Working Paper

Software Tools for Generation, Simulation and Optimization of The Simplified Ozone Model

Piotr L. Zawicki

WP-95-107
September 1995

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 807 □ Fax: +43 2236 71313 □ E-Mail: info@iiasa.ac.at

Software Tools for Generation, Simulation and Optimization of The Simplified Ozone Model

Piotr L. Zawicki

WP-95-107
September 1995

International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 807 □ Fax: +43 2236 71313 □ E-Mail: info@iiasa.ac.at

Foreword

The research described in this Working Paper has been performed by a participant of the Young Scientists' Summer Program 1995 with the Methodology of Decision Analysis (MDA) project in a close collaboration with the Transboundary Air Pollution (TAP) project. The TAP project develops integrated assessment models for a systematic analysis of alternative strategies aimed at improving the air quality. One of those models which is currently being developed deals with the tropospheric ozone which is currently considered as one of the major air quality problems in Europe. The concentrations of ozone can be decreased by balanced reduction of emissions of nitrogen oxides and volatile organic compounds. The relations between emissions of those two pollutants and the corresponding concentrations of tropospheric ozone are nonlinear. Therefore for finding a cost effective strategy of ozone reduction one has to solve a non-linear optimization problem of a considerable size.

The 3 months research summarized in this paper was aimed at the development of software tool prototypes for generation and solution of the corresponding nonlinear optimization problem. The author has developed software tool prototypes for generation of the model in a form required by three non-linear solvers. The paper documents these tools and also provides a description of the underlying mathematical programming problem. The paper also summarizes a number of methodological and technical issues which are related to the model under consideration but which are also relevant to generation and solution of other large scale nonlinear problems.

Abstract

The paper presents an approach to the analysis of the Simplified Ozone Model (or: Integrated Assessment Model for Ozone) that describes the relationship between ozone (O_3) exposure and the emissions of ozone precursors, NO_x (nitrogen dioxide and nitric oxide) and $VOCs$ (volatile organic compounds). The model is currently being developed by the Transboundary Air Pollution project using data calculated by the EMEP Ozone Model which deals with ozone concentrations over long periods and covering the whole Europe. The model is to be used for analysis of different policies of emission reductions and their influence on O_3 concentration experienced by the receptors.

The final version of the model will be a large scale nonlinear programming model. Therefore software tools for the generation and analysis (by both optimization and simulation) of the model are needed. The paper presents the developed prototypes of the needed software tools. The tools include a model compiler which generates C++ code needed by any nonlinear solver for computing values of goal functions, constraints and the Jacobian (which is generated using a symbolic differentation tool). More general problems related to generation and solution of large scale nonlinear programming problems are also discussed and recommendations for further research are summarized.

Contents

1 Introduction 1

2 Problem Description 2
- 2.1 The Subject . . . 2
- 2.2 The Optimization Problem . . . 2
- 2.3 Definitions . . . 2

3 Model and Problem Formulations 3
- 3.1 Decision variables . . . 3
- 3.2 Model outcomes . . . 3
  - 3.2.1 Impact of constraints . . . 4
  - 3.2.2 Objective function . . . 5
- 3.3 Model parameters . . . 5
  - 3.3.1 Developer's parameters . . . 5
  - 3.3.2 User's parameter . . . 5

4 Model Analysis for decision support 6
- 4.1 Simulation . . . 6
- 4.2 Optimization criteria . . . 6
  - 4.2.1 Single criterion optimization . . . 6
  - 4.2.2 Multiple criteria optimization . . . 6
  - 4.2.3 Inverse and softly constrained simulation . . . 7

5 Discussion of results 7
- 5.1 Problems with the model compiler . . . 7
- 5.2 Problems with the solvers . . . 8

6 Software Overview 9

7 The Model Generator 10
- 7.1 Developer's Guide . . . 10
  - 7.1.1 The Algorithm . . . 10
- 7.2 The **gen** specification . . . 11
  - 7.2.1 The **gen** internals . . . 11

8 Solvers 13
- 8.1 Why different solvers ? . . . 13
- 8.2 Which solvers ? . . . 13
  - 8.2.1 MINOS . . . 13
  - 8.2.2 DIDASN++ . . . 13
  - 8.2.3 CFSQP . . . 13

References 14

A The syntax of the files with model's parameters **15**
A.1 File "params.1" 15
A.2 File "params.2" 15
A.3 File "emissions" 15
A.4 File "cost.nox" 15
A.5 File "cost.voc" 16
A.6 File "definitions" 16

B Mathematical Programming Problem **18**
B.1 Naming Convention 18
B.2 Mathematical Programming version of the model formulation 18
B.3 Columns (variables) 20
B.4 Rows (constraints) 20

C The UNIX environment **20**
C.1 Multi-models management 21

D Solver-specific issues **21**
D.1 MINOS 21
D.1.1 Interface with DIDASN++ 21
D.1.2 Problem specification 22
D.2 DIDASN++ 22
D.2.1 Problem specification 22

E Software Tools **22**
E.1 The model compiler 22

F An example of MINOS run **23**

Software Tools for Generation, Simulation and Optimization of The Simplified Ozone Model

Piotr L. Zawicki *

1 Introduction

This paper describes software tools which apply existing methods of analysis and optimization to the Simplified Ozone Model. The core of a decision making system was created using such methods and tools implementing them. This software can be changed almost automatically by the *developer*[1] if it is demanded by the *user*[1] in order to reflect changes to the model. The software is able to help in simulating a given model and in optimizing it (finding a solution to a problem defined by a user).

The Simplified Ozone Model[2] describes the relationship between ozone (O_3) exposure and the emissions of ozone precursors: NO_x (nitrogen dioxide and nitric oxide) and *VOCs* (volatile organic compounds) in the atmosphere. A detailed description of the ozone model can be found in [HSA95], its preliminary version is described in [HeS95]. The model is being developed by the IIASA's Transboundary Air Pollution project using data calculated by the EMEP Ozone Model[3] which deals with ozone concentrations over long periods and covering the whole Europe.

This model can be analyzed in various ways in order to predict the ozone concentrations resulting from different emission reduction policies. We can define some decision scenarios and examine their consequences. Using optimization techniques, it is also possible to find such decision scenarios that correspond to some goals or requirements. We may be able to find values of some variables (decision variables) which minimize (or maximize) the value of other variables (representing objectives or criteria).

The goal of the author's research at IIASA was to make a case study of this model, create tools necessary for the model optimization and look into any difficulties which might occur during that process. It was also to research and define their requirements and check their configuration in order to assure the accuracy of their results. A test version of the model was created. All of the tools were tested on this version and the optimization results were compared.

The structure of this paper is as follows. Section 2 describes the problem by explaining the subject of the model, the subject of this study and some definitions used in this paper. Section 3 presents the model formulation in its mathematical form. The mathematical programming version of these equations is given in Appendix B. Different possiblities of model analysis are discussed in Section 4. Sections 6 - 8 deal with the software tools used in the project. Those tools and programmer's environment are described further in Appendixes C and E. Appendix A defines the syntax of input parameters to the model generator.

*Participant of the Young Scientists' Summer Program 1995 at IIASA. Home institute: Institute of Control and Computing Engineering, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland.

[1] For the definition of *user* and *developer* see Section 2.3.

[2] or Integrated Assessment Model for Ozone

[3] The EMEP MSC-W photo-oxidant model [Sim92]

2 Problem Description

2.1 The Subject

The model deals with a certain number of emitters (sources of NO_x and *VOCs*) and a certain number of receptors (of O_3); both can be arbitrary chosen objects - countries or grids. Every emitter is a source of NO_x and *VOCs* (of course, some values of the emissions may be equal zero) as an effect of human activities and natural emissions. These emitted components are transmitted to different receptors. For each of the receptors, the ozone concentration is calculated while taking into account the NO_x and *VOCs* concentrations over them. Decision variables are defined as annual emissions from the emitter. The ozone concentration over the receptor is the mean of the daily maximum during the six summer month period. This part of the model represents the physical and chemical aspects and can be used to simulate ozone concentration as a result of different emissions. More will be discussed in Section 4.1.

One can also put constraints on these concentrations. In this case, of course if the maximal allowed concentration is low enough, one will have to decrease[4] the emissions to be able to fulfill these constraints. This means that an economical factor should be included because technologies of emission control are costly. Thus, cost functions also have to be provided. These functions are piece-wise linear, every segment of them reflecting marginal costs associated with the technology used to reduce the emission.

2.2 The Optimization Problem

The resulting optimization problem is a large scale NLP (nonlinear problem). In order to use any nonlinear optimization solver one has to supply it with not only functions describing the model, but also with function derivatives (the Jacobian matrix) as well[5]. In order to prepare the latter for such a large model, an automatic differentiation tool must be used. The first version of the model deals with 52 emitters and 25 receptors (countries). In this test approach the nonlinear part of the model has 117 "nonlinear"[6] variables entering 25 nonlinear constraints. Thus the Jacobian matrix has 2,925 entries. The number of linear constraints is greater then 1,000. However, the final version will deal with over 700 hundred grids which makes the model 20 times larger and complicates the task for the solver.

2.3 Definitions

Within this paper we use various names that might have ambiguous meaning. The following are their definitions:

model is a file[7] containing equations' and variables' definitions; this file describes the "core model" which is created by the developer and is constant during the interaction with the user.

compiled model denotes the model files compiled with the model compiler and prepared to be used with a specific solver; this file is also created during the time of model development.

problem denotes the extension of the model by defining the objective function(s) and/or some bounds. It changes the set of feasible solutions into a set of accepted ones.

developer is a person (persons) who is preparing the system for the user; he can change the model and/or prepare it to cooperation with solvers.

---

[4] Compare with the discussion of the model in Section 3.2.1

[5] Every nonlinear solver can approximate these derivatives numerically when they are not defined. However such a approximation made by the finite differences method may be too inaccurate for some models.

[6] Clearly, a variable can not be nonlinear. However, a variable entering nonlinearly a constraint or objective function will be called here "nonlinear" variable.

[7] While it might denote the mathematical form of the model, we use it mostly in the sense of computer software.

user is a person who will use the system during the decision making process; the model specification is constant for him but during the process of decision making he can define or change the problem and its parameters.

solver is a specific software, used to solve optimization problems defined by the user.

MPS file is a file in MPS format; this format is used to specify linear models for solvers. For the description of this format consult [MuS87].

MDF[8] file is a file in DIDASN++-specific format; this format can describe both linear and nonlinear parts of the model. For the description of this format consult [GKPS94].

3 Model and Problem Formulations

As mentioned above, the model deals with a set of emitters and a set of receptors. These two sets can (but do not have to be) equal. In this section, $i \in [1, N]$ corresponds to emitters and $j \in [1, M]$ corresponds to receptors.

3.1 Decision variables

Decision variables in the model are annual emissions in each of the emitters.

n_i – annual emission of NO_x in i-th emitter

v_i – annual emission of $VOCs$ in i-th emitter

3.2 Model outcomes

Model outcomes are used to define either constraints or objectives. Equation (1) gives the ozone concentration in j-th receptor.

$$o_j = k_j + \sum_{i=1}^{N}(a_{ij}v_i + b_{ij}n_i + c_{ij}n_i^2) + \alpha_j en_j^2 + g(en_j, \mathbf{v}) \tag{1}$$

where $\mathbf{v} = \{v_1, \ldots, v_N\}$.

The nonlinear function g can be in one of two versions (2) or (3):

$$g(en_j, \mathbf{v}) = en_j \sum_{i=1}^{N} d_{ij}v_i \tag{2}$$

$$g(en_j, \mathbf{v}) = \beta_j en_j ev_j \tag{3}$$

The mean effective emissions transmitted to the j-th receptor are given by:

$$en_j = \sum_{i=1}^{N} e_{ij}n_i + enn_j \tag{4}$$

$$ev_j = \sum_{i=1}^{N} f_{ij}n_i + evn_j \tag{5}$$

For the current version of the model it was decided to use the function g defined by equation (2). Equations (3) and (5) were not used. Therefore, the evn_j parameters were not needed.

The following equations will be used in place of equations (1)-(5) in order to simplify the notation.

[8] Model Description File.

$$o_j = k_j + \sum_{i=1}^{N}(a_{ij}v_i + b_{ij}n_i + c_{ij}n_i^2) + \alpha_j en_j^2 + en_j ev_j \tag{6}$$

$$ev_j = \sum_{i=1}^{N} d_{ij}v_i \tag{7}$$

$$en_j = \sum_{i=1}^{N} e_{ij}n_i + enn_j \tag{8}$$

For every emitter, the cost associated with different emission levels (equal to the cost of emission reduction to levels n_i, v_i respectively) is defined as follows:

$$cn_i = PWL_{1i}(n_i) \tag{9}$$

$$cv_i = PWL_{2i}(v_i) \tag{10}$$

where PWL means a piece-wise linear function.

3.2.1 Impact of constraints

Let us now analyze some aspects of the equations (6)-(11) and their consequences. Looking at the equation (6) one can see that ozone concentration depends on the square of the NO_x emissions. To explain this phenomena we can quote [HeS95]:

> There is a further complication when NO_x concentrations are particularly high. [..] If, in these circumstances, the NO_x concentration is decreased, there will be a greater number of OH radicals available to react with the *VOCs*, leading to greater formation of ozone. Hence, a reduction in the atmospheric NO_x level may result in an increase in ozone concentration.

From a mathematical point of view, in some countries the solution (the reduction of ozone concentration) might be achieved by increasing the NO_x emissions. Of course, this mathematically correct solution, need not be accepted in the "real world". However, if it happens that the solver finds a point on the "decreasing" slope of the ozone curve, it would stop there and would not go in the opposite direction (this is the way nonlinear optimization solvers work: they stop at a local solution, not necessarily searching for the global one). Such a solution corresponds to the *maximization* of NO_x emissions (which, in turn *minimizes* the cost).

There are two possible approaches to this problem.

The first appears to be the most direct: it puts bounds on NO_x emissions in a way to prevent such behavior. Unfortunately, this is a non-trivial task. The ozone concentration depends, in fact, on effective NO_x (and *VOCs*) concentration at the receptor. It could be very difficult (if at all possible) to find bounds, which would satisfy all of the receptors. At the same time, most of the industrialized countries are already on the decreasing part of the curve. The bounds calculated this way could be either impossible or very expensive to achieve in these countries.

Second approach is simpler, although introduces some danger. As it is well known, the behavior of a nonlinear optimization solver depends very much on the starting point chosen. If we start from the point at which the emissions are minimal, we can be almostsure that the algorithm would find the solution with lower NO_x emissions. The danger in such approach is that it might be only a local solution, that is, that the other solution, although with higher NO_x emissions would be cheaper (and the emissions still lower than the current ones).

3.2.2 Objective function

If we perform model optimization, one of the most natural objective functions might be defined as a sum of all costs caused by the emissions' reduction.

$$cost = \sum_{i=1}^{N}(cn_i + cv_i) \tag{11}$$

This function is to be minimized. Naturally, other objectives of optimization could be also considered, if necessary.

3.3 Model parameters

All the parameters of the model were divided into two groups for simplicity called "developer's" and "user's" parameters. We can also treat them as parameters to the model and to the problem.

3.3.1 Developer's parameters

These are parameters which are set during the developing stage of the model analysis. They reflect the constant relations between variables and equations. They can be treated as a part of the core model formulation, from the user's point of view. On the other hand, the developer may be forced to change them from time to time; for example if the EMEP model is changed. Among them are:

- Transfer coefficients from equation (6): a_{ij}, b_{ij}, c_{ij}.
- Transfer coefficients from equation (7): d_{ij}
- Transfer coefficients from equation (8): e_{ij}
- Parameters of nonlinear term in equation (6): α_j
- Constants for equation (6): k_j
- Parameters to the PWL functions (9) and (10): PWL_{1i}, PWL_{2i}

3.3.2 User's parameter

In this second group we put parameters which are to be defined for the *problem.* They are not used by the developer because there is no need to deal with them during the model formulation. It is the user who has to supply them in order to be able to start the optimization-based phase of the analysis. We can name the following:

- O_j - Maximal ozone concentrations experienced by receptors. These values are needed to set the upper bounds on equations (6).
- $\overline{N}_i$ maximal and $\underline{N}_i$ minimal NO_x emissions in i-th emitter. If they are not given, their values can be deducted from the value of parameters of the last segment of the piece-wise linear cost function.
- $\overline{V}_i$ maximal and $\underline{V}_i$ minimal VOCs emissions in i-th emitter. If they are not given, their values can be deducted from the value of parameters of the last segment of the piece-wise linear cost function.

4 Model Analysis for decision support

In this section possible areas of usage of the Ozone Model will be discussed. Most of the experiments described in this paper are restricted to simple simulation and single criterion optimization, although the software is also designed for multiple criteria and inverse and softly constrained simulation.

4.1 Simulation

Simulation is used for running a model in so-called descriptive mode, which means that we can ask questions such as: *What will happen if ....* The purpose of this is explained in [Mak94]:

> this technique is good for exploring intuition of a decision maker (DM), not only for verification of the model but also for providing a DM with a consequences of applying certain decisions (for example, what would be the value of goals and constraints). One can also consider simulation as an alternative-focused method of analysis that is oriented to identify (examine) the alternatives.

In our case we can set different NO_x and *VOCs* emissions and compare their consequences (changes in ozone concentrations and of coursethe cost of this operation). To do it a program **deb** being a part of the DIDASN++ package can be used. This tool lets the user specify values of all input parameters (variables) and based on this data calculates the values of outputs (constraints and goal functions). Alternatively, some kind of other software can be used. This software would use the compiled model (which, in fact, is a C++ function) by supplying it with an array of arguments (inputs) and using its calculated results (outputs). More information about this software can be found in Appendix E.1.

4.2 Optimization criteria

Thus far only one criterion has be examined, namely minimization of the overall costs. However, we briefly outline below the other techniques that could be used during the model's analysis.

4.2.1 Single criterion optimization

Optimization means asking questions in form *What decisions are likely to be the best for....* Let us again quote [Mak94]:

> Optimization can be considered as a goal-oriented (value-focused) approach that is directed towards creating alternatives. Optimization is driven by hope to reach a set of goals (objectives). Therefore goals are a driving force and the values of decision variables are outcomes.

Therefore, it is only single criterion optimization, we set only one goal – to minimize the costs.

4.2.2 Multiple criteria optimization

Single criterion optimization assume that only one criterion is explicitely selected. This results in treating other criteria as constraints for which values have to be specified. However, it is often desirable to treat all criteria similarly. In that case one could apply methods developed for multi criteria model analysis. One of methodologies of multi-objective analysis and optimization is the aspiration-based technique which uses the maximization of an order-consistent achievement function as a method of aggregating multiple objectives and of interacting with the decision maker.

4.2.3 Inverse and softly constrained simulation

Inverse simulation can give the decision maker another possibility. In this approach model's outcome variables are assumed and we check if it is possible to achieve them using incomes inside allowed bounds. The generalized version of the simulation is described in [WiG96]

> **Generalized inverse simulation** consists in specifying also some reference decision $\overline{x}$ and in testing, whether this reference decision could result in the desired outcomes $\overline{y}$. [..] An aspiration-based multi-objective optimization system can clearly help in such inverse simulation, in which case we stabilize all outcomes and decisions of interest and use partial achievement functions [..] for such stabilized objectives to define an overall achievement function.

Another system function can be simulation with elastic constraints (elastic simulation or softly constrained simulation). The idea is to distinguish between hard constraints which can **never** be violated – such as physical laws, balance equation – and soft constraints which represent only some desired relations and are in fact additional objectives with given aspiration levels.

5 Discussion of results

The nonlinear optimization problems we deal with are large. The size causes problems in different stages of the project. Most of them have been solved but some of them remain unsolved. This section lists and explains these problems.

5.1 Problems with the model compiler

The model compiler is one of the most important components of the system. It is the model compiler which prepares functions to be used with a solver. That is why its (correct) results are critical for the model analysis. The main problem related to the model compilation is the source code of the compiled model. The size of the tested model description file, containing both the linear and the nonlinear parts was 162,830 bytes. It took less then one minute to compile the model. The size of the compiled model (a C++ code of two functions) was 1,585,452 bytes in 57,236 lines. Such a large C++ file was very difficult to compile: it took over 2 hours to do it on the Sun Sparc 10 workstation running SunOS 5.3 and it caused an "Out of memory" error on less powerful machines. Such a large code can not be accepted for the following reasons:

- The code for the currently examined model is too large and the compilation time is too long.
- The final model will be approximately 20 times larger. Therefore the corresponding C++ code (generated with the currently available tools) will have the size over 30 MB.

The solution to the problem lies in the design and implementation of the model compiler (which has been originally designed for small size problems). Its structure has to be changed in order to decrease the code generated. It is possible because this software is still under development. The following ideas have been suggested and some of them are now being implemented.

- The model compiler should treat the linear and nonlinear parts separately. It would allow to simplify the generated code. The present version of the DIDASN++ package does not distinguish between them. Every equation is treated as nonlinear. The code for calculating the derivatives for linear equations is much simpler and smaller.
- In models like ours, most of the equations are created based on one or more templates. Thus, the code calculating their values and the elements of the Jacobian matrix could be very similar and only the parameters would change. If the model compiler accepted the equations in form (ie., by indices) allowing to point out such similarities, then the generated model would be simpler.

Another technique had to be used in order to decrease the size of the code because the size of the model is still to large. Another tool, compressing the generated code, has been developed by the authors of the DIDASN++ package as a temporary solution to such problems. This code, the source code compressor, is able to decrease the size of the code up to 3 times. There are two main drawbacks of this method:

1. The tool is still being developed.
2. The size of the final version of the model can be still estimated (after compression) at 10 MB.

The other problems connected with this tool are only a matter of bugs in the software and could be quickly improved. The model compiler should be able to accept the model description in two parts - the nonlinear part as an MDF file and the linear part as a MSP file. This is not true in the present version. Therefore, an MDF file containing only nonlinear equations is prepared. The linear constraints are defined by an MPS format file. On the other hand, the MDF for DIDASN++ contains both linear and nonlinear equations. This should be improved as soon as possible because the approach implemented by MINOS has obvious advantages for problems with large LP part.

5.2 Problems with the solvers

The compiled model was used by solvers. Usually, this was achieved by linking the compiled model's code with the libraries supplied by the solvers. The MINOS solver requires also a short portions of Fortran subroutines and a file with the problem specification. The problem is an instance of the model with the objective function defined and optionally modified bounds. Generally, bounds and initial values are the only data that are subject to modification for the problem definition.

There are various ways of defining problems for different solvers. MINOS accepts only the MPS file and the specification file. The specification file defines different parameters which are needed by the solver when it reads the MPS file such as the memory requirements. The objective function and the type of optimization (maximization or minimization) are defined in the specification file whereas the initial values and bounds are defined in the MPS file. They are detailed in Appendix D.1. The DIDASN++ solver requires a special file in its own format. This file is prepared by the **pro_win2** program from the package and read by the solver. (See Appendix D.2 for further information.) One must provide the possibility to generate a problem in a format required by a particular solver.

Optimization tests were completed using the two solvers mentioned above. Some parts of the data used in the model (for example the cost functions for the *VOCs* emission) were invented only for the purpose of the study because of lack of real data. Therefore, the actual numeric results have little meaning and are not presented in this paper.

The first tests were done with the MINOS solver. The following topics were put under observation:

- whether the optimal solution was found
- the time of the optimization

It occurred during different tests that for some problems, specification of a large number of minor iteration steps led to reported infeasibilities of the linearized subproblems (although the problem was feasible). It was solved by the "miss or hit" method of setting the number of iterations. This topic requires further investigation aimed at automatic setting of parameters for MINOS.

Some tests with the DIDASN++ solver were also done. Unfortunately, only small models could be optimized. DIDASN++ worked correctly with both of them for which the generated

code was not too large. The "large" can not be precisely defined because it was the C++ compiler which had problems and depending on the state of the system resources some stages of compilation succeeded or failed. It turned out that in order to be able to run model larger then several hundreds of equations with DIDASN++, the model compiler must generate *much* smaller code (as stated in Section 5.1).

The other problem was caused by the size of the data structures used by the solvers. Although in the case of MINOS it was not a problem, because the Jacobian matrix was only 5,726 elements (the nonlinear part only), both DIDASN++ and CFSQP the size of the matrix should be 120,267[9] elements. Needless to say, such a large matrix would be very difficult to manage. On the other hand, the matrix is very sparce because of the large linear part.

6 Software Overview

The software used for the Simplified Ozone Model analysis consists of several different modules. Some parts of the systems are already existing pieces of software (mainly optimization solvers), some are in the testing phase (DIDASN++), and others were created especially for the model (model generator). Figure 1 shows their dependencies.

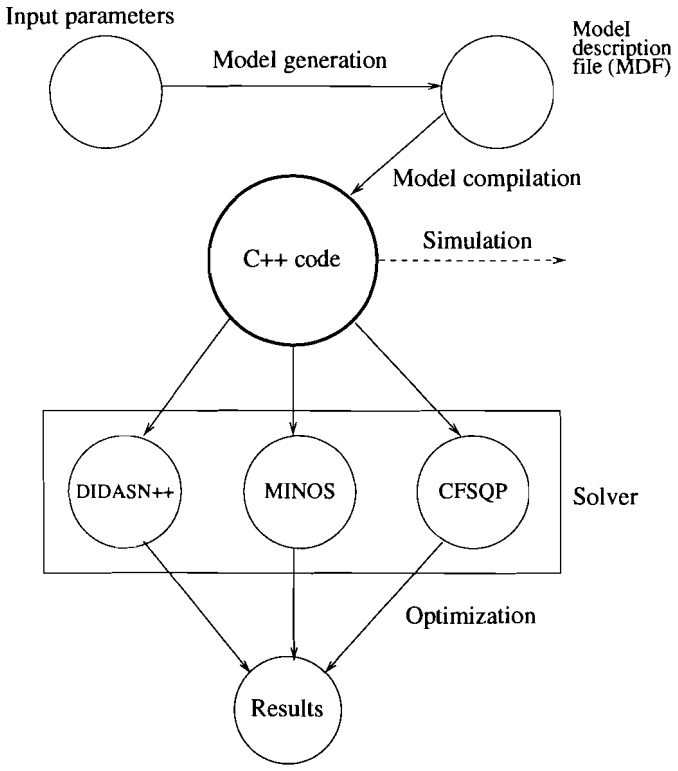

Figure 1: Software overview

The following tools can be used for the ozone model analysis.

Model generation realized by the Model Generator. The Model Generator was written especially for the model.

Model compilation done with help of the DIDASN++ package. The **edi** program from the package translates the model description into the compiled model form. This compiled

[9] 83 variables * 1,449 equations (because in this case some of the variables can be reduced).

model, which is a C++ function, can be linked to any other program that creates a user interface to the model.

Simulation can be performed by the **deb** program from the DIDASN++ package.

Problem generation The problem definition can have different forms while using different solvers. For instance for MINOS it is described as a part of the MPS file and for DIDASN++ it is a binary file prepared by another component of the package.

Optimization Three solvers have been considered for this case study. Their descriptions can be found in Section 8.

7 The Model Generator

The model generation uses the user-supplied model parameters and hard-coded model formulation in order to prepare the model description files. The following files are generated:

- An MPS file describing the linear part of the model
- An MDF file with nonlinear formulas. This file is then compiled by the model compiler (a part of the DIDASN++ package) in order to create a source code calculating the function's values and their gradients. Any nonlinear solver needs such a code.
- Solver specific files, at present include:
  1. The specification file for the MINOS solver;
  2. The interface between DIDASN++-generated code in C++ and MINOS solver procedures which expects a Fortran code.

The model generator reads the parameters specified in Section 3.3.1. Those parameters describe the chemistry and physics of the model and are parameters to the equations shown in Section 3.2. These are:

- parameters to the ozone concentration part
- parameters to the cost functions
- the definition file, which allows to choose subset of the available data

The above mentioned data files have been output from external sources and combined into several ASCII-files. Each file can contain other sets of data. All of them allow comments for the sake of documentation. The list of files and their syntax is presented in Appendix A.

To generate the model the user has to

1. Create the model directory and model files as described is Appendix C.1.
2. Type `make model`

Generated files should be used in different ways for various solvers.

7.1 Developer's Guide

7.1.1 The Algorithm

The present version of the model generator, OzMoGen, consists of two parts corresponding to the stages of the model generation process.

Firstly, the data files are read and converted into files containing C++ code. Then the C++ program, which includes the abovementioned code is compiled by a C++ compiler. This process yields an executable program – **gen**.

Secondly, the **gen** program is run with different parameters in order to create different output files. Those files include model description in both MPS and MDF formats and some auxiliary files.

7.2 The gen specification

The **gen** program is created by the developer at the first stage of the model generation process. In its structure, both the model template and the actual parameters are hard-coded (included during the compilation). The only input parameter to **gen** is its command line. With this parameter the developer can determine which of the files he wants to create. **gen** sends its output (model description or an auxiliary file) to the standard output which, in turn, can be redirected into any file. **gen** is called by issuing the command

```
gen -[d|n|m|s]
```

from the model's subdirectory. We use several options to point out which of the files we want to create. Only one of the options can be used during one program execution. Currently, the OzMoGen accepts the following options and generates appropriate files:

d the model as an MDF file to be used with DIDASN++

n the nonlinear part of the model only; this file also is to be compiled by DIDASN++' model compiler

m the MPS file

s the SPECS file for MINOS

7.2.1 The gen internals

This section is intented to provide all the information necessary for the reader who would like to change the model structure tailored into the model generator.

gen is written in C++. It consist of one main file named **gen.cc** and a number or include files created in the first stage of model generation. The main file is placed in subdirectory **gen** of the **$OZON_HOME** directory while the include files in model's subdirectory. All the parameters defining the model are stored internally in static tables which are created when the generator is called. While the data is read during its compilation, it does not read any files during execution. Include files are pieces of C++ code which define these parameters. The number of emitters and receptors allowed (the size of the tables) is stored in one of the files as preprocessor's **#define** statements.

Each include file is created from one parameter file. For the syntax of the parameter files consult Appendix A. This is a list of the include files:

defs.inc this file is created from the "definitions" file. It contains six elements.

- #define RECEPTORS nn – which puts upper limit on the number of elements of all static tables in "row" dimension. The nn number is taken from the appropriate section of the "definitions" file.
- #define EMITTERS nn – similar to the previous definition, but this is a number of "columns". The nn number is taken from the appropriate section of the "definitions" file.
- definition (and initialization) of the **valid_constraints** table which specifies which receptors are taken into consideration in the model. This is defined in the CONSTRAINS section of the file.

- definition (and initialization) of the **valid_NOx_emission** table which specifies which NO_x emitters are included as variables in the model. This is defined in the VARIABLES section of the file.
- definition (and initialization) of the **valid_VOC_emission** table which specifies which VOCs emitters are included as variables in the model. This is defined in the VARIABLES section of the file.
- #define(s) of parameters set by the USE section of the "definitions" file. This is not used in this version of the model generator but may be useful in the future as the model generator becomes more complicated.

coeffs.inc contains the a_{ij}, b_{ij}, c_{ij}, d_{ij}, e_{ij}, α_{ij}, enn_{ij} and k_{ij} parameters. Each set is represented as a two-dimensional array. This file is built of a number of assignment statements and must be included inside the body of a procedure.

emission.inc contains the list of all current NO_x and *VOCs* emissions. It has a form of a list of assignment statements. The values are assigned to the elements of two vectors - **emission_n** and **emissions_v**. This file must be included into the body of a function.

cos_def.inc defines numbers of segments associated with all of the cost functions. It has a form of a list of assignment statements. The values are assigned to elements of two vectors - **NOX_parts** and **VOC_parts**. This file also must be included into the body of a function.

costs.inc contains parameters of the cost functions. Data from this file is to be fitted into arrays created dynamically (by the main program) according to data from **NOX_parts** and **VOC_parts**. The following tables are filled:

NOX_cos[i][j] – for NO_x; marginal cost for j-th segment of i-th emitter;
NOX_emi[i][j] – also for NO_X; the value of the emission, for which the cost is valid;
VOC_cos[i][j] – as NOX_cos but for *VOCs*;
VOC_emi[i][j] – as NOX_emi but for *VOCs*.

The algorithm is as follows:

1. Allocate memory, set all the tables' elements to zero (because include files do not contain coefficients which are zero). This is done in the main function.
2. Fill all the tables with data. It is done in **set_coefficients**, **set_emissions**, **set_cost_parts**, and **set_cost_fun** functions. Each of them include one of the include files.
3. Unnecessary equations are marked, which permits a decrease in the total number of equations.
4. Different command line options activate the corresponding actions.

   d – generate MDF file with both linear and nonlinear parts
   n – generate MDF file with nonlinear part only
   m – generate MPS file (linear part only)
   s – generate specification file

   In the case of **d** and **n** options, eventually three functions are called (with different parameters corresponding to the chosen option). These are:

   - gen_PARAMETERS – which generates the PARAMETERS section of the MDF file; this section is similar in both cases.
   - gen_VARIABLES – which generates the VARIABLES section of the MDF file.
   - gen_EQUATIONS – which generates the EQUATIONS section of the MDF file; this section includes only nonlinear or all equations.

8 Solvers

It is possible to use several different solvers with the generated model descriptions. At present, model descriptions for DIDASN++ and MINOS are generated. Since the linear part of the model can be presented as an MPS file and the nonlinear part can be translated by the model compiler into C++ file it is possible to use this code with other solvers. The only solver-specific problem is the interface between the solver and its "user function" this is because different solvers (different software written by different developers) expect various forms of the user defined function.

8.1 Why different solvers ?

When solving nonlinear problems it is very difficult to know in advance which solver will be the most appropriate for the task. This is why a few of them should be tested on this problem before an appropriate solver can be chosen. The word "appropriate" means in this context not only the one which gives correct results in reasonable time, but also the one which is admissible for using/distributing (for example due to copyright restrictions), which does not produce too many problems with setting its parameters, etc.

8.2 Which solvers ?

Currently, two solvers have been tested and third is prepared for testing. Each of them has its own advantages and disadvantages. The following sections outline their features.

8.2.1 MINOS

MINOS is a nonlinear solver developed at Stanford University. The 5.3 version of MINOS was used in this research. This version combines the following algorithms:

- the simplex method
- a quasi-Newton method
- the reduced-gradient method
- a projected Lagrangian method

MINOS was used as a world-wide spread solver commonly used for solving nonlinear problems.

8.2.2 DIDASN++

DIDASN++ is a software package that supports model compilation, problem definition and makes it possible to replace the DIDASN++solver with another nonlinear solver. In fact, many parts of the package were used during the preparation of data for other solvers (model compiler). The DIDASN++'s modules used in this project are listed in Appendix E. DIDASN++ uses a variation of the projected gradient method to solve the nonlinear problem.

8.2.3 CFSQP

CFSQP[10] is a solver for [LZT94]:

> the minimization of the maximum of a set of smooth objective functions (possibly a single one, or even none at all) subject to nonlinear equality and inequality constraints, linear and nonlinear constraints, and simple bounds on the variables. In addition CFSQP contains special provisions for efficiently handling problems with many sequentially related objectives/constraints, for example discritized Semi-Infinite Programming (SIP) problems.

[10] C code for Feasible Sequential Quadratic Programming

CFSQP solver is known as an efficient solver for large scale problems. However, the application of CFSQP for the Ozone model requires an efficient model compiler that produces a code of manageable size. Therefore, for the reasons explained above, this solver has not been used so far.

References

[GKPS94] J. Granat, T. Kreglewski, J. Paczynski and A. Stachurski, *IAC-DIDASN++ modular modeling and optimization system: Theoretical foundations*, Technical report, Institute of Automatic Control, Warsaw University of Technology, Warsaw, Poland, 1994.

[HeS95] C. Heyes and W. Schöpp, *Towards a simplified model to descripe ozone formation in Europe*, Working Paper WP-95-34, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1995.

[HSA95] C. Heyes, W. Schöpp and M. Amann, *A simplified model to predict long-term ozone concentrations in Europe*, Working Paper WP-95-xx, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1995. To be published.

[LZT94] C. Lawrence, J. L. Zhou and A. L. Tits, *Users's guide for CFSQP version 2.2*, Technical Report TR-94-16r1, University of Maryland, College Park, MD 20742, 1994.

[Mak94] M. Makowski, *Design and implementation of model-based decision support systems*, Working Paper WP-94-86, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[MuS87] B. A. Murtagh and M. A. Saunders, *Minos 5.1 user's guide*, Technical Report SOL 83-20R, Stanford University, Stanford, California 94305-4022, USA, 1987.

[Sim92] D. Simpson, *Long period modelling of photochemical oxidants in Europe. Calculations for July 1985*, Atmos. Environ. **26** (1992) 1609–1634.

[WiG96] A. Wierzbicki and J. Granat, *Multi-objective modeling for engineering applications in decision support*, in Proceedings of the Twelfth International Conference on Multiple Criteria Decision Making, Lecture Notes in Economics and Mathematical Systems, Springer Verlag, Berlin, New York, 1996. (accepted for publication).

A The syntax of the files with model's parameters

Every of the following files contains a number of lines. Each line consists of a number of fields (columns). They are separated by whitespace[11] characters. Every line which starts with '#' in the first column is ignored; every line which is empty is ignored as well. Column names are not allowed, therefore one can include them as a comment line.

A.1 File "params.1"

This file contains transfer coefficients. They are listed in Table 1. Consult Section 3.2 for their meaning. One doesn't have to include lines for which all 5 coefficients are equal 0.

Table 1: Syntax of the params.1 file

name	description	type
i	the code of the emitter	integer
j	the code of the receptor	integer
a	a_{ij}	floating-point
b	b_{ij}	floating-point
c	c_{ij}	floating-point
d	d_{ij}	floating-point
e	e_{ij}	floating-point

A.2 File "params.2"

This file contains the rest of the parameters. They are listed in Table 2.

Table 2: Syntax of the params.2 file

name	description	type
j	the code of the receptor	integer
k	k_j	floating-point
alpha	α_j	floating-point
beta	β_j[12]	not used
enn	enn_j	floating-point

A.3 File "emissions"

This file contains the current emissions of VOCs and NOx. They are listed in Table 3.

A.4 File "cost.nox"

This file contains the description of cost functions for NO_x.
The file is divided into sections; every section begins with a line with syntax as showed in Table 4.

Inside a section there are a number of lines describing all the points making the cost function. They are listed in Table 5.

[11] space or tabulation

[12] This column is reserved for future usage with equation (3). Currently it has to contain the "NA" string.

[13] should be the same as in emissions file

Table 3: Syntax of the file emissions

name	description	type
i	the code of the emitter	integer
v	the VOCs emission	floating-point(in 100 ktones)
n	the NOx emission	floating-point(in 100 ktones)

Table 4: Syntax of the headers in the file cost.nox

name	description	type
$	the dollar character	1 ASCII character
i	the code of the emitter	integer
emi	the NOx emission for the last point of this PWL-function[13]	float(in ktones)

CAUTION: emi is given in ktones not 100 ktones

A.5 File "cost.voc"

This file contains the description of cost functions for VOCs. The file is divided into sections; every section begins with a line with syntax as showed in Table 6.

Inside a section there is a number of lines describing all the points making the cost function. They are listed in Table 7.

A.6 File "definitions"

This file contains a set of definitions for the model. These features allow to define the number of receptors and emitters used by the model generator as well as to choose out from the whole set of parameters those, which will be used in the model (this mean to choose a number of columns and rows).

There are two obligatory lines in the file:

EMITTERS nn defines the number of emitters allowed in the model.

RECEPTORS nn defines the number of receptors allowed in the model.

REMARK: Both numbers can be overestimated.
There are also three optional sections which, if present, must follow the obligatory ones.

- VARIABLES
- CONSTRAINS

[14] should be the same as in emissions file

Table 5: Syntax of the data lines in the file cost.nox

name	description	type
cost	the marginal cost at this point	floating-point
emi	the emission coordinate of this point	floating-point(in ktones)

REMARK: the lines should be ordered by increasing the cost (and decreasing the emission, of course) - PWL_{1i} must be convex.

Table 6: Syntax of the headers in the file cost.voc

name	description	type
$	the dollar character	1 ASCII character
i	the code of the emitter	integer
emi	the VOCs emission for the last point of this PWL-function[14]	float(in ktones)

CAUTION: emi is given in ktones not 100 ktones

Table 7: Syntax of the data lines in the file cost.voc

name	description	type
cost	the marginal cost at this point	floating-point
emi	the emission coordinate of this point	floating-point(in ktones)

REMARK: the lines should be ordered by increasing the cost (and decreasing the emission, of course) - PWL_{2i} must be convex.

- USE

The first two can be used as follows: you can include (by putting the "+" character at the beginning of the line) or exclude (with the "-" character) any variable/constraint from the set of all available. Excluded variables will become parameters (constants) and excluded constraints will just be ignored. In one line you can specify one variable/constraint to be included, excluded or have a range of them. In addition in the variables section you can specify which emission (NO_x or VOCs) is taken into consideration. The syntax for every line from the VARIABLES section is:

```
[+|-] [n|v] number[-number]
```

and for every line from the CONSTRAINS section:

```
[+|-] number[-number]
```

where

- number is any integer number from 1 to the number defined in EMITTERS or RECEPTORS sections;
- in the range definition the second number must be higher than the first;
- any number of white space characters can be inserted between tokens.

The third statement is not used in this version of file generator. It is changed into C–preprocessor's #define statement. This way the developer can pass additional informations into the C++ file. This informations can, for example, change the way the generator treats some other parameters. An example of the definitions file follows:

```
EMITTERS 54
RECEPTORS 47
VARIABLES
- n 1-54
- v 1-54
+ n 6-9
+ v 6-9
```

```
CONSTRAINS
- 1 - 47
+ 1-10
```

This file defines that in the source files there can be up to 54 emitters and up to 47 receptors (numbered starting from 1). Only emissions from emitters with numbers from 6 to 9 are treated as decision variables (the rest make parameters) and only receptors from 1 to 10 define the valid constraints of ozone concentration.

B Mathematical Programming Problem

B.1 Naming Convention

Both the MPS and MDF files are generated by the Model Generator, and it implies the all names are also created by the generator program. As a rule all the names in the model are created using the same manner. They consist of the name part and the of zero or more indices separated by th underscore (_) character. Thus, the ozone concentration in 10-th receptor from equation (6) would be:

```
OZON_10
```

the transfer coefficient $a_{5,17}$ between 5-th emitter and 17-th receptor would be:

```
A_05_17
```

and the cost function from equation (11) would be:

```
COST
```

All the numbers are counted from 1.

B.2 Mathematical Programming version of the model formulation

The equations defined in Section 3 have to be changed slightly in order to be accepted by solvers. Although most of the equations were written to be easily interpreted, some changes were introduced. This section is intended to be a documentation of these changes. All the equations in the form as they appear in the MDF and MPS files are listed below. Some of the possible constraints are reduced if all their coefficients are equal to zero or the variables or constraints that depend on them are for some reasons, skipped.

- Equation (8) yields a number of rows (linear constraints). The number of them is determined by the number of the receptor for which:
  - some of the e_{ij} coefficients are non-zero or:
  - the en_j is non-zero

  These rows look as the following:

```
EN_01 = E_02_01 * N_02 + E_08_01 * N_08 + E_09_01 * N_09 +
        E_10_01 * N_10 + E_12_01 * N_12 + E_15_01 * N_15 +
        E_17_01 * N_17 + E_19_01 * N_19 + E_27_01 * N_27 +
        E_47_01 * N_47 + E_48_01 * N_48 + ENN_01
```

- Similarly, equation (7) yields a number of rows (linear constraints). The number of them is determined by the number of receptors for which:

– some of the d_{ij} coefficients are non-zero

These rows look as the following:

```
EV_01 = D_02_01 * V_02 + D_08_01 * V_08 + D_09_01 * V_09 +
        D_10_01 * V_10 + D_12_01 * V_12 + D_15_01 * V_15 +
        D_17_01 * V_17 + D_19_01 * V_19 + D_27_01 * V_27 +
        D_47_01 * V_47 + D_48_01 * V_48
```

- The rows, generated from equation (6) are divided into parts in order to simplify the notation and the process of the automatic differentiation. The present version of the model compiler also limits the length of the line to 256 characters. Each of the constraints **OZON_00** is split into several number of auxiliary rows called **O_00_00**. The number of the latter depend on the number on terms in the constraints. This constraint is generated if:

  – some of the a_{ij}, b_{ij} or c_{ij} for this constraints are non-zero or
  – the α_j parameter is non-zero
  – the en_j and ev_k with the same index were generated

```
O_01_01 = K_01 + A_02_01 * V_02 + B_02_01 * N_02 + C_02_01 * N_02 * N_02 +
                 A_08_01 * V_08 + B_08_01 * N_08 + C_08_01 * N_08 * N_08 +
                 A_09_01 * V_09 + B_09_01 * N_09 + C_09_01 * N_09 * N_09 +
                 A_10_01 * V_10 + B_10_01 * N_10 + C_10_01 * N_10 * N_10

O_01_02 = A_12_01 * V_12 + B_12_01 * N_12 + C_12_01 * N_12 * N_12 +
          A_15_01 * V_15 + B_15_01 * N_15 + C_15_01 * N_15 * N_15 +
          A_17_01 * V_17 + B_17_01 * N_17 + C_17_01 * N_17 * N_17 +
          A_19_01 * V_19 + B_19_01 * N_19 + C_19_01 * N_19 * N_19

O_01_03 = A_27_01 * V_27 + B_27_01 * N_27 + C_27_01 * N_27 * N_27 +
          A_47_01 * V_47 + B_47_01 * N_47 + C_47_01 * N_47 * N_47 +
          A_48_01 * V_48 + B_48_01 * N_48 + C_48_01 * N_48 * N_48 +
          ALPHA_01 * EN_01 * EN_01      + EN_01 * EV_01

OZON_01 = O_01_01 + O_01_02 + O_01_03
```

- The two piece-wise linear function from equations (9) and (10) were translated into two sets of linear constraints. In order to do it, the variables having the value of every of costs were introduced. Every segment of the cost functions is described by a pair (a_i, e_i) where

  – a_i denotes the marginal cost of the segment;
  – e_i denotes the emission, below which the constraints is significant;
  – v_i denotes the value of the cost function for the corresponding emission e_i;
  – c_i denotes the cost of the emission's reduction;
  – x denotes a variable (in this case it is one of the emissions).

  Each segment is described by the equation: $-a_i(x - e_i) + v_i + c_i \leq 0$

```
CA_02_02 = -1.055970 * (N_02 - 2.164580) + 0.036699 - CN_02
```

B.3 Columns (variables)

In the model two types of variables can be distinguished.
First are the decision variables. In the following descriptions "00" is used to indicate a number. All of the numbers consist of the same number of digits, (2 in this example) therefore leading zero(s) are kept.

N_00 for the NO_x emission in each of the emitters.

V_00 for the VOCs emission in each of the emitters.

Second, variables used to change the inequality constraints put on the ozone concentration into the equality ones:

OZ_00 these variables are equal to the ozone concentration in each of the receptors.

variables, which are the effective emissions of NO_x and VOCs:

EN_00 effective NO_x emission in each of receptors.

EV_00 effective $VOCs$ emission in each of receptors.

and variables of the piece-wise linear functions defining the cost:

CN_00 cost of given by N_00 NO_x emission in each of the emitters.

CV_00 cost of given by V_00 VOCs emission in each of the emitters.

B.4 Rows (constraints)

The following constraints are given in the model:

OZON_00 nonlinear constraints on the ozone concentration; they are a MP version of equation (6)

EVAUX_00 auxiliary constraints using to calculate the EV_00 variables; they are a MP version of equation (7)

ENAUX_00 auxiliary constraints using to calculate the EN_00 variables; they are a MP version of equation (8)

CA_00_00 constraints put on cost of NO_x emission in each of the emitters; there is a number of these constraints for every of them.

CB_00_00 constraints put on cost of VOCs emission in each of the emitters; there is a number of these constraints for every of them.

COST a constraint used to calculate the cost (this is the goal function).

C The UNIX environment

Although the final version of the system will be portable into different software platforms (MS-Windows), for development purposes the UNIX operating system was chosen. The variety of programmer's tools already existing in this operating system make UNIX the most convenient developer's environment. In this appendix we will cover several topics important in the process of software preparation.

- the directory structure

  All the system files are stored in one directory and its subdirectories. The name of the directory and place in the file system is up to the developer's preferences. However, it must be defined with the environment variable **$OZON_HOME**. There are several important subdirectories.

  bin the directory contains all the executables of the model generator

  models the directory contains different models definitions. For a detailed description of creating a new model see Appendix C.1. This directory can also contain subdirectories with solver-specific files.

  doc the directory contains different documentation file (including this document in LaTeX format)

- the environment variables

  The environment variable **$OZON_HOME** has to be defined and set to the root of the above mentioned directory tree. The environment variable PATH should point also to the **$OZON_HOME**/bin subdirectory.

C.1 Multi-models management

During the stage of developing the model it is usually necessary to create several versions of the model in order to check their performance and find any errors in data. Each of the models resides in a separate subdirectory of **$OZON_HOME**/models named according to the developer's preferences. The name of the directory is in the same time the name of the model used by each and every of the systems' programs. In order to create a new model description do as follows:

1. Create a new subdirectory[15]
2. Change the current directory to the new subdirectory.
3. Run the *prepare* program – this will create some files needed for further activities.
4. Create (copy, prepare, edit ...) files with developer's parameters. The list of these files and their contents is presented in Appendix A.
5. Type `make model`

This will create a number of files (and subdirectories) with model-specific data.

D Solver-specific issues

For some solvers additional files are needed in order to interface the solver with DIDASN++-supplied code. Another important thing is to generate the problem definition. It has different forms for different solvers.

D.1 MINOS

D.1.1 Interface with DIDASN++

In order to connect the MINOS solver with the C++ code generated by DIDASN++, some additional code (interface) is needed. This code has the purpose of translating the data output by MINOS into a form accepted by the DIDASN++-generated code and vice versa, translating its output into input accepted by MINOS. There are some reasons to do it:

[15] Consult the appropriate UNIX manual pages if needed.

- MINOS expects the user defined function to be written in Fortran. It causes the parameters to be passed in manner of the Fortran parameters passing (by pointers).
- For the same reasons the C++ code for the Jacobian matrix output has to be transformed into a code accepted by the Fortran code (rows vs. columns).
- Not all the equations defined in the MDF file are really needed by MINOS, some of them are to be just ignored (auxiliary equations). Also the Jacobian matrix is to be reduced.

The supplied code (that is placed in subdirectory **$OZON_HOME**/solver/minos) realizes the above mentioned actions.

D.1.2 Problem specification

Problem specification for the MINOS solver is placed in the MPS file. Therefore, the MPS file generated by the **gen** program has to be changed in order to specify the bounds of variables and constraints. The specification file also has to be updated with the name of the objective function and the type of optimization.

D.2 DIDASN++

D.2.1 Problem specification

Problem for the DIDASN++ solver is defined using the **pro_win2** program from the package. This program allows to define the problem interactively.

E Software Tools

E.1 The model compiler

The model compiler reads input file in the MDF format and creates a file with two C++ function. These functions are able to calculate function values (**calc_outcome**) or both function values and the Jacobian (**calc_both**). The generated file has name corresponding with the model's name and an extension ".C" In order to make these function callable from a C[16] functions small changes to the code have to be done. These are done by a **compile** shell script which also calls the model compiler. If the model name is *modelname* the output file would be named *modelname*.cc
The model compiler generates also a binary file (extension .bin) which contains the description of the model including names of variables and outcomes. This file is also needed by the problem generator in order to create the problem description file.

[16] or Fortran

F An example of MINOS run

```
       ==============================
       M I N O S    5.3    (Oct 1990)
       ==============================

OPTIONS file
------------
     BEGIN OZON
        MINIMIZE
        OBJECTIVE =             COST
        PROBLEM NUMBER          1313
        COLUMNS                  200
        ROWS                    1200
        ELEMENTS                6000
        NONLINEAR CONSTRAINTS     25
        NONLINEAR VARIABLES      117
        NONLINEAR OBJECTIVE VARIABLES      0
        JACOBIAN               DENSE
        SUMMARY FILE               9
        MPS FILE                  10
        NEW BASIS FILE            11
        UPPER BOUND           1000.0
        LOWER BOUND          -1000.0
        ITERATIONS             15000
        MAJOR ITERATIONS        1500
        MINOR ITERATIONS          30
        PENALTY PARAMETER        1.0
        SUPERBASICS LIMIT          6
        PRINT LEVEL (JFLXB)    00000
        VERIFY LEVEL               0
        CYCLE LIMIT                1
        CYCLE PRINT                2
        WORKSPACE (TOTAL)     600000
     END OZON

Reasonable WORKSPACE limits are         0 ...   61742
Actual     WORKSPACE limits are         0 ...  300000 ...  300000  words of  Z.

MPS file
--------
     1    NAME          OZON
     2    ROWS
  1159    COLUMNS
  3987    RHS
  5143    BOUNDS
  5279    ENDDATA



Names selected
--------------
OBJECTIVE      COST      (MIN)        1
RHS            RHS01               1105
RANGES                               0
BOUNDS         BOUND13              93

No. of rejected coefficients             1
No. of Jacobian entries specified        0
No. of INITIAL  BOUNDS  specified       42
No. of superbasics specified            42
Nonzeros allowed for in LU factors   177680
```

```
Matrix statistics
-----------------
              Total      Normal        Free       Fixed     Bounded
Rows           1156        1080           1          75           0
Columns         159           0           0           0         159

No. of matrix elements               5726     Density       3.115
Biggest                          6.1756E+01  (excluding fixed columns,
Smallest                         0.0000E+00   free rows, and RHS)

No. of objective coefficients          42
Biggest                          1.0000E+00  (excluding fixed columns)
Smallest                         1.0000E+00

Initial basis
-------------
No basis file supplied
Cheap test on FUNCON...
The Jacobian seems to be OK.
XXX  The largest discrepancy was    1.33E-06  in constraint    12

.
.
.

FUNCON called with NSTATE =   2
Clock  1 Mean time for entire program                 63.42 seconds
Clock  2 Mean time for solving problem                58.50 seconds
ENDRUN
```