices defined by sets that can be selected to define collections of entities used for various model instances. This can easily be achieved by providing each user with a selection of sets that correspond to the data that is available for her/him (this information is managed by the `mResources` object).

In addition to consistency and completeness, the data needs to be documented. The role of documentation is twofold. First, it makes it possible to get (possibly from different DBs) the subsets of data needed for a particular task (e.g., model instantiation, definition of a preferential structure, analysis of results) in an efficient and robust way. Second, it is essential to assure modeling transparency, i.e., to be able to provide information about each element of data. In any well-organized activity, however, documentation is developed; therefore the "only" problem is to structure this activity to be consistent with the whole modeling. Efficient and

robust documentation is certainly a prerequisite for effective maintenance and use of huge data resources.

The data in SMT is maintained by DBMSs through the Web interface, with XML used for handling meta data, and by objects of the `mData` and `mDoc` types. We outline here the functionality of these classes without presenting details of their implementation.

For each compound element of the data there is a `mData`-type object that maintains the corresponding meta data, and includes a `mDoc`-object which documents creation, modification, and use of data. Both types of objects store and process information in the XML data format.

Meta data includes not only the structure of the data (i.e., definitions needed by DBMSs to handle the data), but also information needed by `Entity`-type objects. Hence, meta data provide the complete information needed to access data from a DBMS using any SQL-based utility. In addition to this standard way, classes can be derived from `mData` for handling data in different formats, including:

• BLOB (Binary Large Object) type of data;
• HDF[4] formats; and
• Text files.

Although SMT uses XML for data, it does so in a way that is different from that used in commonly known XML-based applications, which typically documents each data item separately. Such an approach has some advantages (it is easy to implement), but it is not practicable for large amounts of data because it documents and stores each data item in a redundant (a lot of information is repeated for each data item) and verbose (a data item from a collection having several indices may require more than 100 characters) text form; thus, a substantial amount of resources is required for parsing huge amounts of such texts.

SMT uses XML only for meta data, which contain all the necessary information about the data structure (including types and units of each data element) and documentation; sparse or dense data structures are used depending on the sparsity characteristics of the corresponding data items. Therefore, the actual data is stored without any redundant information. Moreover, if necessary (e.g., for huge amounts of data) more efficient ways (e.g., based on BLOB or HFD) can be used to combine the advantages of a standard use of DBMSs with efficient handling of large amounts of numerical data.

The documentation of data processing and use can be done on different levels of detail. The basic information (such as date, user name, options requested for each object to be used) is stored automatically by each SMT application. Additionally, a user accessing a DB with privileges for data creation or modification is asked to write comments, which are logged. A more advanced documentation (e.g., automatic logging of changes in a way that allows for documenting the complete history of modifications, and optional undoing of the changes) can be included in applications that manipulate data.

Finally, we mention that various collections of data registered in SMT resources have different statuses that also determine access to each of them. A set of data is first accessible only to a person who collects and cleans the data; it is then made readable for a small team that may verify the data before it is made readable for a defined group of users. Write permissions (which imply persistent modification of the data) are typically restricted to a small group of developers. However, users with the read-only access to the data may have a possibility of modifying a copy of selected elements of data, and storing the modified data as a new resource.

---

[4]The public domain library, developed by the National Center for Supercomputing Applications, Illinois, USA (URL: `http://hdf.ncsa.uiuc.edu/HDF5`), for efficient and portable handling of large amounts of data.

4.5   Instance definition

Model instance is defined by a selection of two objects: a model specification, and a set of data to be used later for defining all the sets of the compound variables, and all the parameters in the constraints, as well as for the Jacobian and Hessian, if the latter are used.

Instance creation starts with a selection of a model specification (from the choices available for the user, which are provided by SMT using an object of `mResources`-type). The user then defines in an interactive way all the sets needed for definitions of collections for the compound entities and constraints. The choices presented to the user are limited to those for which the corresponding data is available for this user. The dialogs for these selections are generated dynamically using the dictionaries of pertaining indices (see Section 4.2.3 for details). In the next step, the user selects in a similar way the data needed for definitions of the parameters of the model. During this procedure, the user can optionally redefine the values of selected parameters (such as lower and/or upper bounds, tolerances, etc).

If the requested instance is defined successfully, then it is added to SMT resources. This is archived by a two-step procedure. First, the model specification and all used data items are assured to be persistent (i.e., the objects that are not yet in the corresponding DW are stored there). In this way it is always possible to generate an equivalent model instance without the need to keep multiple copies of persistent objects. Second, the instance is stored in the resources DB. The stored version contains automatically generated documentation about the definition of the instances, and mappings of the persistent objects needed for generation of an equivalent instance. The persistent objects also keep track of the instances that use them, so that they can be removed whenever all instances using a particular object are removed from SMT resources.

4.6   Instance analysis

Model analysis is probably the least-discussed element of the modeling process. This results from the focus that each modeling paradigm has on a specific type of analysis. However, the essence of model-based decision-making support is precisely the opposite; namely, to support various ways of model analysis, and to provide efficient tools for comparisons of various solutions. In order to do that consistently, one needs to:

- Assure that the set of feasible solutions is not restricted by constraints corresponding to any preferential structure; and
- Organize two levels of analysis: first, for model instances (each, by definition, has the same set of feasible solutions), second, comparative analysis of results obtained for various instances (which typically have different sets of feasible solutions).

Restricting the set of feasible solutions by constraints corresponding to a preferential structure has serious consequences, the symptoms of which are commonly known; the source of the problem, however, is rarely treated correctly. A countless number of optimization runs ends with reporting the problem as infeasible. This clearly indicates that the formulation of the problem was wrong (because each decision problem typically has an infinite number of solutions). Modelers and users can be happy if they see such a report (because they can analyze the source of the problem, and hopefully correct the formulation of the optimization tasks). However, if an optimal solution is found, then there is little incentive to check if optimality is achieved on the actual set of feasible solutions, or on its (often dramatically smaller) subset. This observation provides a strong argument for not restricting the set of feasible solutions by including instance constraints reflecting a preferential structure in the model.

The instance analysis is composed of a sequence of steps, each of which consists of:
- Selection of the type of analysis, which includes simulation, single-criterion optimization,

soft simulation, multicriteria model analysis.
- Definition of the corresponding preferential structure (see Section 2.2).
- Selection of a suitable solver, and specification of parameters that will be passed to a solver.
- Generation of a computational task representing a mathematical programming problem the solution of which best fits the user preferences.
- Monitoring the progress of the computational task, especially if it requires a substantial amount of computing resources.
- Translation of the results to a form that can be presented to the user.
- Documenting and filing the results, and optional comments of the user.

While the basic functionality for instance analysis is being implemented, full support of this stage requires much more work. This issue is briefly discussed in Section 4.9.

4.7   Model analysis

Model analysis is composed of comparative analyses of results from analyses of various instances. This is a very much-needed element, but its implementation is one of the most challenging open problems still waiting for a solution. The main difficulty is how to help users carry out various comparisons of solutions to problems having different sets of feasible solutions, which result from different values of model parameters and/or different values of external decision variables. Such comparative analyses may probably be represented in a knowledge base, and should help users either to define other instances of the model, or to concentrate on a deeper analysis of those instances that best represent the decision problem.

An effective approach to model analysis needs to be based on advanced applications of knowledge engineering, and this requires much greater resources than are currently available for the development of SMT. Therefore SMT currently supports only the data management of results obtained for various model instances.

4.8   Modeling environment

A modeling environment is an infrastructure composed of modeling resources that support the whole modeling cycle. Modeling resources are composed of modeling tools, models, and data, and computing hardware supporting their use.

The availability of resources is not restricted to a local computer. Instead, a client-broker-server architecture on the Internet enables users to search for specific resources in distant locations, or to contribute new resources by easily adding them to this system. Thus, the applications needed can consist of dynamically shared and geographically distributed resources (data, models, and modeling tools). Knowledge about existing model specifications and data should be stored in a (distributed) knowledge base, permitting the user to apply existing classes and subclasses for his own model specification. Thus in most cases it is not necessary for the user to build the desired specifications from scratch. Unfortunately, this paragraph outlines a vision that is realistic, but requires a large amount of resources for its implementation.

SMT implements only portions of such a modeling environment. It does support all the elements of the structured modeling process needed for actual implementation of complex models; it fully integrates the modeling process with a DBMS through the Web, and the use of modeling resources (composed of models, data, modeling tools) available on heterogeneous hardware and software at distant locations, thus it also enables collaborative work across physical and disciplinary boundaries. However, the tools supporting the management of a modeling process are still rather primitive, and their use requires a substantial amount of tacit knowledge.

4.9   Open challenges

The complexity of the modeling process is due to the diversity of model types, methods of model analysis, and tools used in the modeling process. These elements need to be configured, to exchange information, to be executed, and to be monitored at various steps of the modeling process. This process is not simple. On the contrary, even some of its component elements are complex. For example, one type of modeling tool is the solver, a piece of software that provides a solution for a given type of computational problem. Solvers, especially for large problems, are highly specialized for efficiency for certain types of computational problems. There is no one-to-one correspondence between the type of model and a solver used for its analysis, because different types of analysis result in various types of computational tasks. Moreover, even for the same type of computational problem, different solvers have varying efficacy, depending on the detailed characteristics of the problem. Therefore, applying a suitable solver to each computational task (which often requires specification of a case-specific set of parameters for the solver) requires specialized skills.

Converting tacit knowledge (especially that acquired with different modeling paradigms in different application areas) into explicit reusable knowledge is still an open scientific issue that is regarded as very important for improving the quality and efficiency of knowledge-based work, see e.g., (Borst, 1997; Gruber, 1995; Jarke, Klemke and Nick, 2001). The knowledge to be provided should correspond to the needs of both the model developers and the end users, and should provide support on two levels: first, user-oriented support throughout the whole modeling process; second, technical-type support for configuring, running, and monitoring modeling and computational tasks.

In addition to knowledge-based modeling support, there are other challenges that need to be met in order to provide all the functionality specified by the requirement analysis presented in Section 3.2. In particular, the current implementation of SMT is restricted to linear models. Extensions needed for handling more complex types of AMs, and for various modeling paradigms, can be implemented incrementally. Moreover, the support for documentation of the modeling process needs to be enhanced.

5   Conclusions

The modeling technology presented here handles the components of a modeling process in a structured way exploiting the advantages of DBMS, XML, and OOP technologies.

SMT has the following key advantages:

1. It supports basic modeling activities (and their documentation) through the full model life cycle.
2. It promotes quality (e.g., by assuring consistency throughout the whole modeling process, and using commercial DBMSs), and facilitates separation of data, model specification, model instantiation, and model analysis.
3. It is efficient; in particular its implementation conforms to the three basic requirements for applications suitable for large models:
   - Each type of object is handled efficiently, i.e. without overhead caused by functionality needed for other types of objects;
   - Common (for more than one type of object) functionality is provided without reimplementations; and
   - Modeling process is integrated with modern DBMS technology, thus providing a natural way to use proven (and familiar to many practitioners) technology for management of all persistent elements of the whole modeling process.

4. It is open to future extensions. Although the current implementation supports only linear models, SMT is designed for AMs, and therefore functionality needed for more complex types of AMs can be added by exploiting the advantages of OOP.

While some features of SMT are already present in various modeling systems, SMT is probably the only implementation of SM that is fully integrated with DBMS, and can actually be used for collaborative development and for the distributed use of models that are as complex as RAINS.

The prototype implementation of SMT also has a number of limitations, as summarized in Section 4.9. However, the author is convinced that the prototype of SMT will provide an advanced starting point for gradual extensions toward a modeling environment that will meet the requirements summarized in Section 3.2.

Acknowledgments

References

Ackoff, R.: 1967, Management misinformation systems, *Management Science* **14**(4), 43–89.

Ackoff, R.: 1979, The future of operational research is past, *Journal of OR Society* **30**(2), 93–104.

Alcamo, J., Shaw, R. and Hordijk, L. (eds): 1990, *The RAINS Model of Acidification*, Kluwer Academic Publishers, Dordrecht, Boston, London.

Amann, A. and Makowski, M.: 2000, Effect-focused air quality management, *in* Wierzbicki et al. (2000), pp. 367–398. ISBN 0-7923-6327-2.

Bhargava, H. and Krishnan, R.: 1998, The World Wibe Web: Opportunities for operations research and management science, *INFORMS Journal on Computing* **10**(4), 359–383.

Bhargava, H., Krishnan, R. and Mueller, R.: 1997, Decision support on demand: On emerging electronic markets for decision technologies, *Decision Support Systems* **19**(3), 193–214.

Bisschop, J. and Roelofs, M.: 2004, AIMMS, *The Language Reference*, Paragon Decision Technology, Haarlem, The Netherlands.

Booch, G.: 1991, *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, New York, Sydney, Bonn, Tokyo.

Borst, W.: 1997, Construction of engineering ontologies for knowledge sharing and reuse, *Technical report*, University of Twente, Enschede, The Netherlands.

Brooke, A., Kendrick, D. and Meeraus, A.: 1992, GAMS, *A User's Guide, release 2.25*, The Scientific Press, Redwood City.

Carlsson, C. and Fullér, R.: 2002, *Fuzzy Reasoning in Decision Making and Optimization*, Physica Verlag, New York.

Carlsson, C. and Walden, P.: 1995, Hyperknowledge and expert systems: A case study of knowledge formation processes, *in* J. Nunamaker and R. Sprague (eds), *Proceedings of HICSS-28*, IEEE Computer Society Press, Los Alamitos, pp. 73–82.

Chapman, C.: 1988, Science, engineering and economics: OR at the interface, *Journal of Operational Research Society*.

Chapman, C.: 1992, My two cents worth on how OR should develop, *Journal of Operational Research Society* **43**(7), 647–664.

Charnes, A. and Cooper, W.: 1967, *Management Models and Industrial Applications of Linear Programming*, J. Wiley & Sons, New York, London.

Codd, E.: 1970, A relational model for large shared data banks, *Comm. ACM* **13**(6), 377–387.

Cohen, M., Kelly, C. and Medaglia, A.: 2001, Decision support with Web-enabled software, *Interfaces* **31**(2), 109–129.

de Champeaux, D., Lea, D. and Faure, P.: 1993, *Object-Oriented System Development*, Addison-Wesley Publishing, New York, Sydney, Tokyo.

Dolan, E., Fourer, R., More, J. and Manson, T.: 2002, Optimization on the NEOS server, *SIAM News* **35**(6), 1–5.

Dolk, D.: 1988, Model management and structured modeling: The role of an information resource dictionary system, *Comm. ACM* **31**(6), 704–718.

Dolk, D.: 2000, Integrated model management in the data warehouse era, *EJOR* **122**(2), 199–218.

Drucker, P.: 2001, The next society, a survey of the near future, *The Economist,* **361**(8246 (Nov. 3)), 3–22.

Drud, A.: 1992, CONOPT - a large scale GRG code, *ORSA Journal on Computing* **6**(2), 207–218.

ETAN Expert Working Group: 1999, Transforming European science through information and communication technologies: Challenges and opportunities of the digital age, *ETAN Working Paper September*, Directoriate General for Research, European Commission, Brussels.

Fink, E.: 2002, *Changes of Problem Representation*, Springer Verlag, Berlin, New York.

Fourer, R. and Goux, J.: 2001, Optimization as an internet resource, *Interfaces* **31**(2), 130–150.

Fourer, R., Gay, D. and Kernighan, B.: 1993, AMPL*, A Modeling Language for Mathematical Programming*, The Scientific Press, San Francisco.

Fourer, R., Gay, D. and Kernighan, B.: 2003, AMPL*, A Modeling Language for Mathematical Programming, Second Edition*, Duxbury Press, Belmont, CA.

Fourer, R., Lopes, L. and Martin, K.: 2004, LPFML – a W3C XML schema for linear programming, *Technical report*, Department of Industrial Engineering and Management Sciences, Northwestern University, Chicago, Illinois.

Funke, B. and Sebastian, H.: 1996, Knowledge-based model building with KONWERK, *Working Paper WP-96-105*, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Geoffrion, A.: 1987, An introduction to structured modeling, *Management Science* **33**(5), 547–588.

Geoffrion, A.: 1989, Integrated modeling systems, *Computer Science in Economics and Management* **2**, 3–15.

Geoffrion, A.: 1992a, Indexing in modeling languages for mathematical programming, *Management Science* **38**(3), 325–344.

Geoffrion, A.: 1992b, The SML language for structured modeling: Levels 1 and 2, *Operations Research* **40**(1), 38–57.

Geoffrion, A.: 1992c, The SML language for structured modeling: Levels 3 and 4, *Operations Research* **40**(1), 58–75.

Geoffrion, A. and Krishnan, R.: 2001, Prospects for operations research in the e-business era, *Interfaces* **31**(2), 6–36.

Gondzio, J. and Sarkissian, R.: 2003, Parallel interior point solver for structured linear programs, *Mathematical Programming* **96**(3), 561–584.

Greenberg, H.: 1992, Intelligent analysis support for linear programs, *Computers and Chemical Engineering* **16**(7), 659–674.

Greenberg, H.: 1995, Mathematical programming models for environmental quality control, *Operations Research* **43**(4), 578–622.

Gruber, T.: 1995, Towards principles for the design of ontologies used for knowledge sharing, *International Journal of Human-Computer Studies* **43**, 907–928.

Haklay, M.: 2003, Public access to environmental information: past, present and future, *Computers, Environment and Urban Systems* **27**, 163–180.

Hloyningen-Huene, P.: 1993, *Restructuring Scientific Revolutions*, The University of Chicago Press, London.

Jarke, M., Klemke, R. and Nick, A.: 2001, Broker's longue - an environment for multi-dimensional user-adaptive knowledge management, *Proceedings of the HICSS-34 Conference*, Hawaii.

Kang, M., Wright, G., Chandrasekharan, R., Mookerjee, R. and Worobetz, N.: 1997, The design and implementation of OR/SM: A prototype integrated modeling environment, *Annals of Operations Research* **72**, 211–240.

Keeney, R. and Raiffa, H.: 1976, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, J. Wiley & Sons, New York.

Kim, H.: 2001, An XML-based modeling language for the open interchange of decision models, *Decision Support Systems* **31**, 429–441.

Kottemann, J. and Dolk, D.: 1992, Model integration and modeling languages: A process perspective, *Information Systems Research* **3**(1), 1–16.

Kuhn, T.: 1970, *The Structure of Scientific Revolutions*, The University of Chicago Press, Chicago.

Lee, E. and Neuendorffer, S.: 2000, MoML – a modeling markup language in XML, version 0.4, *Technical report*, University of California, Berkeley, USA.

Liang, T.: 1988, Development of a knowledge based model management system, *Operations Research* **36**(6), 849–863.

Liu, B.: 2002, *Theory and Practice of Uncertain Programming*, Springer Verlag, Berlin, New York.

Maclean, D.: 1985, Rationality and equivalent redescriptions, *in* M. Grauer, M. Thompson and A. Wierzbicki (eds), *Plural Rationality and Interactive Decision Processes*, Vol. 248 of *Lecture Notes in Economics and Mathematical Systems*, Springer Verlag, Berlin, New York, pp. 83–94.

Makowski, M.: 1994, Design and implementation of model-based decision support systems, *Working Paper WP-94-86*, International Institute for Applied Systems Analysis, Laxenburg, Austria. Available on-line from `http://www.iiasa.ac.at/~marek/pubs/`.

Makowski, M.: 2000, Modeling paradigms applied to the analysis of European air quality, *EJOR* **122**(2), 219–241. available also as IIASA's RR-00-06.

Makowski, M.: 2001, Modeling techniques for complex environmental problems, *in* M. Makowski and H. Nakayama (eds), *Natural Environment Management and Applied Systems Analysis*, International Institute for Applied Systems Analysis, Laxenburg, Austria, pp. 41–77. ISBN 3-7045-0140-9.

Makowski, M. and Nakayama, H. (eds): 2001, *Natural Environment Management and Applied Systems Analysis*, International Institute for Applied Systems Analysis, Laxenburg, Austria. ISBN 3-7045-0140-9.

Makowski, M. and Wierzbicki, A.: 2003a, Modeling knowledge in global information networks, *4th Global Research Village Conference. Importance of ICT for Research an Science: Science Policies for Economies in Transition*, KBN (the Polish State Committee for Scientific Research), and OECD (the Organization for Economic Co-operation and Development), Information Processing Centre, Warsaw, pp. 173–186. draft version available from `http://www.iiasa.ac.at/~marek/pubs/prepub.html`.

Makowski, M. and Wierzbicki, A.: 2003b, Modeling knowledge: Model-based decision support and soft computations, *in* X. Yu and J. Kacprzyk (eds), *Applied Decision Support with Soft Computing*, Vol. 124 of *Series: Studies in Fuzziness and Soft Computing*, Springer-Verlag, Berlin, New York, pp. 3–60. ISBN 3-540-02491-3, draft version available from `http://www.iiasa.ac.at/~marek/pubs/prepub.html`.

Mannino, M., Greenberg, B. and Hong, S.: 1990, Model libraries: Knowledge representation and reasoning, *ORSA Journal on Computing* **2**, 1093–1123.

Nakamori, Y. and Sawaragi, Y.: 2000, Complex systems analysis and environmental modeling, *EJOR* **122**(2), 178–189.

Paczyński, J., Makowski, M. and Wierzbicki, A.: 2000, Modeling tools, *in* Wierzbicki et al. (2000), pp. 125–165. ISBN 0-7923-6327-2.

Pidd, M.: 1999, Just modeling through: A rough guide to modeling, *Interfaces* **29**(2), 118–132.

Piela, P., McKelvey, R. and Westerberg, A.: 1993, An introduction to the ASCEND modeling system: Its language and interactive environment, *Journal of Management Information Systems* **9**(3), 91–121.

Rapoport, A.: 1989, *Decision Theory and Decision Behaviour, Normative and Descriptive Approaches*, Vol. 15 of *Theory and Decision Library, Mathematical and Statical Methods*, Kluwer Academic Publishers, Dordrecht, Boston, London.

Ruan, D., Kacprzyk, J. and Fedrizzi, M. (eds): 2001, *Soft Computing for Risk Evaluation and Management*, Physica Verlag, New York.

Sawaragi, Y. and Nakamori, Y.: 1991, An interactive system for modeling and decision support – Shinayakana system approach, *in* M. Makowski and Y. Sawaragi (eds), *Advances in Methodology and Applications of Decision Support Systems*, number CP-91-17 in *Collaborative Paper*, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Sawaragi, Y., Nakayama, H. and Tanino, T.: 1985, *Theory of Multiobjective Optimization*, Academic Press, New York.

Schöpp, W., Amann, M., Cofała, J. and Klimont, Z.: 1999, Integrated assessment of european air pollution emission control strategies, *Environmental Modelling and Software* **14**(1), 1–9.

Srinivasan, A. and Sundaram, D.: 2000, An object relational approach for the design of decision support systems, *European Journal of Operational Research* (127), 594–610.

Sterman, J.: 2002, All models are wrong: reflections on becoming a systems scientist, *Systems Dynamics Review* **16**(4), 501–531.

Stewart, T.: 1992, A critical survey on the status of multiple criteria decision making theory and practice, *OMEGA, International Journal of Management Science* **20**(5/6), 569–586.

Stroustrup, B.: 1997, *C++ Programming Language, Third Edition*, Addison-Wesley, Reading, Bonn, Amsterdam.

Thorsteinsson, E.: 1999, xMPS, the extended mps format for non-linear programs, *Technical report*, Carnegie Mellon University, Mathematical Sciences Department, Pittsburgh, USA.

Tsai, Y.-C.: 2001, Comparative analysis of model management and relational database management, *Omega* (29), 157–170.

Tversky, A. and Kahneman, D.: 1985, The framing of decisions and philosophy of choice, *in* G. Wright (ed.), *Behavioral Decision Making*, Plenum, New York, pp. 25–42.

van Waveren, R., Groot, S., Scholten, H., van Geer, F., Wösten, H., Koeze, R. and Noort, J.: 1999, *Good Modelling Practice Handbook*, STOWA, Utrecht, The Netherlands. `http://waterland.net/riza/aquest/gmpuk.pdf`).

Wierzbicki, A.: 1977, Basic properties of scalarizing functionals for multiobjective optimization, *Mathematische Operationsforschung und Statistik, s. Optimization* **8**, 55–60.

Wierzbicki, A.: 1997, On the role of intuition in decision making and some ways of multicriteria aid of intuition, *Multiple Criteria Decision Making* **6**, 65–78.

Wierzbicki, A. and Makowski, M.: 2000, Modeling for knowledge exchange: Global aspects of software for science and mathematics, *in* P. Wouters and P. Schröder (eds), *Access to Publicly Financed Research*, NIWI, Amsterdam, the Netherlands, pp. 123–140.

Wierzbicki, A., Makowski, M. and Wessels, J. (eds): 2000, *Model-Based Decision Support Methodology with Environmental Applications*, Series: Mathematical Modeling and Applications, Kluwer Academic Publishers, Dordrecht. ISBN 0-7923-6327-2.

Wright, G., Worobetz, N., Kang, M., Mookerjee, R. and Chandrasekharan, R.: 1997, OR/SM: A prototype integrated modeling environment based on structured modeling, *INFORMS Journal on Computing* **9**(2), 134–153.

Yu, P.: 1985, *Multiple-Criteria Decision Making: Concepts, Techniques, and Extensions*, Plenum Press, New York, London.

Yu, P.: 1990, *Forming Winning Strategies, An Integrated Theory of Habitual Domains*, Springer Verlag, Berlin, New York.

Zadeh, L.: 1965, Fuzzy sets, *Information and Control* **8**, 338–353.

Zimmermann, H.: 1978, Fuzzy programming and linear programming with several objective functions, *Fuzzy Sets and Systems* **1**, 45–55.

A   Programmer-oriented outline of SMT structures

Along with the background presented in Section 4.2, we provide here more technical characteristics of some classes used in SMT. The declarations of the classes presented here have been simplified to include only those elements that are essential for understanding the structure of SMT, because a presentation of the full code would require too much space. Moreover, in the actual implementation template classes are extensively used. The syntax of those classes, however, is rather complex (especially for readers not familiar with C++); therefore we present only one template class, and other template classes are presented as traditional classes. Summing up these reservations we ask readers familiar with C++ to treat the presented declarations of classes as pseudocode.

We start with an outline of the fundamental base class:

```
class Entity {
public:
   Entity(String &symbol, String &units, ... );
protected:
   bool in_range(double val) {return val >= lower && val <= upper;}
private:
   Entity();            // to prevent not-well defined objects
   String symbol;    // used in algebraic expressions
   String units;     // used in semantic check of expressions
   String name;      // used in documentation
   double lower;     // lower bound for values
   double upper;     // upper bound for values
   double zero;      // zero tolerance for values
};
```

The class `Entity` handles the data of a primitive entity, i.e., the data that is required for any element of an algebraic expression. The public constructor requires values of all elements of the data, and its implementation assures that the data of the created object conform to the specified requirements (e.g., formal requirements for symbols, and names, syntax of units, consistency of bounds, a range for zero tolerance). The default constructor is private, which prevents objects being created with undefined data by mistake (for this class use of default values for data does not make sense). The function `in_range(double val)` is used for checking, if the value is within the bounds. Because this function is of the protected type, it can be used only by objects of the derived classes.

```
class Param : public Entity {
public:
    Param(String &symbol, ..., value = INFTY);
    bool in_range() {return val >= lower && val <= upper;}
    double value() {return val;}
private:
    Param();           // to prevent not-well defined objects
    double val;        // value of the parameter
};
```

The class `Param` handling model parameters is inherited from the class `Entity`. Therefore, its constructor needs to provide all the values needed by the `Entity` constructor, and additionally a value for the parameter. Sometimes it is practical to create the `Param` objects prior to defining their values. This is achieved by not specifying the value for the constructor, which in such a case will use the predefined value of `INFTY`, which is conventionally a large number (i.e., clearly outside of the domain of any entity of the model).

```
class Variable : public Entity {
public:
    enum VarType{DECISION, OUTCOME, EXTERNAL, AUX};
    enum MathType{FLOAT, INTEGER, BINARY};
    Variable(String &symbol, ..., VarType var, MathType math);
private:
    mVect<unsigned short, mSet> index; //  definitions of indices
    VarType type;     // type of variable (decision, outcome, etc)
    MathType math;    // math_type
};
```

The class `Variable` handles compound variables, and is inherited from the class `Entity`, which is easy to justify, if one observes that objects of the `Variable` type need all the functionality of the `Entity` class in addition to the functionality required for handling various types of compound variables. The latter includes the definitions of sets of indices, and the declarations of two types: first defining the role of the variable in the model, and second its mathematical programming type.

As an example of template classes, we present a vector class which is designed for efficient handling of data (whose size is not necessarily known when the object is constructed) of various classes. Therefore the template uses two parameters, one for the type of indices, and the second for the type of elements to be stored. Moreover, this class can also be used as a base class for more specialized types of vectors, in particular for sparse vectors, which are extensively used for large problems.

```
/* template vector class
   I - type of indices (unsigned: short, int or long)
   T - type of elements
*/
template <class I, class T> class mVect {
public:
   mVect(I length = 0, const T * const v_ = 0);
   mVect(const mVect &copy);   // copy ctor
   virtual ~mVect() {          // destructor
      if(length_)
         delete[] v;
   }
   mVect& operator = (const mVect &copy);
   T& operator[] (I i) const {
      if(i >= length_)
         msg_fatal("mVect::operator[]: assessing %d-th "
            "element, only %d defined.", i, length_);
      return v[i];}
   virtual void resize(I new_lenght);
   virtual bool resize(double fraction, bool strict = false);
   void fill_el(I pos, const T& value);   // resize, if necessary
   I length() const {return(length_);}
protected:
   I length_;         // length of the vector
   T *v;              // space for values
};
```

Simple examples of using this class include declarations of collections of names (of a standard data type `String`), and of pointers to variables, for which the data type is defined in SMT:

```
typedef unsigned short USHORT;
mVect<USHORT, String> names;
mVect<USHORT, Variable*> x;
```

More complex structures can also be efficiently built, e.g., dense matrices of real numbers:

```
typedef mVect<USHORT, mVect<USHORT, double> > R_MATRIX;
R_MATRIX a_matrix, another_matrix;
```

Note that while the type definition may look strange to readers not familiar with C++, use of such objects is very natural, e.g., the value of an element of such a matrix is obtained in a very familiar way:

```
double a_value = a_matrix[i][j];
```

where i and j are indices of a row and a column, respectively. Moreover, the `operator[]` is defined in such a way that the validity of indices is checked.

These simple examples illustrate the way of using OOP for building the data structures needed for structured modeling. Commonly used operations (e.g., algebraic) are typically available from well-tested libraries; more specific functionality can often be programmed on a higher level (e.g., in a base class), which greatly improves the effectiveness and robustness of development of applications.