International Institute for
Applied Systems Analysis
Schlossplatz 1
A-2361 Laxenburg, Austria

Tel: +43 2236 807 342
Fax: +43 2236 71313
E-mail: publications@iiasa.ac.at
Web: www.iiasa.ac.at

Interim Report IR-04-051

Support of Model Analysis within Structured Modeling Technology

Cezary Chudzian (C.Chudzian@elka.pw.edu.pl)

Approved by

Marek Makowski (marek@iiasa.ac.at)
Senior Research Scholar, Risk, Modeling and Society Program

September 2004

Contents

Foreword

This paper presents the results the author achieved during his participation in the Young Scientists Summer Program (YSSP) 2004. However, the impact of these results is wider than it can be seen from this paper. This is because of the synergy resulting from team work.

Three participants of the YSSP 2004: Bartłomiej Prędki, Cezary Chudzian, and Vladimir Molchanov were members of the team working on the development of the Structured Modeling Technology (SMT). The other two members of the team were Michał Majdan (who has spent five months at IIASA on leave from the National Institute of Telecommunications, Warsaw, Poland) and myself.

The development of SMT is a long-term challenging undertaking that requires collaborative work of researchers having experience not only in methods and tools for advanced modeling but also knowledge and skills in DBMSs (Data Base Management Systems), XML (Extensible Markup Language), and object-oriented programming of Web-based applications.

Michał Majdan has designed the user interface to, and basic data structures of SMT. He had been coordinating the design of elements developed by other colleagues in order to be able to smoothly combine all elements into one system. This work has not been documented yet.

The contributions of the other three members of the team have been described in three Interim Reports (IRs), which constitute a kind of virtual set describing the collaborative work. I briefly summarize the scope of each IR encouraging the reader to become familiar with all of them:

- Bartłomiej Prędki (IR-04-050) has implemented an extension of SMT (originally designed for algebraic models) by implementing a prototype handling of decision rule models; he has adapted a suite of software supporting application of decision rules for analysis of qualitative data to work with SMT. Moreover, he tested the concept by a medical case study developed in collaboration with the Ottawa University.
- Cezary Chudzian (IR-04-051) has developed the key elements of SMT that support a part of modeling process composed of instance definition, specification of preferential structure for various types of model analysis, and efficient handling of underlying complex and large data structures (e.g., for parametric optimization, and diversified sets of results, both composed of huge amounts of data).
- Vladimir Molchanov (IR-04-052) has explored possibilities of using XML for automatic documentation of the modeling process, and implemented a prototype of automatic documentation of model specification, which is the most difficult element of the documentation due to the complexity of the structure of symbolic specification and the requirement for supporting gradual modifications of the descriptive part of the documentation (which is added to the part resulting from the

interactive model specification).

Finally, I would like to stress that it has been a pleasure to be the leader of the SMT team during the Summer 2004. Each member of the team not only has very good professional skills but also abilities necessary for team work, strong dedication to achieve good results, and to have fun during the short periods spent away from keyboards.

We plan to make the SMT publicly available in 2005. Therefore, I invite the readers to not only become familiar with the IRs mentioned above, but also to visit `http://www.iiasa.ac.at/~marek` in Spring 2005 to check further developments of SMT.

Marek Makowski

Abstract

This report presents the design and implementation of basic components of Structured Modeling Technology (SMT), and discusses in more detail the issues related to model analysis. First, the representation of symbolic model specification composed of entities, sets, indices and relations in the SMT is outlined. Then, the structure of a data warehouse for storing values of model parameters, along with the data sets hierarchy, giving strong support for the tracking of changes in data is explained. The concept of initial data import and subsequent updates allows to implement an efficient data versioning mechanism. Next, the model instance (a selection of a specification and a data set) is described.

The main topic discussed in the paper is model analysis, which is composed of a series of analyses of various instances. The analysis of each instance is in turn composed of a set of analysis tasks, each of which can be performed with a different analysis type, such as optimization, simulation and multicriteria analysis. Each type of model analysis uses a specific representation of a preferential structure. Moreover, some analysis tasks (e.g., parametric optimization) result in a possibly large number of computational tasks. Thus, the paper presents an efficient approach to automatic generation of DBMS's data structures that supports the whole modeling process, and discusses in more detail such issues of DBMS-based management as: definition of model instances, specification of user preferences, generation and management of computational tasks, and handling results of a possibly huge number of analyses.

Finally, plans on further SMT development with special attention to analysis of results and distributed computations are summarized.

Keywords: model-based decision support, structured modeling, algebraic models, model analysis, preferential structure, results analysis, distributed computations.

Acknowledgments

This paper reports results of research done during the author's participation in the Young Scientists Summer Program (YSSP) 2004, held at the International Institute for Applied Systems Analysis in Laxenburg, Austria. The author was a member of the team working on the development of the Structured Modeling Technology framework (SMT), which is a long-term activity of IIASA's Risk Modeling and Society program.

The author would like to give many thanks to Dr Marek Makowski for his supervision during the YSSP 2004, and to Michal Majdan for the cooperation.

Finally, the author would like to thank the Polish National Member Organization of IIASA for the financial support which made his participation in the YSSP possible.

About the Author

Cezary Chudzian received his M.Sc. in Computer Science from Warsaw University of Technology in 2002. The title of his thesis was "Graphical Environment for Solving Global Optimization Problems with Ψ-transform Algorithm". He is a researcher at the Advanced Information Technology Department of the Polish National Institute of Telecommunications, and a second year Ph.D. student at Warsaw University of Technology. His Ph.D. thesis is on knowledge discovery in large databases. His main scientific interests include: practical applications of knowledge discovery techniques, machine learning theory, global optimization, and software engineering, especially related to efficient use of DBMSs for management and analysis of large data sets.

In September 2004 Cezary Chudzian joined the Risk, Modeling and Society Program for three months to work on the implementation of the Structured Modeling Technology, and on its application to complex models.

Support of Model Analysis within Structured Modeling Technology

Cezary Chudzian[*] *(C.Chudzian@elka.pw.edu.pl)*

1 Introduction

Structured Modeling Technology (SMT) is a consistent framework for supporting the whole modeling process and is designed for efficient and effective support for complex models. SMT is designed to cover all the steps of the modeling process, from symbolic specification, and data management, through model instantiation to analysis and results' maintenance. SMT is based on the well established technology of relational databases, it uses the Web as the interface for all functionalities and provides automatic documentation of all modeling activities. Currently it is possible to develop and analyze two types of models within the framework: algebraic and decision rules.

One of the characteristic features of SMT is its support of persistency for modeling activities. Namely, every created data modification, model instance, or any performed analysis and its results, may be either easily reconstructed or retrieved from the database.

Access to SMT functionalities is realized through the WWW interface. Thus the modeling process is supported without having any specialized software installed on the user's computer. In other words, the only thing the user needs is a web browser and an internet connection. Modeling resources are stored on the SMT server, and are accessible through the Web from any location, at any time, by users authorized to have access to selected parts of them. Thus collaborative modeling, by interdisciplinary teams, located at distant locations is effectively supported. Moreover, collaborative modeling implies the demand for transparency of the whole modeling process. Especially the automatically generated documentation of the model specification and data, as well as modeling activities of the collaborating teams' members have to be transparent.

In this report, special attention will be paid to the organization of data handling of a model instance within SMT and an even closer look will be taken at the topic of model instance analysis.

Data warehouse and concepts of updates allow for persistency of the data part and for handling large amounts of data in an efficient manner.

[*]National Institute of Telecommunications, Szachowa 1, 04-894 Warsaw, Poland, *and* Warsaw University of Technology, Faculty of Electronics and Information Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland.

Every model instance defined may be analyzed in many ways and each analysis type has a specialized preferential structure associated with it. Model analysis task may result in a possibly big number of computational tasks, where the latter is understood here as task specification for an external computer program, namely a solver. Computational task must have computational resources allocated, and have to be scheduled in a queue of tasks to be performed. The huge amount of resulting data returned from solvers is then stored back in the SMT database. Because of their large volume, results have to be efficiently managed.

The remaining part of the report is organized as follows. Section 2 provides an overview of the modeling process. An outline of objects involved in symbolic model specification can be found in Section 3. Section 4 covers organization of the data part in SMT. Model instantiation is the topic of Section 5. The part of this report that is most detailed is devoted to the analysis of model instance. Section 6 starts with an overall look at the issue of analysis of model instance. Several types of analysis tasks are discussed in more details in 6.1. The specification of preferential structure is presented from the user's point of view in Section 6.2.1, and the implementation details are provided in Section 6.2.2. Management of computational tasks associated with analyses is covered in Section 6.3, while problems of dealing with results coming from analyses are presented in Section 6.4. Conclusions and further steps follow in Section 7.

2 Modeling process

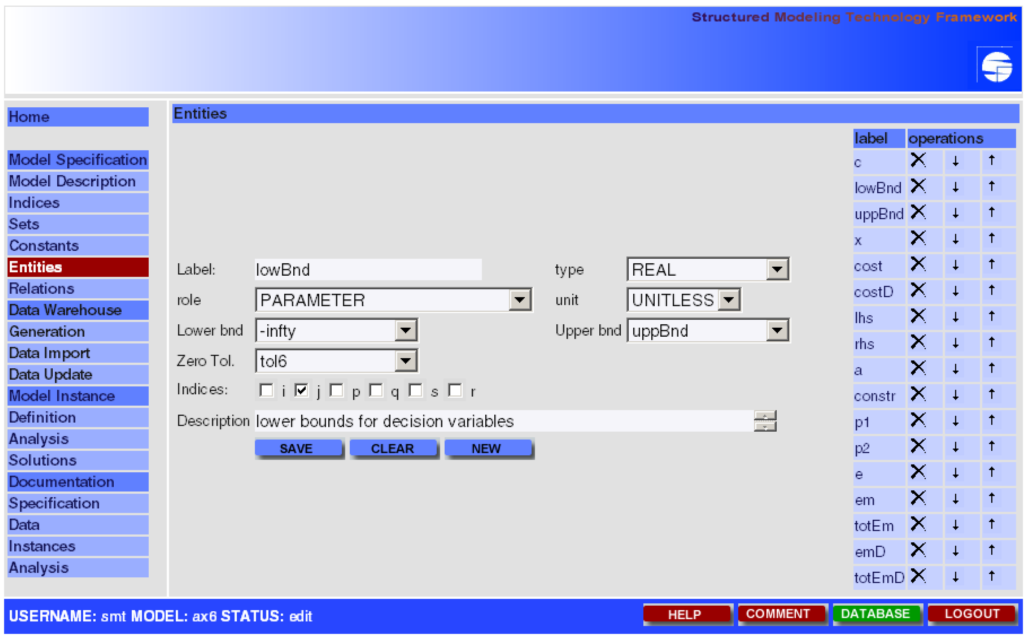
In order to provide a framework for structuring this report we start with outlining the basic components of the modeling process in the context of Structured Modeling (SM). Modeling is a network of activities, is often referred to as a *modeling cycle*, or a *modeling process*, or a *modeling lifecycle*. 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 SMT.

Modeling process starts with analysis of the problem for which the model will be developed. This stage is a collaborative work of the domain experts and modelers. As it is not practicable to attempt to formalize this stage, the problem analysis phase is not supported by SMT.

SMT supports all other phases of the whole modeling process:

- symbolic (mathematical) specification of the model;
- management of data to be used for definition of parameters of relations declared in the symbolic specification;
- definition of model instances;
- specification of various types of analysis of instances, and management of results of analyses.

We will discuss these phases in subsequent sections at different levels of detail which correspond to the focus of the research reported in this paper. Background and methodology of Structured Modeling is presented in (Geoffrion, 1987; Geoffrion, 1989; Geoffrion, 1992). More detailed discussion of Structured Modeling Technology (SMT) is given by Makowski (2004). An extensive discussion of model-based decision support methods with a detailed presentation of modeling tools and their applications to complex problems is presented by Wierzbicki, Makowski and Wessels (2000). Readers interested in modern decision making may want to consult (Wierzbicki and Wessels, 2000).



Label: lowBnd
type: REAL
role: PARAMETER
unit: UNITLESS
Lower bnd: -infty
Upper bnd: uppBnd
Zero Tol.: tol6
Indices: ☐ i ☑ j ☐ p ☐ q ☐ s ☐ r
Description: lower bounds for decision variables

label	operations		
c	✗	↓	↑
lowBnd	✗	↓	↑
uppBnd	✗	↓	↑
x	✗	↓	↑
cost	✗	↓	↑
costD	✗	↓	↑
lhs	✗	↓	↑
rhs	✗	↓	↑
a	✗	↓	↑
constr	✗	↓	↑
p1	✗	↓	↑
p2	✗	↓	↑
e	✗	↓	↑
em	✗	↓	↑
totEm	✗	↓	↑
emD	✗	↓	↑
totEmD	✗	↓	↑

USERNAME: smt MODEL: ax6 STATUS: edit

Figure 1: Entity definition screen of SMT (with the navigation panel on the left side).

Users of SMT are guided through the modeling process with the help of a navigation panel (which is illustrated on the left side of Fig. 1), which only has those choices active that correspond to modeling actions that can be performed at a given phase of a modeling process.

3 Model specification

Model specification of SMT consists of:
- indices,
- sets,
- entities, and
- relations between entities.

Any large and/or complex model is built up using the concept of primitive entities (parameters, variables, relations) and indices that expand primitive entities into compound ones. Thus a specification of any complex model starts with the definition of indices.

Two types of indices are supported by SMT:
- collection type, which can be unordered (although various forms of ordering are typically used for analysis and/or reporting purposes),

- sequence type, which are ordered and have basic properties of a sequence (such as an initial value, step, and number of elements); this type is used e.g., in dynamic models.

Moreover, an index can be defined as a composition of other indices, e.g., for simplifying the notation. Composition of two, or more indices that appear often together in a symbolic model specification is, for the sake of more understandable model formulation, substituted with one compound index.

Another type of index is called *alias*, which defines an alternative name for a given index. This type of index is typically used to avoid ambiguity and possible inconsistency in model specification, e.g., in cases when an entity is indexed more than once by the same index (e.g. a_{ii}).

Values of indices are organized into sets. A set in SMT is defined as an enumeration of all possible values for an index in a specific context (e.g., within a relation or a set of relations). Sets are declared during the model specification, their definitions (i.e., actual specification of values of indices that belong to a given set) is being done by the specification of corresponding data. Sets can also be indexed (see e.g., A_i in equation (1) on page 5). An indexed set is actually a member of a collection of sets; in other words a member of the collection is defined for every value of the indexing index (which is in turn member of another set).

The concept of entity has been introduced to capture common functionality of various elements (e.g., parameters, variables, relations) used in a model specification. An entity is uniquely identified by a label, and has other attributes that define it: the entity's role in the model, type (in terms of mathematical programming), indices (used for defining compound entities). Moreover, depending on the entities' roles also other attributes are defined, such as lower and upper bounds, zero tolerance, sets of indices, etc.

As an example we provide a few characteristics of selected types of entities:

- Entities in the roles of constants have their values defined within the model specification.
- Entities that are parameters have lower and upper bounds defined (possibly using other parameters, or constants). Their actual values are defined later by selection of data used for the definition of a model's instance.
- Entities that are variables are divided into subsets of decisions, outcomes, auxiliary and external (independent) ones. Their lower and upper bounds are defined symbolically using either parameters or constants.
- Entities that declare relations have attributes specifying the type of the relation (e.g., an assignment or a constraint), zero tolerance, and (for constraints only) entities that will be used for lower and upper bounds definition.
- For each entity defining a relation, an algebraic expression that uses the previously defined parameters and variables, is defined. For compound relations the sets of indices are also selected.

Entities of all types except for constants are declared by the same interface illustrated in Fig. 1 on page 3. Thus the user does not need to learn any modeling language syntax to use SMT. Moreover, basic elements of good modeling practice are enforced by requesting a complete declaration of each entity. The right side of the form shown in Fig. 1 contains the list of already declared entities. The user may edit any of them, remove those which are not needed,[1] or interactively redefine the order in which entities are listed.

Exemplary assignment relation (1) illustrates almost all of the concepts used for symbolic model specification outlined above.

$$em_{ipa} = \sum_{t \in T_{pa}} e_{ipat} x_{ipat} \quad i \in I,\ p \in P,\ a \in A_i \tag{1}$$

Namely:
- i, p, a, t are indices,
- e_{ipat} is a compound parameter,
- em_{ipa} and x_{ipat} are compound variables, (outcome and decision, respectively),
- T_{pa} is indexed set used in the assignment-type of relation that defines the variable em,
- Sets I, P, A_i define the set of relations to which the symbolic formula (1) will actually be expanded, whenever a representation of an instance of the model is generated.

4 Data

An efficient, robust, and well documented handling of any large model data typically requires a big part of resources assigned to the model development and maintenance. In order to be able to trace and document each set of data used in instantiations of the model, and to reconstruct every instance of the model, one needs to assure persistency of every data modification. In order to achieve this, SMT exploits the concept of data warehouse. A data warehouse is a kind of data repository fulfilling the persistency requirement.

However, during the model analysis typically only a small part of a large data set is modified. Storing the full data set, each time a part of data is modified, would not be efficient. Thus, changes of data are handled by applying the concept of data updates. Updates allow to avoid storing whole modified data sets. Moreover, updates can be made incrementally, i.e., can be applied to previously made updates. Thus the structure of updates forms a tree hierarchy with initial data import as the root, and updates as nodes at lower levels of the tree illustrated in Fig. 2 on page 6.

The whole hierarchy is defined in a meta data layer. Every single data record is linked to exactly one update. Complete data set definition is given by the whole path, from a selected tree node, up to the root.

DBMS tables of the data warehouse are generated automatically on the basis of the model specification. This approach not only saves a lot of resources but also guarantees consistency between the definitions of parameters (being a part of the model specification) and the data warehouse.

Moreover, during the warehouse generation process, not only data structures for parameters and sets are defined, but also DBMS tables for storing the results of analyses of instances are created, as their form depends on the model specification as well.

[1]The remove operation will be denied, if the selected entity is used in declaration of another entity.

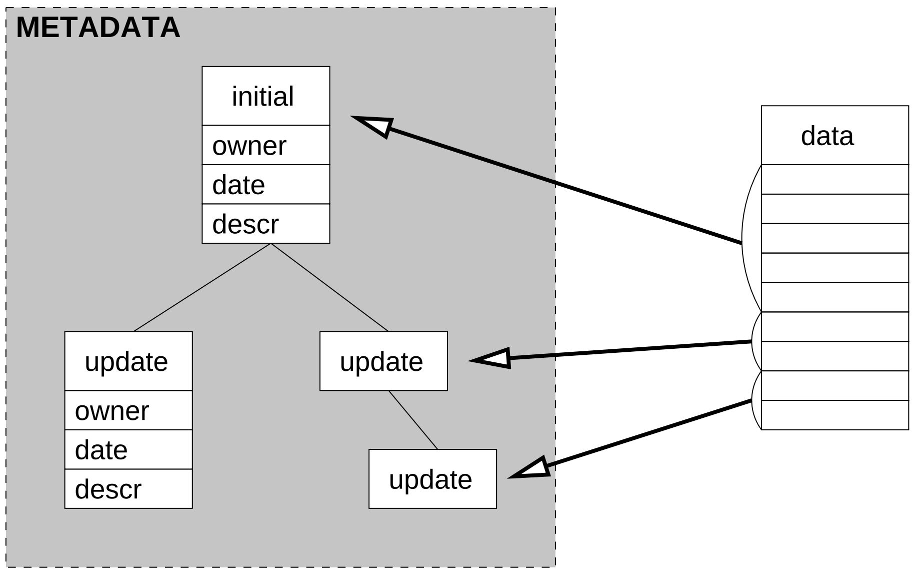


Figure 2: Organization of data updates tree.

5 Model Instance

A model specification together with a chosen data update (which implies applications of all updates on the path from this update up to the initial data import) define a model instance (which is also called a substantive model or a core model). An instance defines a realization of the symbolic specification using a selected set of data composed of the indices' sets (which will be used for expanding compound entities) and values of parameters used in the relations. More on organization of analysis of a model instance can be found in Section 6.

A schematic view of SMT objects representing model specification, data and instances is shown in Fig. 3.

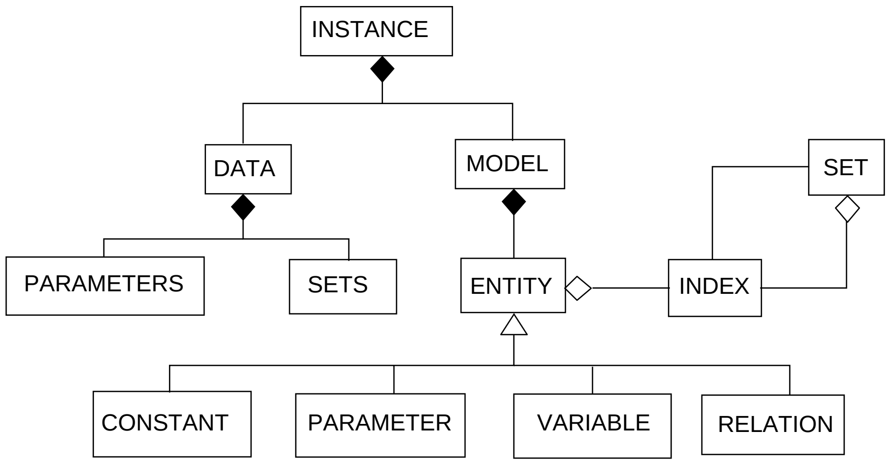


Figure 3: Objects within model specification, data and instance parts of SMT.

Finally, we stress that in SMT the definitions of model instances are stored in a symbolic way. Thanks to the persistency of all the data used in the modeling process, it is not necessary to actually store parameterized versions of instances. Such representations are generated only, when needed for computational tasks, and are removed from the system after the computational task does not need them anymore.

6 Instance analysis

Model instances are analyzed using diversified paradigms of model analysis. Each analysis of an instance provides a solution (from a typically infinite set of solutions) having certain properties. These properties correspond best to the preferences of the user. The key problem of any complex decision making is due to the fact that for a complex problem there is never a single adequate representation of user preferences. Thus the aim of model analysis is to support a learning process in which users actually learn about the properties of the modeled problem: diversified types of model analysis provide a variety of insights into the problem, and in this way support the process of learning about the problem, which eventually ends up with identifying a small set of solutions that are interesting for the user.

The structure of mathematical model utilization in decision making support is illustrated in Fig. 4.

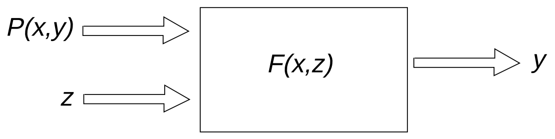


Figure 4: Structure of model usage for decision making support.

A model is a set of relations between variables, which are abstract representations of factors that need to be considered, in order to evaluate the consequences of a particular decision. Variables used for the model specification are grouped according to their role in the model, namely:

- decisions (model inputs): variables controlled by the user, denoted here by x,
- external or independent variables: model inputs being out of the user's control, denoted here by z,
- outcomes (model outputs), used for evaluation of implemented decisions, denoted here by y,
- auxiliary variables, which are defined for various reasons by model developers but are usually of no interest for model users.[2]

The model user is mainly interested in the relations between decisions x and the corresponding consequences measured by outcomes y. Thus the solution part, that a model user is focusing on, is composed of a pair of vectors (x, y).

Not all of the solutions are worth analyzing in a specific decision-making problem, especially in a typical case when the number of all possible decisions is infinite. For

[2]Therefore auxiliary variables are not discussed in this paper.

example, a very simple linear regression $y = ax$, with a parameter a, and even constrained domain of x is characterized by an infinite set of (x, y) solution pairs.

The aim of model analysis is to help the user identify a manageable set of solutions for more detailed consideration. The selection of such solutions is based on exploiting the concept called preferential structure $P(\cdot)$, which induces partial ordering of solutions.

Each analysis task utilizes a given representation of the user's preferential structure, which in turn is used for the definition of a mathematical programming task. The solution of such a task provides values of decision variables. The implementation of decisions, that have been found, results in values of outcome variables that correspond best to the given representation of the user's preferences.

Below we outline various types of analysis and the associated representations of the preferential structure.

6.1 Types of analysis

Here the types of analysis for algebraic models will be presented, to provide a background for the later discussion of the organization of the analysis process.

As illustrated in Fig. 4 preferences are in general expressed as a function of decisions and outcomes. A selected realization of a representation of preferences $P(x, y)$ combined with a definition of the model instance is used to define a computational task. Such a task should[3] provide a solution that best corresponds to a selected realization of preferences $P(\cdot)$.

We give an outline here of the most commonly used types of analysis:

Simple simulation: The preferential structure $P(\cdot)$ is defined by given values of decision variables (thus it does not concern the outcomes of a model):

$$x = \hat{x} \tag{2}$$

where x is a vector of decision variables, and $\hat{x}$ are the corresponding desired values.

However, for non-trivial models it is rather difficult to specify values of $\hat{x}$ that are feasible (i.e., do not cause infeasibilities of model's constraints).

Soft simulation: Is a simple but useful modification of the standard simulation that uses $P(\cdot)$ in the form:

$$\min |x - \hat{x}| \tag{3}$$

A corresponding optimization problem always has a solution.[4] If $\hat{x}$ is feasible, then the soft simulation will provide the same result as the simple simulation

[3] Large or badly conditioned optimization problems typically have an infinite number of solutions with values of goal functions that differ less than an optimization tolerance. However, in such cases regularization techniques should be used to provide a unique solution; see (Makowski, 2001) for more details.

[4] This is assuming that the modeled decision problem has at least one solution. Such an assumption is obvious for any properly modeled decision problem (because such models are developed to analyze typically large sets of feasible solutions).

approach. However, since the feasibility of $\hat{x}$ is typically difficult to assess, therefore it is rational to use the soft simulation method in order to get a solution, also in typical situations when the specified, desired decisions $\hat{x}$ are not feasible. In the latter case the solution will contain the values of decisions x which are closest to $\hat{x}$ in the sense of the norm $|.|$. Thus the solution (for infeasible $\hat{x}$) may strongly depend on the selection of the norm (most typically weighted Chebyshev norms are used).

Inverse simulation: has a preferential structure analogous to that of soft simulation (3):

$$\min |y - \hat{y}| \tag{4}$$

The aim of using it is to find a solution that results in desired values of outcomes $\hat{y}$.

There are several other forms of simulation-type analysis. The reader interested in this type of analysis may want to consult (Makowski and Wierzbicki, 2003).

Single criterion optimization: is the most often used analysis type. It assumes that one of the outcome variables is the criterion that is optimized[5] and optional bounds are placed on other outcome variables. For single criterion optimization the $P(\cdot)$ takes the form (5) with optional bounds defined by (6).

$$\min y_{goal} \tag{5}$$

subject to:

$$l_p \leq y_p \leq u_p, \qquad \forall p \in P \tag{6}$$

Optimization of the goal function (5) subject to bounds (6), combined with the relations defined by the substantive model, should[6] result in a unique solution.

Parametric optimization is used to define a set of single criterion optimizations. For instance in an environmental model, while analyzing cost-effective strategies for improving air quality, maximum allowed emission levels may be changed to examine effects of diverse regulations. In such a case the goal function (5) can be defined by the outcome variable defined as the cost of reducing emissions, and different combinations of upper bounds in 6 can be set for outcome variables y_p that correspond to emission levels. From the user point of view it is convenient to define a single analysis task with parameterized bounds (9):

$$l_i \leq y_p \leq u_j, \qquad \forall i \in L_p, \forall j \in U_p, \forall p \in P \tag{7}$$

where L_p and U_p are sets of indices enumerating elements of sets of bounds (lower and upper, respectively) for y_p.

[5] In the follow-up presentation for the sake of simplifying it, we assume that the criterion is minimized. However, the same approach can be used for maximized or target-type (that minimize the distance to a given target value) criteria.

[6] See the note above about applicability of regularization techniques.

Note that such a compact representation, and an easy interface to defining the sets of bounds may result in possibly many optimization tasks. For example, a specification of 10 elements of upper bounds for 5 outcome variables results in 10^5 optimizations.

Presentation of other analysis types, including multicriteria model analysis[7] can be found e.g., in (Makowski and Wierzbicki, 2003). Granat and Makowski (2000) discuss in detail the methodology and a modular software for interactive specification and analysis of aspiration-based preferences.

6.2 Specification of preferences

Before discussing the implementation of software handling preferential structure representation, we illustrate the user interface supporting the specification of the preferential structure.

6.2.1 Interface to specification of parametric optimization

A SMT user is interactively guided through the forms in which he/she specifies the type of analysis, and all the elements needed for the specification of his/her preferential structure. There are many possible paths of such a process which is composed of easy to complete, elementary steps.

As an illustration, a very simple example is used, which has a preferential structure in the following form:

$$\begin{array}{ll} \min cost & \\ em_{AUSTRIA,SO_2} & \leq 11 \\ em_{AUSTRIA,CO_2} & \leq em^{mx}_{AUSTRIA,CO_2} \end{array} \tag{8}$$

where $em^{mx}_{AUSTRIA,CO_2}$ takes 10 values uniformly distributed between 6 and 10.

The goal is to minimize the outcome variable *cost* (which defines the overall costs of emission reduction). A simple bound on a maximum level of SO_2 emission in one country (Austria) is set, and a parameterized bound for CO_2 emission from the same country is defined.[8]

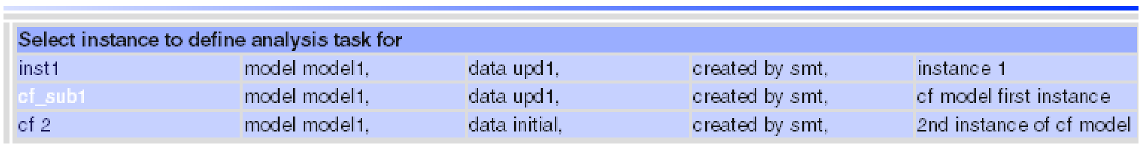
After a new analysis task option is selected, the whole process is organized into steps composed of:

- selection of the model instance for which the analysis task will be defined,
- definition of the task (identifier to be selected by the user, author and date inserted by SMT),
- selection of an outcome variable to be the goal function,
- selection of outcome variable(s) for which simple bounds will be defined,
- selection of outcome variables(s) for which parameterized bounds will be defined.

Fig. 5 and 6 on page 11 show the dialogs for the selection of the model instance (from the list of all model instances available to the user making the analysis), and selection of the analysis type, respectively.

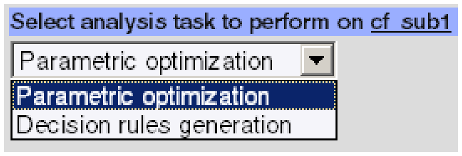
[7]In fact inverse and soft simulations are special cases of multicriteria model analysis.

[8]This example is built by extracting a small element of an actual parametric optimization, and is defined to make consecutive steps of analysis definition easier to follow. Thus one should not interpret it as an example of actual analysis.



Select instance to define analysis task for				
inst1	model model1,	data upd1,	created by smt,	instance 1
cf_sub1	model model1,	data upd1,	created by smt,	cf model first instance
cf 2	model model1,	data initial,	created by smt,	2nd instance of cf model

Figure 5: Selection of a model instance.

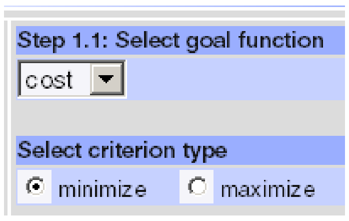


Select analysis task to perform on cf_sub1

Parametric optimization

Parametric optimization
Decision rules generation

Figure 6: Selection of analysis type.



Step 1.1: Select goal function

cost

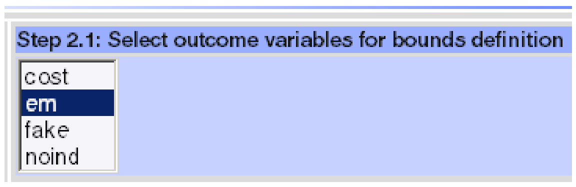
Select criterion type

[✓ minimize] maximize

Figure 7: Selection of an outcome variable to serve as the goal function.

Specification of user preferences starts with a choice of an outcome variable (*cost*) which will be used as a goal function (see Fig. 7). From the same dialog the user selects the type of optimization (minimization or maximization). Since the variable *cost* is not indexed, the dialog for selecting values of indices will be skipped.

The next step is definition of simple bounds (i.e., lower and/or upper constraint for a selected outcome variable).[9]



Step 2.1: Select outcome variables for bounds definition

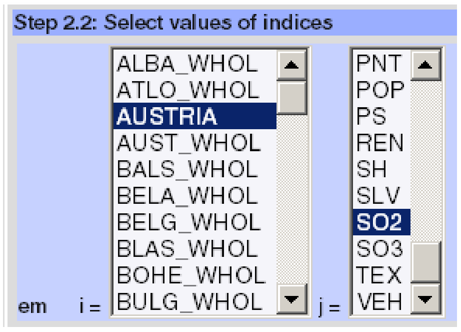
cost
em
fake
noind

Figure 8: Selection of an outcome variable to be bounded.

In the example shown in Fig. 8 variable *em* is selected from all of the outcome variables (automatically extracted from the model specification and displayed in the

[9]Note that all entities of the model have lower and upper bounds symbolically defined during the model specification. Thus the actual values of the bounds are defined by the model parameters, which in turn are defined by a chosen update of data used for the definition of the analyzed instance of the model. Therefore, defining additional (more tight) bounds within a specification of preferences simply supports the analysis of trade-offs between outcome variables, which is a part of the traditional, single-criterion optimization paradigm of model analysis.

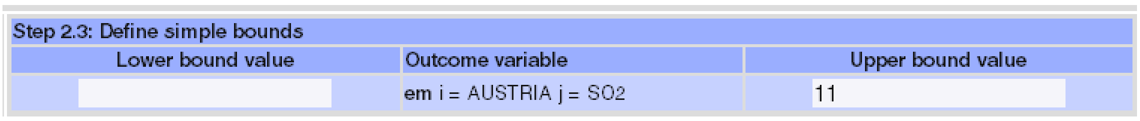
choice list).


Step 2.2: Select values of indices

em	i = ALBA_WHOL, ATLO_WHOL, AUSTRIA, AUST_WHOL, BALS_WHOL, BELA_WHOL, BELG_WHOL, BLAS_WHOL, BOHE_WHOL, BULG_WHOL	j = PNT, POP, PS, REN, SH, SLV, SO2, SO3, TEX, VEH

Figure 9: Selection of values of indices for the variable selected to be bounded.

The variable is indexed, therefore the user is guided to the next dialog (shown in Fig. 9), in which he/she selects (out of the choice list generated by the query defined by SMT using the information about the chosen update of data) values of indices.

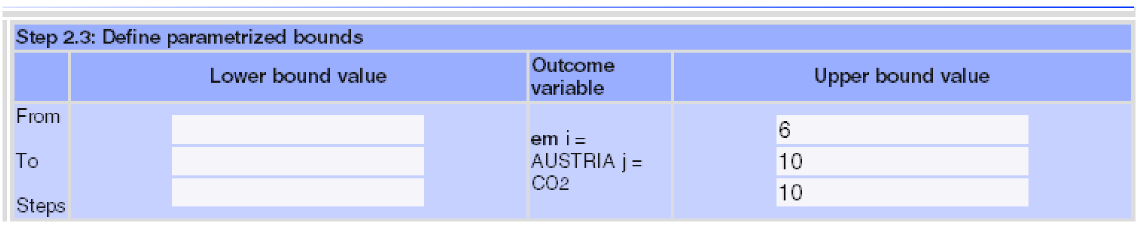

Step 2.3: Define simple bounds

Lower bound value	Outcome variable	Upper bound value
	em i = AUSTRIA j = SO2	11

Figure 10: Definition of the value of a simple bound.

In the presented example values of indices SO_2 and *Austria* have been selected. The final step of simple bound definition is accomplished by defining the value of the upper bound for the selected variable (equal to 11) in the dialog illustrated in Fig. 10.

Note that no optional lower bound is defined for this variables therefore the corresponding field in the form is left empty.


Step 2.3: Define parametrized bounds

	Lower bound value	Outcome variable	Upper bound value
From		em i = AUSTRIA j = CO2	6
To			10
Steps			10

Figure 11: Definition of parameters (range and number of steps) of a parameterized bound.

The process of specification of parameterized bounds is similar to that of defining simple bounds. In the last equation of (8) the parameterized bound with 10 values of maximal level emission of CO_2 in Austria is defined. The selection of an outcome variable (in the example $em_{AUSTRIA,CO_2}$) is done in a simple way. The specification of parameterized bounds is supported by the dialog illustrated in Fig. 11 in which each parametric bound is defined by a triple, composed of the starting and ending values of the interval, and the number of steps to be made within this interval.

All the information collected during the analysis definition stage is stored in the database in intermediate form. It can be viewed, edited, copied or deleted any time by any user who is authorized to access a specific definition. When the definition is considered to be completed the user can lock the task, which means the task is no longer editable. Then the corresponding tree structure is built up in the SMT database[10], and the definition of the preferences is available in a form suitable for generating the computational task(s).

However, before we discuss (in Section 6.3) the implementation of the computational tasks' management, an overview of the actual implementation of the preferential structure handling will be presented.

6.2.2 Implementation of instance analysis

The data structure and processing are hidden from the user. However, we summarize them here for those readers who might be interested in the implementation. Analysis task with the most complex definition of the preferential structure, namely parametric optimization is used for the presentation of the implementation issues.

Data structures for storing all the information on the analysis, including the whole pathway from type specification, through preferential structure to definitions of computational tasks are to be designed and created at the beginning.

Structuring a representation of preferences is crucial for proper design of the module responsible for the maintenance of the analysis process. For all types of model analysis and the corresponding forms of the preferential structures, the elementary preference item (or basic block of preferential structure) may be represented by a triple (*variable, role, value*). Here *variable* is either the decision or outcome variable, *role* indicates the role of the item in the preferential structure, and *value* contains optional value needed for some types of items. A variable's specification consists of a compound entity of a type variable which has selected values for all its indices. Its role depends on the analysis type. It may be a goal function, lower or upper bound, desired value of an outcome or decision variable, etc. Variables which have specific roles (e.g. lower/upper bound), must have additionally values assigned.

Such items serve as elements from which a representation of the preferential structure is built. As will become clear soon, the structures for single criteria optimization and different types of simulation tasks can be defined using the same schema.

Basically to represent a definition of a parametric optimization analysis task out of items described above, one needs to implement functionality of nested loops, iterating over all values specified for every parameterized bound. Each loop is for one upper/lower bound. For instance, following two parameterized bounds:

$$\begin{array}{rcl} \text{1 to 7 in 3 steps} \ \leq & y_1 & \leq \text{6 to 10 in 3 steps} \\ & y_2 & \leq \text{10 to 11 in 2 steps} \end{array} \qquad (9)$$

[10]See Section 6.2.2 for details.

may be expressed in "loop form", with the use of basic blocks as follows:

```
for $lb_1$ in (1, 4, 7) do
    for $ub_1$ in (6, 8, 10) do
        for $ub_2$ in (10, 11) do
            ($y_1$, "is greater than or equal to", $lb_1$)
            ($y_1$, "is less than or equal to", $ub_1$)
            ($y_2$, "is less than or equal to", $ub_2$)
        end
    end
end
```

Algorithm 1: Nested loops of parametric optimization

To meet the requirement of being able to uniquely identify every single definition of the optimization task (that corresponds to a realization of the most inner loop of Algorithm 1) it was decided that every such realization would have a corresponding item in the SMT database.

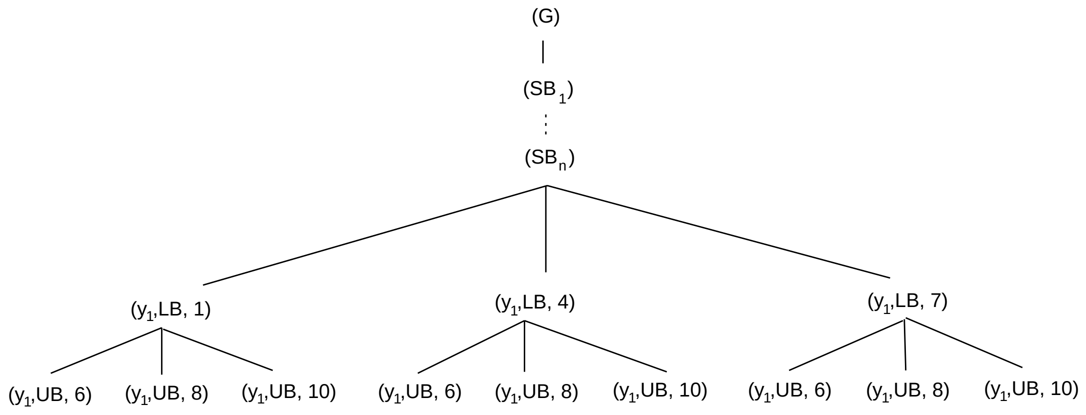


Figure 12: A tree of parameterized bounds.

Fig. 12 shows the physical tree representation of nested loops for first bound defined in (9). A path starting from a leaf and going up to the root has one-to-one correspondence to the most inner loop instantiation.

In Fig. 12 G stands for a triple representing the optimization goal, SB_i are simple (non-parameterized) bounds. Parameterized bounds are depicted in more detail, with upper bound (UB) and lower bound (LB) roles assigned. It is worth noticing, that some of paths in a tree may be excluded from the list of optimization tasks as they are for sure infeasible (e.g.the path from the root to the leftmost leaf of the rightmost lower level subtree, with $y_1 \geq 7$ and $y_1 \leq 6$). Those paths are therefore omitted when the tree is built up.

In order to keep the system portable, none of the DBMS vendor specific extensions for tree structure implementation was used (e.g., *connect by* clause of Oracle SQL). Instead, representations of preferential structure trees are based on the adjacency list model, developed in the graph theory field. Each tree node, except for the root, simply holds a reference to its parent node.

Additional remarks may be made on avoiding redundancy in the tree structure. Even in a very simple case, as the one illustrated in Fig. 12, at the second branching level all preference items are repeated in a way, where the same set of children exists for each upper level node. In order to avoid undesired repetitions of items, the concept of structure-content separation was utilized. Tree structure and (*variable, role, value*) triples are stored separately, with pointers relating nodes to items logically assigned to them.

6.3 Management of computational tasks

As mentioned above, a preferential structure representation of an analysis task is generated in the database in the form of a tree. Each leaf of the tree corresponds to a single computational task. Computational tasks are called runnable tasks (*RTasks*) as each of them results in one execution of a solver (which is a software specialized in solving a specific type of mathematical programming problem).

Every computational task is executed by a software package which organizes a sequence of the needed computations. Such a sequence is typically composed of the following software execution:

- Generator of a mathematical programming problem. Generators are specialized in selected types of computational tasks. A generator takes the ID of RTask as input, and reads from the DBMS all the information necessary to generate the task in a format required by a selected solver. Based on the corresponding definitions of the model instance and the preferential structure, appropriate input for a solver is generated (e.g., MPS or LPDIT format for LP solvers).
- Solver of the mathematical problem.
- Postprocessor which converts the results provided by the solver into the form that can be loaded into the SMT data warehouse. Results are then stored in the warehouse together with an ID of RTask, which makes it possible to link them to the appropriate model instance and a realization of user preferences. It, in turn, allows to regenerate any piece of data used, for any type of analysis, and any specification of user preferences.

Fig. 13 on page 16 provides an overview of the analysis process from the model instantiation to the results associated with various specifications of user preferences. A simplified UML diagram of objects supporting the model analysis phase is depicted in Fig. 14 on page 16.

One of the open problems of management of a large number of computational tasks is their automatic scheduling for distributed computations. RTasks are initially grouped according to the analysis task they belong to. However, a more sophisticated grouping might be more efficient, e.g., based on estimated demand for computational resources. A more sophisticated (than the currently implemented based on the FIFO[11] principle) scheduler could assign priorities using various criteria specified by users, administrators of groups of users, and by system administrators. Such priorities could then be used for scheduling computational tasks.

[11]First in, first out.

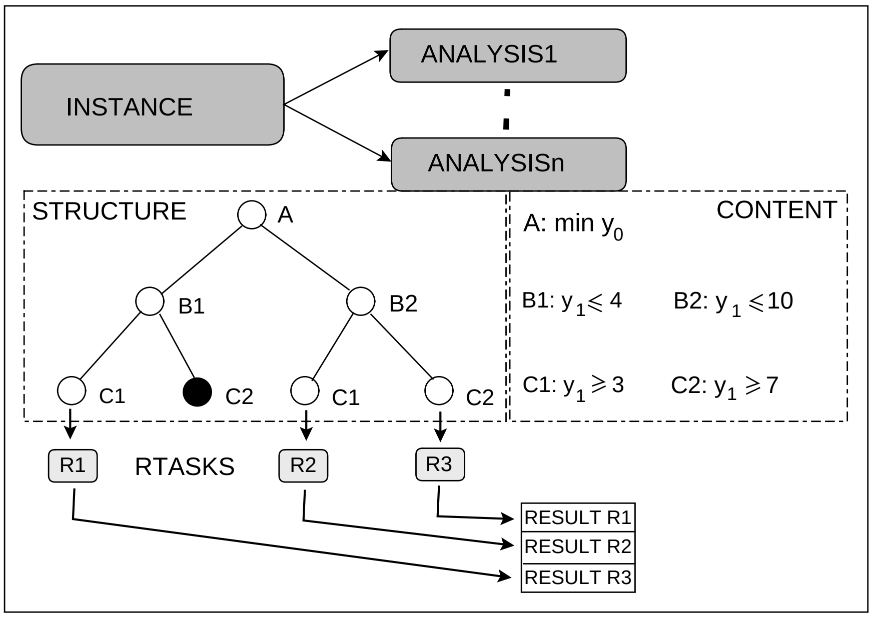


Figure 13: Components of the analysis process (from definition of instances to results).

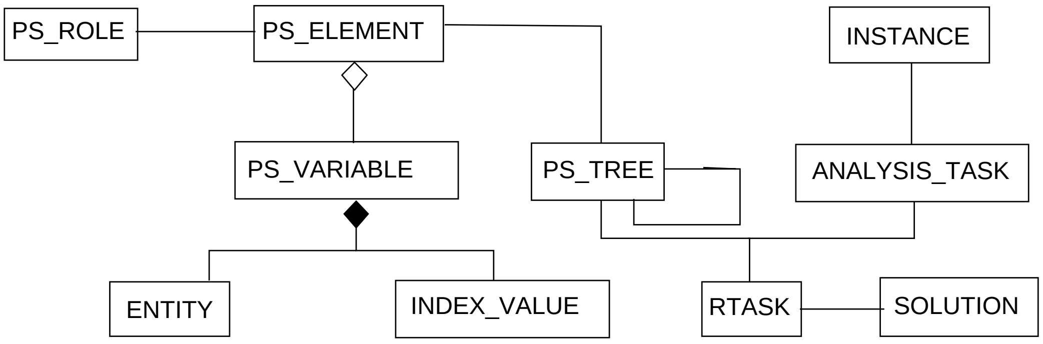


Figure 14: UML representation of objects supporting analysis.

Finally we want to stress that iterative analyses, like parametric optimization, deserve special attention in the context of computational resources. As discussed above it is easy to generate a large number of optimization tasks, say 10^5. However even if one computational task processing (composed of a generator-solver-postprocessor sequence) takes only about one minute, the wall-clock time needed to complete such an analysis on a single-processor computer would be counted in months. Thus a distribution of computational tasks over a computational Grid is necessary for analyses of this type.

6.4 Management of the analysis' results

Data warehouse structures for storing the results are generated from the model specification by the SMT system automatically. For algebraic models the results are stored in tables which have structures designed for optimizing space and access time (which is the primary concern for a big number of analyses of large models). Proposed data structure allows for flexible specifications of elements of results that are actually stored (e.g., primary and dual solutions). Moreover, the same set of tables is used for all possible instances and analysis types of a given model. Therefore, the RTask ID is stored with each element of results to allow for an effective retrieval of solutions corresponding to a given analysis.

Thus an efficient results' management for a large number of complex models' analyses has been implemented. However, another challenging problem still needs to be addressed. Namely, providing support for the analysis of large result sets. For this purpose at least two types of methods and techniques may be applied:

- various types of reports generated on demand, which should include also a diversified graphical representation of selected results,
- data mining for discovering knowledge represented by selected results.

These issues will be addressed in the future stages of SMT development.

7 Conclusions and further steps

This report gives an overview of the current stage of the SMT framework development. The design and structure of the SMT part handling various types of analyses of substantive models is presented in more detail. All system components representing elements of algebraic models are outlined along with interrelations between them. In particular, issues of data warehouse design, especially of saving the storage space occupied by values of model parameters, set enumerations, and results of instance analysis are presented. Distinct features of SMT, namely persistency of data and application of a data update hierarchy are also discussed.

The design of the model analysis module has been driven by the needs of parametric optimization, which is the type of analysis that requires the most complex data structure, which also needs to be efficient for large models. Such a structure can also effectively support simpler types of analysis, like classical optimization or different types of simulation. Thus one consistently implemented framework can support diversified types of complex model analysis.

Preferential structures can be specified easily: users can define preferences for various types of analysis in quite a straightforward way, and are guided through simple dialogs which are automatically organized for each selected type of analysis, according to attributes of selected variables. Thus all the complexity of the actual implementation is hidden from the users.

There are at least five challenging problems that still need to be addressed in order to improve the functionality of the SMT for large models:

- scheduling policies for computational tasks,
- exploitation of capabilities of grid environments,
- application of solvers that can exploit structures of large optimization problems, see e.g., (Fragnière, Gondzio, Sarkissian and Vial, 2000),
- efficient data-mining based support for analysis of large sets of results, see e.g., (Granat, 2003), and

- various (created on demand) views (including graphical visualization of large sets of multidimensional data) on the model parameters and results.

We briefly comment only on two of these problems:

- In recent years grid technologies have been developing very fast, see e.g., (Foster, Kesselman and Tuecke, 2001; De Roure, Jennings and Shadbolt, 2003; Expert Group Report, 2003). Thus, a possibility of exploiting capabilities of grid environments will be examined in the near future. Two aspects will be explored. First, as distributed computing is a must for a comprehensive and efficient analysis of large models, the capabilities of computational grids (including solvers exploiting structures of large optimization problems) will be utilized. Second, we plan to explore how semantic grids can be used for a better organization of modeling resources (composed of models, data, and modeling tools).
- Especially in the case of parametric optimization and complex algebraic models, where complexity arises from the scale of the problem, the results of an analysis will grow into a huge amount of information. Thus special attention has to be paid to knowledge discovery from data. The application of automatic data analysis with statistical and machine learning techniques is one of the next steps.

However all challenging problems listed above should be addressed in future versions of SMT.

References

De Roure, D., Jennings, N. and Shadbolt, N.: 2003, The semantic grid: A future e-science infrastructure, *Technical report*, Dept of Electronics and Computer Science, Southampton University, Southampton, UK.

Expert Group Report: 2003, Next generation grid(s), European grid research 2005-2010, *Technical report*, European Commission, Brussels.

Foster, I., Kesselman, C. and Tuecke, S.: 2001, The anatomy of the grid. enabling scalable virtual organizations, *International Journal of Supercomputer Applications* **15**(3), 200–222.

Fragnière, E., Gondzio, J., Sarkissian, R. and Vial, J.-P.: 2000, Structure exploiting tool in algebraic modeling languages, *Management Science* **46**(8), 1145–1158.

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

Granat, J.: 2003, Data mining and complex telecommunications problems modeling, *Journal of Telecommunications and Information Technology* (3), 115–120.

Granat, J. and Makowski, M.: 2000, Interactive Specification and Analysis of Aspiration-Based Preferences, *EJOR* **122**(2), 469–485. available also as IIASA's RR-00-09.

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, available from `http://www.iiasa.ac.at/~marek/pubs/prepub.html`.

Makowski, M.: 2004, A structured modeling technology, *EJOR*. (in press), draft version available from `http://www.iiasa.ac.at/~marek/pubs/prepub.html`.

Makowski, M. and Wierzbicki, A.: 2003, 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`.

Wierzbicki, A. and Wessels, J.: 2000, The modern decision maker, *in* Wierzbicki et al. (2000), pp. 29–46. ISBN 0-7923-6327-2.

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.