Working Paper

Platform-Independent Storage of Data: An Application of HDF for the Simplified Ozone Model

IJsbrand G. Haagsma

WP-96-133
November 1996

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

Platform-Independent Storage of Data: An Application of HDF for the Simplified Ozone Model

IJsbrand G. Haagsma

WP-96-133
November 1996

International Institute for Applied Systems Analysis _ A-2361 Laxenburg _ Austria

Telephone: +43 2236 807 _ Telefax: +43 2236 71313 _ E-Mail: info@iiasa.ac.at

Foreword

This paper summarizes the results of the author’s research conducted during the 1996 Young Scientist Summer Program (YSSP) at IIASA's Methodology of Decision Analysis (MDA) Project. The aim of this research was twofold. First, methodological issues related to the design and implementation of decision support systems (DSSs) were studied. Second, a particular practical problem, namely platform-independent data storage, was examined in more detail.

The practical part of the research resulted in the development of a C++ class library that serves as an interface to the public domain library HDF (hierarchical data format). This applied research was motivated by the collaboration between the Transboundary Air Pollution (TAP) and MDA Projects. The TAP Project is developing and ozone model for assessing the results of various policy options aimed at reducting tropospheric ozone concentrations. This model uses data that are provided from different sources; therefore, a platform-independent data format is very useful for this application.

The developed C++ template class library supports the handling of sparse and dense, one- and two-dimensional arrays; therefore, it can easily be used by other applications as an interface to the HDF library. For this, the software was made available from the IIASA Web server.

Abstract

Many definitions are given for decision support systems (DSSs) and much research effort is put into the development of these systems for general use. Too much of this effort, however, is directed toward the development of computer tools and not enough emphasis is placed onto the architectural design of such systems. This paper tries to combine architectural design with a real life application. One key element in designing a DSS, that of platform independence, is emphasised using the simplified ozone model as an example. Adaptation of the hierchical data format (HDF) to store scientific data sets is one of the options available for reaching a certain level of platform independence.

Contents

Platform-Independent Storage of Data: An Application of HDF for the Simplified Ozone Model

IJsbrand G. Haagsma

1 Introduction

Software developed for decision support systems (DSSs) should be robust (reliable), reusable, and portable [14]. Reusable software should have a modular structure, meaning that parts of the software developed for one purpose can also be used in other cases. Therfore, these modules should be very reliable and well tested. Reusability is enhanced when the software is portable; that is, when it can be used on many different computer platforms running different different software versions and different kinds of operating systems.

Current trends toward developing DSSs for large heterogeneous computer networks underline the need for software portability. Data sharing is the backbone of a well-designed DSS. For very complex software systems, it is advisable to use commercial database management systems (DBMSs) that are supplied with a fourth-generation language such as SQL (structured query language) to ensure cross-platform accessibility of data. However, having a full featured DBMS with all available features in a model-based DSS might be too resource-consuming compared with the rest of the software. This need for computer resources on the part of large DBMSs creates the demand for an alternative way to achieve data-access portability.

One alternative wich is used in this paper, to store data, is to use a standard such as the hierarchical data format (HDF). When the HDF interface is used in the computer program, the data are stored in a standard file format that can be used in different computer systems without conversion. Using HDF was one of the ideas developed during the author's work on generic DSSs. The ideas regarding DSSs presented in Makowski [14] are clearly model based, where the reuse of software modules for the models is important. Development of a library of robust reusable utilities and tools is one of the key elements in his discussion. The ideas regarding model-based DSSs complement the ideas presented in the next section.

The discussion in section 2 focuses on a flexible design for a DSS. Flexible design allows easy modification of the DSS after it has been implemented in software. This can be accomplished when the design is modular such that modules can be added, removed, or replaced without redesigning the DSS. These modules are, for example, numerical models or databases. For a particular problem several of these modules are combined to compute results. Communication between these modules is important, and the adoption of communication standards enhances the ease of adding new modules to the DSS. Too many standards however, lead to a system that is not flexible with regard to incorporation of existing software, because this software was not designed using these standards as a reference. A trade-off needs to be made between flexibility and the level of standardisation. The use of HDF originates from previous work on generic DSSs [13]. Another aspect of flexibility is portability, which

enables the software to be easily ported to other platforms.

2 Decision support systems (DSSs)

It is almost impossible to imagine modelling the environment computer systems and networks. For complex problems, the choice is often made to develop a DSS that provides the user with all the necessary information. These tailor made often do not fall into a predefined and conceptualised framework.

It must be mentioned here that several authors (e.g., Kaden *et al.* [10]) emphasise that a DSS is a system to support decisions and, to make decisions. It follows then that a DSS should not supply the decision maker with an optimal solution, but should provide tools to ensure rational decision making [1].

2.1 Computers, organisations, and people

The requirements that a DSS should fulfil largely depend on the area of application and the expectations of the user, that is, the decision maker. During the development of a DSS and the DSS software, these requirements are major considerations for the choices that need to be made. This observation does not hold for the development of a generic DSS that can be used at multiple sites. We have to consider a generic DSS more as a set of guidelines for it's development or as an outline for it's design. Because the user and the area of application have a great influence on DSS development they should be part of a generic DSS. User requirements may change; thus, for a flexible DSS it is necessary to include the user in the system. Defining a DSS that only comprises the computer side of the modelling process is, therefore, very limiting. Looking only toward the electronic part of a DSS is quite common, however ([1], [12], [18]). The place of a DSS in the organisation in which it is used is not often discussed. Kaden *et al*. [10] briefly mention the organisation system as a component of a DSS, but do not discuss how the organisational structure should influence the architecture of a DSS. Johanns and Haagsma [9] describe the place that an information system can have in an organisation dealing with integrated water resources management and the impact the system can have. An information system as such can be part of a larger DSS, as will be described later. It is extremely important to be aware that the knowledge and the experience of the people using a DSS and the models that come with the system largely determine the usefulness of the DSS. Furthermore, the DSS's organisational embedding will determine whether the features of the DSS will be used to their full extent.

In the concept of a DSS described in this paper, the decision makers and analysts are a part of the system. Thus, during the development of the DSS the developers should be concerned about the kind people that will use the features of the DSS.

2.2 Generic DSSs

Many DSSs are developed as part of a project where the software for the DSS is tailored to the area of application or even to a specific project. Reusing (part of) the software developed

for other projects is often prohibitive as more often than not it would require major modifications in the software. The aforementioned process of developing DSSs is a result of the fact that most of these systems are developed by commercial organisations that are not able or willing to spend time and money to design a DSS that has a certain number of generic parts and features.

The generality of a DSS can be defined as the possibility of postponing certain decisions (software, data, platform, etc.). A consequence of the option to postpone making is that the underlying design of the DSS must be able to cope with possible changes in the future, and thus expensive. A trade-off must always be made between the costs of developing such a powerful system and the need to postpone making choices. For a generic DSS, however, a sufficient number of choices must be made to enable the integration of generic or specific components into the design.

A generic DSS consists of the design combined with a set of guidelines to be followed and the standard to be adopted. Depending on the amount of detail put into the generic version , the area of application will be broader or more limited. An actual DSS, is fully developed with all the software present, using the guidelines, standards, and design of the generic DSS; hence, the area of application of an actual DSS is always very limited. In contrast, a generic DSS is a starting point for the study of different problems, always with a similar path to the solution in mind.

2.3 Components

Several generic components of a DSS can be identified. Figure 1 shows four of these components:

• Measurement system;

• Information system;

• Model system;

• Analysis system.

Each of these components describes a set of smaller components in a recursive way. In an actual DSSs not all of these components need to be present; a selection can be made, excluding some of the components. If these components were to be excluded from the generic design, the result could be a lack of standards and guidelines needed for including such components in the actual DSS. Communication standards are an important part of the data interface. These standards include formats on how numerical models should write their data to a disk and how they should communicate with each other, when used simultaneously for solving more complex problems.

Two important components that are absent in Figure 1 are the human factor and the environmental setting. Of course, the environment should not be a part of the DSS, but it is necessary to keep in mind that changes in the environment often trigger questions that require certain measures to be taken. The required measures will be the result of a decision-making process, supported by a DSS, and will have a certain impact on this environment. It is the

author's conviction that, especially in the analysis system (or the interpretation layer), the human factor is so predominant that it should be included in the design. When that choice is made, a slightly different structure of the components in a DSS can be drawn. This choice is supported by Kainuma *et al.* [11], who mention that any precise models of reality will never incorporate all human concerns. Experts, decision makers, and analysts should therefore be part of what they call an integrated support system.

Figure 1. Components in a generic DSS

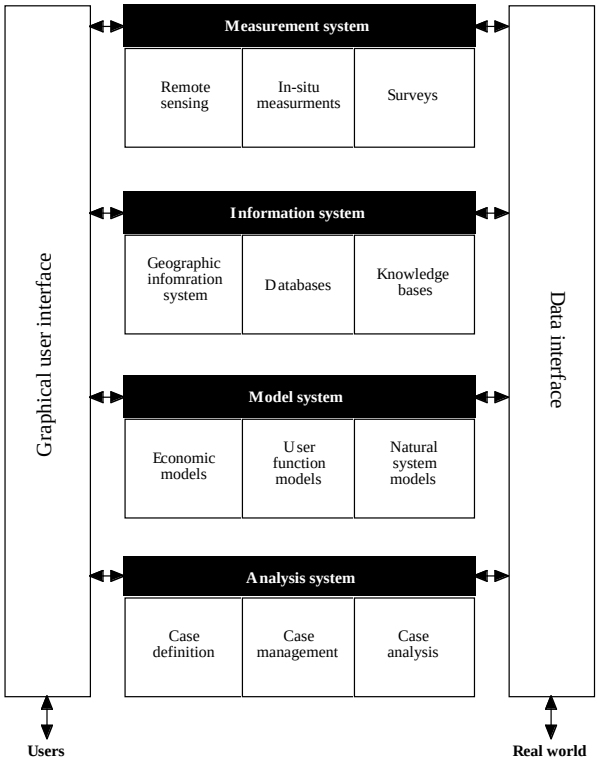


Figure 2. DSS architecture

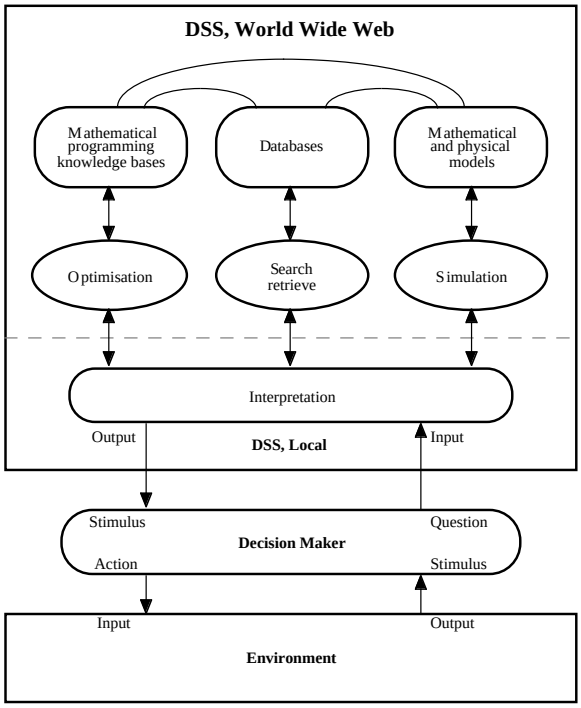


Figure 2 shows the decision maker reacting to and acting on the environment. He or she uses the DSS as a support system on which to base actions. For simple decisions the decision maker will not consult the DSS; but for more complex problems he or she will pose the question towards the DSS. On a policy-making level these questions generally cannot be anticipated by the software developer. The question of the decision maker therefore needs to be interpreted manually by analysts. Here we introduce the human factor into the system. These analysts convert the decision maker's question into a model or question that is suitable for a computer. They will also interpret the results of the computer calculation, converting it back into a format understandable to the decision maker. Methods typically used to solve environmental problems are:

• Optimisation;

• Simulation;

• Data search and analysis.

The software that implements these methods does not necessarily have to reside on a local computer. The software can be distributed on a wide area network and can even have some

kind of World Wide Web interface. Implementations using Java allow the use of remote models.

Solving mosts probems requires the use of a combination of methods. Consequently, data will be exchanged between the different software used, and standardisation in communication and in data storage is therefore required. A traditional DBMS is normally capable of handling this kind of communication and data storage. However, incorporating a large commercial database into the software solution is sometimes prohibitive. Simpler solutions are available and are, for the most part, preferred. One of these solutions is HDF, which is used for a case study described in this report. For more complex studies, where simplicity of the whole system is no longer a requirement, the DBMS can be used to store the bulk of the data. Text-based data files will normally be used by the individual application in a proprietary format. For the data that are shared among the applications, using a format like HDF is advisable.

The main difference between Figures 1 and 2 is the absence of an interpretation layer in Figure 1. The analysis system in Figure 1 consists of tools that enable the analysts to convert the decision maker's question into a format suitable for use in a computer program and for analysing the results. Several tools for interpretation and comparison are available, but the results of the interpretation and the comparison of cases still depend on the analyst. The author is convinced that the human factor in the analysis system is so important that it is better to include it in an interpretation layer. This interpretation layer, of course, requires considerable detail for practical purposes. The tools used by the analysts are normally available on a local computer, which supports the division of the DSS into two layers, one represented by a wide area network and the other by a local area network.

2.4 Model bases

An actual DSS that is developed for a particular problem will consist of several tools and numerical models. Common practice is to decompose large problems into smaller parts. For the study of each of these parts, it is normally not necessary to use more than one numerical model. Because the actual DSS is tailored to a problem, the developers will have already made choices for the numerical models based on the characteristics and parameters of the area that is studied.

When a generic DSS is designed, the nature of the problem or the domain characteristics are not known beforehand. A model base can be very useful in these cases. The model base should consist not only of models that are capable of studying the different processes for which the DSS is designed, but should also contain models that study the same processes using different analytical or numerical techniques. Models from different engineering disciplines will contribute to the model base. Multidisciplinarity in environmental engineering is the driving force behind this evolution.

According to De Leo *et al.* [4], the model selection process is important. They developed a is able to assist the user in selecting the best model from the model base. This selection process is interactive, and the user is constantly informed about the selection decisions that are made. The tool they use is flexible enough to select multiple models when possible. If no suitable model is available from the model base, the tool can suggest modifications to the problem

specifications so that the modified model can be solved by one of the models in the model base.

Ryan and Sieh [17] explicitly state that the data management system should be at the centre of the DSS architecture. They suggest that a data-centred approach consequently means one core database for multiple applications; however, as discussed earlier, this database may be distributed as long as it is commonly accessible. The linkage between the applications and the database is made through data management interfaces (DMIs). For each module or application a new DMI must be developed, which is a small investment compared with recoding the entire interface.

The central data management system can also be replaced by a data server [6]. This approach is still data centred, but the application now requests the necessary data from the data server, which returns the data in the requested format. The advantage is that multiple databases can be used more easily. Furthermore, not all applications need to access the whole, often extremely large, database. Only the necessary data need to be retrieved through the data server.

When a DMI or data server is developed, certain communication standards should be adopted. The use of a standard data format by the models in the model base enhances communication between them, as this communication mostly involves the exchange of scientific data. HDF or an equivalent format should therefore be part of a DMI or data server.

3 Simplified ozone model

The simplified ozone model describes the relationship between tropospheric ozone and the ozone precursors nitrogen oxides (NO_x) and volatile organic compounds (VOCs). The number of emission sites of NO_x and VOCs is defined and a larger number of reception areas of ozone is chosen. The ozone concentration at each reception site is calculated from emitted NO_x and VOCs. Part of the model involves the representation of the physical and chemical aspects used to simulate the ozone concentration. For public health reasons the goal is to keep the ozone concentration within certain limits. An optimisation model can be used to determine the NO_x and VOC emission sites (emitters) at which it is most cost effective to take measures to reduce ozone concentrations at the selected reception sites (receptors). An optimal solution keeps the ozone concentration within the preset limits at the receptors for all available scenarios at minimal cost.

3.1 Source-receptor relationships

Ozone is formed by chemical reactions between NO_x and VOC that are driven by solar radiation. In polluted air and under suitable weather conditions, ozone concentrations can build up in a matter of days due to an imbalance in the chemical reactions caused by the available pollutants. Furthermore, the pollutants and the precursors can travel considerable distances and across national borders. An essential requirement of an integrated assessment model for ozone is a simplified but reliable description of the ozone formation process in order to represent the source-receptor relationships involved. It is possible to envisage several

ways of condensing the results of more complex models of ozone formation to achieve this. One approach is to use statistical techniques to summarise the results obtained from a complex mathematical model for a large number of emission reduction scenarios. The simplified ozone model is built using a theory with a physical and chemical foundation. This has resulted in a mathmatical description of the physical and chemical processes that is widely accepted. The decision to use extrapolation methods on the measured data can only supported by pragmatic reasons, for example, the need to oversimplify the physics in a model.

It is important to realise that there are no direct emissions of ozone in the atmosphere; all the ozone there has been formed by chemical reactions in the air. The concentration of ozone (O_3) is balanced by the concentration of nitrogen dioxide (NO_2) and nitric oxide (NO), and the two corresponding reaction rates J_1 and k_3, the reaction rates for the NO_2 photolysis and the reaction for NO_2 regeneration, respectively:

$(O_3) = J_1 (NO_2) / k_3 (NO)$.

The photolysis of NO_2 is described by two reactions where $h\upsilon$ is the product of Planck's constant and the frequency of solar radiation, $O(^3P)$ is the ground state of an oxygen atom and M can be any inert molecule such as nitrogen or oxygen.

$NO_2 + h\upsilon \rightarrow NO + O(^3P)$,

$O(^3P) + O_2 + M \rightarrow O_3 + M$.

The regeneration of NO_2 is described by the following reaction.

$NO + O_3 \rightarrow NO_2 + O_2$.

Any other reaction that converts NO to NO_2 will facilitate the net production of ozone during daylight. Such processes are possible in polluted atmospheres containing VOCs, primarily originating from pollutant emissions, but also from natural sources. The chemistry of these reactions with VOCs is very complex, as VOCs can be divided into numerous classes each with different reaction types. Heyes and Schöpp [7] and Heyes *et al.* [8] give the most important reactions. It must be stated here that it is not merely the presence, but more the ratio between NO_x and VOC in the atmosphere, that determines the concentration of ozone.

Dry deposition is a major sink for ozone. This process is regulated by stomatal uptake in vegetation canopies and depends on light, temperature, and humidity.

Also, large- and local-scale mixing processes affect the ozone concentrations measured at the ground level to a considerable extent.

The complete representation of the complex interaction between precursor emissions, photo-chemistry, transport, and deposition in a mathematical model is currently only possible with considerable simplification. In the simplified ozone model, the description of the processes is divided into three parts: the atmospheric boundary layer, the ground level, and the free troposphere. Heyes *et al*. [8] give detailed information regarding the assumptions made during the simplification.

3.2 Integration of information

To describe the complex relationship between NO_x and VOC emissions and the exposure of ozone combined with the search for cost effective-solutions, it is necessary to integrate information from the following areas:

• Current and future emissions of NO_x and VOCs, both man made and natural;

• Abatement technologies available for NO_x and VOCs, and their costs;

• A concise description of the source-receptor relationships, taking into account meteorological influences on ozone formation;

• Studies of the effects of ozone on agricultural crops, forests, and human health, leading to the establishment of critical levels for ozone.

It is crucial to realise that all this information should be accessible and organised in such a way that scenario analysis (exploring the costs and environmental impacts of alternative emission reduction scenarios) and optimisation (the systematic search for cost-effective solutions) are possible [8]. The implementation of the resulting integrated assessment model will be heterogeneous in the sense that many different kinds of computer models and software utilities are required to process the information that is available. A modular setup for models and software around the data used, as discussed section 2, will be necessary for maintenance and flexibility reasons. It is likely that both UNIX-based workstations and personal computers will be used to process the data. It is important for the integrated assessment model to be able to store and retrieve data independent of the sort of computer or operating system that is used, so that conversion of data is unnecessary. Furthermore, the organisation of the data is a key aspect of the integrated assessment model. Rather than inventing a completely new data format structure for storage in text files, it would be much simpler to adopt a certain standard and to agree on the naming of the variables and the data structures that are used in the source code. Of course, commercial database management systems could be used to organise data. However, these databases would decrease the simplicity of the software considerably.

It is important that the calculations be efficient enough to permit numerous scenario runs for analysing the costs and benefits of a wide range of control strategies. Robustness analyses are required to make sure that the model can cope with missing data and data imperfections.

3.3 Data

This paper is not the place to fully describe all the necessary data for the simplified ozone model. However, to get a feel for the amount and structure of the data needed, some of them will be described in this section. For this purpose, one of the major equations is used as an example. The mean concentration of ozone at each receptor region j, $(O_3)_j$, can be calculated using

$$\overline{(O_3)}_j = k_j + \sum_{i=1}^{M} (a_{ij} v_i + b_{ij} n_i + c_{ij} n_i^2) + a_j \overline{en}_j^2 + \overline{en}_j \sum_{i=1}^{M} d_{ij} v_i .$$

The coefficients i and j refer to a particular emitter and receptor, respectively, and M is the number of emitters. The matrices a_{ij} and b_{ij} contain coefficients that describe the linear contribution from VOC emissions, v_i, and NO_x emissions, n_i; furthermore, c_{ij} is a correction term to allow for nonlinearities due to possible high NO_x emissions from some countries. All these matrices are input parameters for the model and are known up front; they are typically sparse and are stored as sparse matrices. The sparse matrix d_{ij} describes the interaction between NO_x and VOCs along these trajectories, whereas the dense vector α_i allows for some additional nonlinearities at receptor j. All coefficients are preset, which means that no prior knowledge regarding the chemistry involved is used to set these coefficients. In subsequent versions this feature might be added to the model; thus the model and the handling of data should allow for these changes.

Because there are many zero coefficients in the data, it was decided to distinguish between sparse and dense data sets. Dense data sets are stored as vectors or arrays and sparse data sets are stored in triples, one vector containing the data, one vector containing coefficients and one vector containing the column length in the case of sparse matrices. The matrix and vectors below show the two different representations of a sparse matrix a_{ij}, of which the transposed form is written out for notational reasons. The vector *a_cl* contains the column length of a_{ij} for each of the columns, *a_ri* contains the row-index, and *a_vl* contains the corresponding values. It is relatively simple to make a function that handles the conversion between the two representations; however, the storage of a sparse matrix as a vector lead to considerable savings on the amount of data that must be written. The main reason for using sparse matrices in the simplified ozone model is that excluding zero-coefficients from the optimisation model saves considerable computing time. This way of storing the data is only used when writing to a disk; in the source code itself different structures are used that deal with sparse matrices and vectors and dense matrices and vectors. The two representative forms of a sparse matrix are given as an example. For convenience the transposed form of the full matrix notation is given.

$$a_{ij}^T = \begin{pmatrix} 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$a\_cl = (3 \quad 6 \quad 3 \quad 4 \quad 4 \quad 1)$$

$$a\_ri = (3 \quad 9 \quad 15 \quad 0 \quad 1 \quad 8 \quad 13 \quad 17 \quad 18 \quad 4 \quad 7 \quad 16 \quad 1 \quad 8 \quad 10 \quad 14 \quad 2 \quad 6 \quad 12 \quad 19 \quad 11)$$

$$a\_vl = (4 \quad 1 \quad 4 \quad 8 \quad 1 \quad 2 \quad 2 \quad 2 \quad 1 \quad 2 \quad 9 \quad 7 \quad 3 \quad 2 \quad 2 \quad 5 \quad 4 \quad 4 \quad 9 \quad 9 \quad 3)$$

A full overview of the storage capabilities developed for the simplified ozone model is given later in the text.

4 Standards

One of the difficulties of integrating a variety of databases and models into a DSS is the naming of variables and objects. Moreover, the dimensions of the variables may be different (e.g. feet or meters). Naming conflicts are more obvious when combining databases than when combining models. The merging of two programs is not a trivial task when each of the source codes makes extensive use of variable names used in the other source, but with different meaning. Although Bhargava *et al*. [2] suggest a solution to the problem of unique name violations, it is worth the effort to deal with this problem prior to the integration. The easiest solution is to agree on a dictionary describing the essential jargon. When source codes of computer programs are not merged, but are linked as independent modules, there is no need to be afraid of a unique name violation within the computer programs. In these cases the modules do not share a common library of objects or variables.

4.1 Metadata

Data in environmental management can be inconsistent or even contradictory. Inconsistencies in air pollution data occur regularly when a measurement threshold is used. In field measurements for air pollution, this threshold can differ depending on the instrument that is used, but it can also depend on different organisational policies. If a person wants to retrieve data with a minimum concentration that is below the highest threshold in the database, then he or she will never know whether some data have been missed. In this case, with a system of heterogeneous distributed databases, metadata could have indicated which data could not be interpreted.

Metadata is necessary to store data in a kind of self-explanatory [16], thus providing information about the kind of data that are stored in the database. Other relevant knowledge that can be stored as metadata may be the quality of data, data origin, the data updating sequence, data accuracy, etc. Furthermore, the access rights and physical storage paths of the data can be included.

4.2 Platform independence

One of the principal concerns when developing model-based DSSs is how flexible the software needs to be. The simplified ozone model complies with the ANSI standards for C++, which means that any compiler should be able to compile the code. Thus, in principle the model is portable, such that it can be used on more than one platform. Because standards still seem to vary, portability cannot be guaranteed, however it should be relatively easy to support a wide range of platforms.

Another problem regarding portability is due, not to different standards used for computer languages, but to different standards that arise used for the physical storage of data. When storing values as text the problems are due to the different conventions used as end-of-line markers. Each of the three prinicipal kinds of operating systems; MS-DOS, UNIX, and Macintosh does this in a different way. Converters are of course available, but they reduce the user-friendliness of the software system and make it less efficient. Furthermore, storing values

as text requires a specification, describing the way the data is stored in the file. This specification should be used by all the software in the system and is typically tailored to the system and likely to need regular updates. Inventing such a specification for storage of data consequently means more maintenance effort in the future for the software system.

Rather than using text-based files for storage of data, developers like to use binary files, because accessing these type of files is faster than accessing text files. The problem of using binary files is not the way end of lines are coded, but the bit-order in which values are stored. There are two ways operating systems like to store values, either in little-endian (least-significant bit first) or in big-endian (most-significant bit first) order. The orders are not compatible and a “holy war” between the two camps still exists [3]. There are solutions that can be used to get around this ordering problem. One of these solutions is implemented in HDF; another is the use of the external data representation (XDR) language. The relation between both of them will be discussed in the next section.

5 Hierarchical Data Format (HDF)

The National Center for Supercomputing Applications (NCSA) has been working for some time on the development of HDF, a multi-object file format for sharing data in a distributed environment. The principal requirements for storing scientific data addressed by HDF are

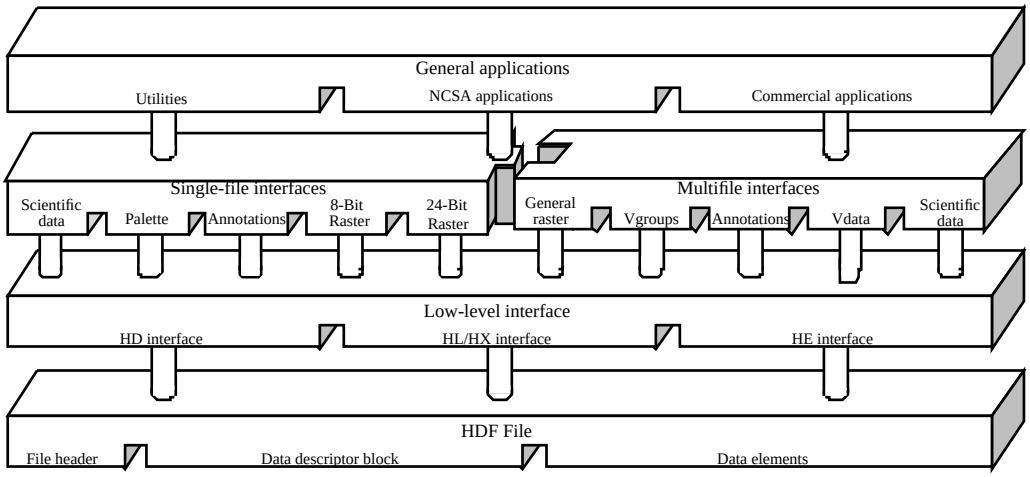
• Support for the types of data and metadata commonly used by scientists;

• Efficient storage of and access to large data sets;

• Platform independence;

• Extendibility for future enhancements and compatibility with other standard formats.

Figure 3. The HDF interfaces



The principal advantage of HDF over a format that is developed for a special case is that HDF consists not only of a specification for a file format, but is supplemented with application programming interfaces (API) and other supporting software (Figure 3). The API consists of a set of routines for each data type that makes the process of storing and accessing data in an HDF file relatively simple and straightforward. An API consists of functions or procedures

that need to be included in the source code; however, the use of these functions and procedures is simpler than the use of the standard I/O routines provided by FORTRAN-77 and C. The source code needs only to be linked to the necessary HDF libraries to make the API operational.

For advanced software development a low-level interface is also provided, although, for normal development of applications, one does not need this low-level interface. The API, however, uses the interface for the actual storage and retrieval of the data, without the user explicitly knowing this.

The following are among the important features of HDF:

• Information about data in a file can be obtained from the file rather than from another source;

• HDF standardises the formats and descriptions of many commonly used data sets;

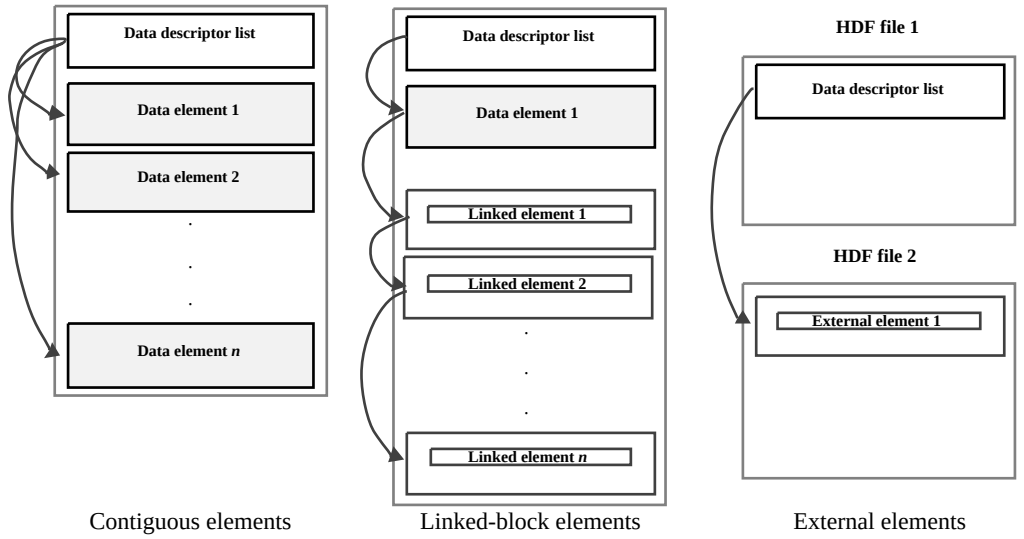
• HDF can be extended to accommodate virtually any kind of data by defining new tags or new combinations of tags.

5.1 File format

HDF files rely on the fact that everything is written in big-endian order, which makes the HDF files the same on every platform regardless the order of storage (little- or big-endian) used by the computer. A 32-bit hexadecimal value identifies the file as an HDF file, and the libraries that handle the low-level interface, which differ among platforms, actually take care of the bit swapping when necessary.

There are three options for storing data objects; the default option is storage of contiguous elements where each data element is described in a data descriptor list. For completeness all manners, of storing objects in HDF files are shown in Figure 4.

Figure 4. Storage of data elements in HDF files



The network common data format (NetCDF) developed at the Unidata Program Center uses XDR to store data and can be used as an alternative to HDF. Similar to HDF, XDR stores the data in a big-endian order and a number of libraries handle the low-level storage of data. NetCDF and HDF can be used interchangeably; however, there are some conceptual differences. To effectively use both data models together, it is important that for differences to be well understood. Details regarding these differences can be found in Folk *et al*. [5].

A feature that is not explored in this paper is the possibility of creating a hierarchical structure in the data.

5.2 Application programming interface (API)

Currently, only one API, the SD API, is used in the simplified ozone model. The SD API handles the storage, management, and retrieval of multidimensional arrys of integer or floating-point data, along with their dimensions and attributes, in more than one file. Another API, the VS API, can also be of interest; it deals with multivariate data stored as records in a table, which are also referred to as Vdata.

Any multidimensional array can be stored as a scientific data set (SDS) as long as it is associated with a dimension record and a data type. Additional information like dimensions, and attributes can be stored along with the SDS.

Before an SDS is actually written to a file, the elements are converted from the native format into a standard HDF format. HDF uses the IEEE 32- and 64-bit floating point formats and big-endian ordering. It is possible for a user to store the data elements in a native format using one of the options for storing data elements in an alternative way.

Storing data as Vdata is especially useful when storing a collection of records in fixed-length fields. Vdata has the advantage over SDS of being able to store tables where different columns contain different data types. This is not possible using the SD API. However, in the VS API there is no provision for changing the size of the tables once a Vdata element is created and Vdata element is not capable of storing data sets in more than two dimensions.

5.3 Version and platforms

Version 4.0r2 of the HDF library for the Solaris 2.4 operating system on SUN workstations that is used with the simplified ozone model. In the near future, versions for several popular compilers running under MS Windows 3.11 and Windows 95 should become available. Previous versions of HDF supported several compilers running under MS Windows 3.11. The latest information is available at the following Web location: http://hdf.ncsa.uiuc.edu/.

6 Tailoring HDF to the simplified ozone model

Adopting a standard for an existing model will require some adaptations. The simplified ozone model has so far used several text-based files for input and output. The classes used in the C++ code were not developed with the aim of using them in conjunction with HDF.

Therefore, a clear strategy had to be developed to implement HDF with the simplified ozone model. The following requirements were identified during that process:

• Communication should take place with only one HDF file;

• A utility for conversions between the text-based files and the HDF file should be developed;

• All relevant classes used in the source code should be supported;

• It should be possible to make data temporarily unavailable for the model;

• There should be the option to overwrite existing data in the HDF-file.

6.1 Communication

To make the bookkeeping as simple as possible, it was decided to store all data in a single HDF file. Because the API has direct access to the data in the HDF file, there should be no problem with data access time, even when the file grows quite large. For large text-based files, the access time might become a problem. Moreover, it is much easier to support different data types within one file using HDF than it is using text-based files, because the necessary annotation is largely hidden from the user in HDF-files.

Overall, the advantage to using a single HDF file is that the user now only needs to know on what project he or she is working and which data he or she needs to use. No knowledge about which data are stored in what file is required, making the model relatively user friendly.

6.2 Conversion

The simplified ozone model uses several input files. Each of these input files consists of one or more data sets of the same type. These data sets can be classified according to type in the following way:

• Dense vectors

  These are represented by normal arrays in C++ and are stored as a one-dimensional data set in HDF. Some of the elements may be equal to zero, but these zero-elements are stored in the array. The indexing of the elements is continuous, thus there should be no gaps in the data. To ensure this, a function was created that converts the indices used in the text file into a private index that does not contain gaps. The private index is used for all data to ensure data integrity.

• Sparse vectors

  When many of the elements are zero it is sometimes useful to save storage space and use a sparse vector description. A sparse vector is stored as two arrays in an HDF file; one of the arrays contains the indices of the sparse vector, the other contains the values. This might suggest that the use of sparse vectors is only advantageous when a vector contains more than 50% zeros. However, because the storage of indices normally takes less space than, for example, doubles, this balance point could be reached with even a lower percentage of

zeros. Moreover, this disregards the fact that the calculations in the model take less CPU-time when a sparse vector representation is used.

• Dense matrices

Dense matrices in an HDF file are simply stored in one array. In addition, one extra variable containing the column length should be stored. The SD API is capable of storing dense matrices directly; however, representing a dense matrix as one array and the column length was done to be consistent with the description of sparse matrices.

• Sparse matrices

Sparse matrices in the simplified ozone model are represented as a dense vector of sparse vectors. Again, a single array is used to store the values of these sparse vectors. In addition, a vector containing the row indices and a vector containing the column length are necessary to obtain a complete description of a sparse matrix.

For the conversion of the text files a function has been written that decides, based on the first line in the file, what type of data the file contains. For each type, a function has been written that stores the data in the correct classes. Using the functions in the HDF class, which will be described later, the data sets can be added to an HDF file. No provision has been made to convert an HDF-file back to its original text-form.

6.3 Classes

An HDF class in C++ was developed that deals with the processing of data in an HDF file, which is an object in the HDF class. The following classes are supported by the HDF class:

- `mVect<unsigned short, unsigned short>`
- `mVect<unsigned short, double>`
- `mVect<unsigned long, unsigned short>`
- `mVect<unsigned long, double>`

  These classes are dense vectors, where the first parameter of each template indicates whether the vector can contain a small or large amount of elements. The latter two classes are mainly supported because dense matrices are stored in an HDF file using the `mVect` class. Because the number of elements in these matrices grows quadratically, it might be necessary to have long vectors. The second parameter indicates the data type of the vector elements. The unsisned short type is supported, because it is necessary for sparse matrices to store column length and row indices, which are of this type.

- `mSpVect<unsigned short, double>`

  This class is used to describe sparse vectors.

- `dMatrix<unsigned short, unsigned short>`
- `dMatrix<unsigned short, double>`

  These classes are used to work with dense matrices, where the class that contains unsigned short elements is used to work with strings of character data. In effective, this class is used to describe a vector of strings.

- `mMatrix<unsigned short, double>`
  This is the sparse matrix class. Sparse matrices can be thought of as dense vectors in which each of the elements is effectively a sparse vector.

All these classes are described in more detail by Makowski [15].

6.4 Overwriting data

There are several options for storing data. The first choice is whether to store data in an existing file or not; there is also the option of forcing the creation of a new HDF file. The creation of a new file will overwrite any existing old file and will thus cause a loss of data.

Several data sets are stored within an HDF file. It is possible to also overwrite these individual data sets. For that reason, the `process` function in the HDF class contains a function that can overwrite existing data sets.

In some cases it is convenient to check whether a data set with a certain name already exists in the HDF file; this check can be made using the `present` function in the HDF class.

6.5 Hiding data

Sometimes the simplified ozone model must run without certain data. Because HDF unfortunately does not have a provision for deleting data sets, the only way to accomplish these runs is to include a flag with the data that checks whether the data are available to the model or not.

HDF can attach attributes to data sets. For hiding the data from the model, an attribute “available” is introduced. This Boolean attribute checks whether the simplified ozone model can use the data set or not.

In the HDF class, the function `mk_avail` changes, hides, or restores the data from the model and the function `available` checks whether the data are hidden or not. Initially the data set will be available when it is stored.

7 Conclusions

The use of HDF or any other similar standard has proved to be quite useful for storaging data sets that are used in combination with computer programs written in C++. HDF was applied in the simplified ozone model. Adopting this standard made communication with the data sets easier, and the neeed to make choices about the way to store certain object classes helped identify possible inconsistencies in the data.

For the simplified ozone model, a tailored class of functions has been written in C++ to store the data in an HDF file regard to the data types. HDF is developed for FORTRAN and C. The transition to C++ is, in that way, very easy; however, if one likes to use the full features of C++, it becomes very cumbersome.

A great step forward in the development of HDF would be an interface to C++ with all the available features, where different object classes and data types would be automatically detected. Currently, the correct data type must be provided for each store and retrieve action. An incorrect data type in the function arguments will lead to erroneous reading and writing of data.

Despite the latter disadvantage, HDF is a promising development in communication between models and data when the use of large DBMSs is not necessary for the application. The software developer should consider using HDF or one of its alternatives particularly, when different types of data sets and/or different platforms are used. Furthermore, software developers should keep in mind that it is likely that platform independence will become more and more important, due to the evolution to more heterogeneous software systems.

Currently, the data provided for the simplified ozone model are still in the conventional text format. The increasing amount of tools and utilities that support HDF can be used to provide data in HDF. The developed conversion software can then be dropped, which is a considerable advantage with respect to maintenance. At this moment changes in the file format always causes some confusion, because the person who develops of the conversion software is not the person who creates the data files. When the data files are directly created in HDF, one a cause for confusion can be eliminated.

This latter problem should be looked at more carefully and the possibilities of using some visualisation tools that can handle HDF files for analysis of the data should be explored. Information regarding these utilities and tools can be found at the Web location given at the end of section 5.

Acknowledgement

During my stay at IIASA, I had the opportunity to work on a real world example, which required me to become familiar with C. I am very grateful that Marek Makowski took the time and effort to patiently teach me the basic principles of C++. I would also like to thank Pawel Bialon, who helped me to debug my written codes. Without their help, I do not think that I would have been able to introduce HDF at IIASA.

References

1. Arnold, Uwe & Gerald T. Orlob 1989. Decision support for estuarine water quality management. *Journal of Water Resources Planning and Management* **115(6)**, 775-792.
2. Bhargava, Hemant K., Steven O. Kimbrough & Ramayya Krishnan 1991. Unique names violations, a problem for model integration or you say tomato, I say tomahto. *ORSA Journal on Computing* **3(2)**, 107-120.
3. Cohen, Danny 1981. On holy wars and a plea for peace. *Computer* **14(10)**, 48-54.
4. De Leo, G., L. Del Furia, C. Gandolfi & G. Guariso 1990. Models and data integration for groundwater decision support. In Gambolati, G., A. Rinaldo, C.A. Brebbia, W.G. Gray & G.F. Pinder (eds), *Computational Methods in Subsurface Hydrology. Proceedings of the Eighth International Conference on Computational Methods in Water Resources, Venice, 11-15 June 1990, Vol. 1*, 531-536. Computational Mechanics Publications, Southampton, UK.

5. Folk, Michael, Laura Kalman & William Whitehouse 1996. *HDF User's Guide.* NCSA, Champaign, IL, USA.
6. Haagsma, IJsbrand G. 1995. The integration of computer models and data bases into a decision support system for water resources management. In Simonovic, Slobodan P., Zbigniew Kundzewicz, Dan Rosbjerg & Kuniyoshi Takeuchi (eds), *Modelling and Management of Sustainable Basin-Scale Water Resource Management. Proceedings of the XXI Assembly of the IUGG, 1-14 July 1995, Boulder. IAHS Publication No 231*, 253-261. IAHS Press, Wallingford, UK.
7. Heyes, Chris & Wolfgang Schöpp 1995. Towards a Simplified Model to Describe Ozone Formation in Europe. WP-95-34. International Institute for Applied Systems Analysis, Laxenburg, Austria.
8. Heyes, Chris, Wolfgang Schöpp, Markus Amann & S. Unger 1996. A Simplified Model to Predict Long-term Ozone Concentrations in Europe. WP-96-12. International Institute for Applied Systems Analysis, Laxenburg, Austria.
9. Johanns, Remco D. & IJsbrand G. Haagsma 1995. Integrated water resources management in the information age. In Niemczynowicz, Janusz & Kerstin Krahner (eds), *International Symposium "Integrated Water Management in Urban Areas", 26-30 September 1995, Lund*, 169-179. University of Lund, Lund, Sweden.
10. Kaden, S., A. Becker & A. Gnauck 1989. Decision-support systems for water management. In Loucks, Daniel P. (ed.), *Systems Analysis for Water Resources Management: Closing the Gap between Theory and Practice. Proceedings of the Third Scientific Assembly of the IAHS, Baltimore, May 1989. IAHS Publication No. 180*, 11-21. IAHS Press, Wallingford, UK.
11. Kainuma, M., Y. Nakamori & T. Morita 1989. Integrated support system for large-scale modeling and simulation. In *SICE '89: Proceedings of the 28th SICE Annual Conference, Tokyo, 25-27 July 1989, Vol. 2*, 1209-1212. IEEE, Piscataway, NJ, USA.
12. Loucks, Daniel P. 1995. Developing and implementing decision support systems: A critique and a challenge. *Water Resources Bulletin* **31(4)**, 571-582.
13. LWI, 1995. *Generiek Decision Support Systeem. Projectgroep Estuaria en Kusten, Werkpakket 2*. Land Water Milieu Informatietechnologie, Gouda, The Netherlands.
14. Makowski, Marek 1994. Design and Implementation of Model-based Decision Support Systems. WP-94-86. International Institute for Applied Systems Analysis, Laxenburg, Austria.
15. Makowski, Marek 1997. LP-DIT++, C++ Class Library for Data Interchange Tool for Linear Programming Problems, Version 2.06. WP-97-XXX. International Institute for Applied Systems Analysis, Laxenburg, Austria.
16. Michalski, R. & F.J. Radermacher 1993. Challenges for information systems: Representation, modeling, and metaknowledge. In Holsapple, Clyde W. & Andrew B. Whinston (eds), *Recent Developments in Decision Support Systems. NATO ASI Series F: Computer and System Sciences, Vol. 101*, 3-22. Springer-Verlag, Berlin, Germany.
17. Ryan, Thomas P. & David Sieh 1993. Integrating hydrologic models, geographic information systems and multiple databases: A data centered approach. In *Proceedings of the Federal Interagency Workshop on Hydrologic Modeling Demands for the 90's. US Geological Survey Water-Resources Investigations Report 93-4018*, 26-34. US Geological Survey, USA.
18. Simonovic, S.P. & M.J. Bender 1996. Collaborative planning-support system: an approach for determining evaluation criteria. *Journal of Hyrology* **177**, 137-251.