A CONFIGURABLE FRAMEWORK FOR MULTIDISCIPLINARY ANALYSIS INTEGRATION AND MANAGEMENT

Jean-Yves Trépanier , François Guibault , Benoît Ozell , Djamel Bouhemhem
Centre de Recherche en Calcul Appliqué (CERCA)

Keywords: *MDO, framework, data management, integration*

Abstract

This paper presents the data model developped in the context of the VADOR application framework project. The purpose of the VADOR framework is to enable the seamless integration of commercial and in-house analysis applications in a heterogeneous, distributed computing environment, and to allow the management and sharing of the data in order to provide support to MDO. A multi-tiered client-server architecture has been devised, which comprises a client GUI for interactive data definition and execution launching, separate data and execution servers, and autonomous remotely executable application wrappers. The data model is at the core of the system and its contents and relations constitutes one of the main deliverables of the project.

1 Introduction

Engineers performing design-and-analysis activities in large aerospace companies are increasingly relying on computational-based approach to analyze product performance and to guide the design process. Their toolbox is made up of a large number of heteregeneous COTS and legacy applications which all together constitutes the engineering expertise of the design team.

The resulting complexity of this heteregeneous computational environment build over decades inhibits seamless application of multidisciplinary analysis and optimisation (MDO) practices [6] and brings significant challenges in the area of collaboration, data sharing and data management. In this context, there is a need for a software infrastructure which will facilitate collaboration and data sharing, while providing comprehensive data management capabilities in line with modern information technologies standards. This is the subject of the VADOR (Virtual Airplane Design Optimisation frameRk) project. The specific objectives of the VADOR project are:

1. To develop a state-of-the-art software framework capable of supporting an MDO paradigm in a collaborative design environment.

2. To implement, within the framework, data management capabilities to closely follow the design data used and shared by the design team.

At the center of this framework lies the relational data model, which describe the information to be managed by the framework and their relations. The present paper provides a detailed description of the relational data model which has been developped for the project and gives example of use of this data structure in typical engineering situations. The next section will briefly describe the overall architecture of the framework and the data model will be presented in section 3. Conclusions and future projects will be given in the last section.

2 Architecture

Figure 1 illustrates the architecture which is composed of the following elements: the Graphical User Interface (GUI), the Librarian Server, the Executive Server, the Database Management System (DBMS) and the CPU Servers. Communication between servers is socket-based, and the concept of object serialization is used to transfer components across the system. The various elements composing the system are described in the following.

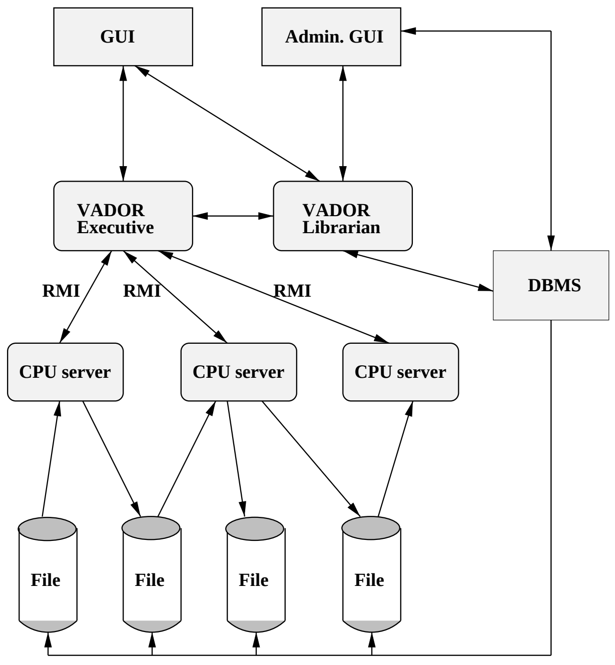

Fig. 1 VADOR architecture

2.1 Graphical User Interface

The VADOR software has been designed to be used by a large number of engineers working in collaboration on a design project and accessing the system through a graphical user interface (GUI). The GUI is a Java program running on the user's machine which provides an interface between engineers and the VADOR services. The first class of services concerns the data and process definition and the tools registration. This is performed via four different tools called *BUILDERS*. The second class of services include data management services and data inspection tools. These two classes of services are described below. Figure 2) illustrate a typical views of data and processes proposed to the VADOR users.

2.1.1 *Data and Process Definition*

The GUI provides the user with tools to contruct new *DataComponents type* and to define new *StrategyComponents*. The tool used to define Atomic *DataComponent* type is called the DCBuilder while the Composite DCBuilder is used to describe the composition of data into a hierarchical tree structure.

The tool used to define Atomic *StrategyComponents* is called the Atomic StrategyBuilder while the Composite Strategy Builder is used to describe processes including loops and if constructs.

2.1.2 *GUI Data Management and Inspection Services*

The data management services provided through the GUI includes a data classification layer and an automatic naming scheme for *DataComponent* and data files created by the system. Using the information contained in *DataComponent* attributes, the VADOR software allows users to trace the upstream history of a given piece of data including the creator of the data, the programs used to create the data and the input data used. Users can also be informed on the downstream influence of a given piece of data by asking which data has been produced using a given piece of data as input. The result is a data management system which provides a comprehensive documentation about data dependencies and data influences.

2.2 Librarian Server

The Librarian Server is a Java server program providing services for the handling and archival of Components. The Librarian stores permanently the components in a relational database using the JDBC driver. The DBMS currently in use is the MySQL database system. It is important to emphasize that the present architectural design separates the engineering data usually contained in data files, from descriptive information. Only the descriptive information, or metadata, will be stored in the relational database. The users data

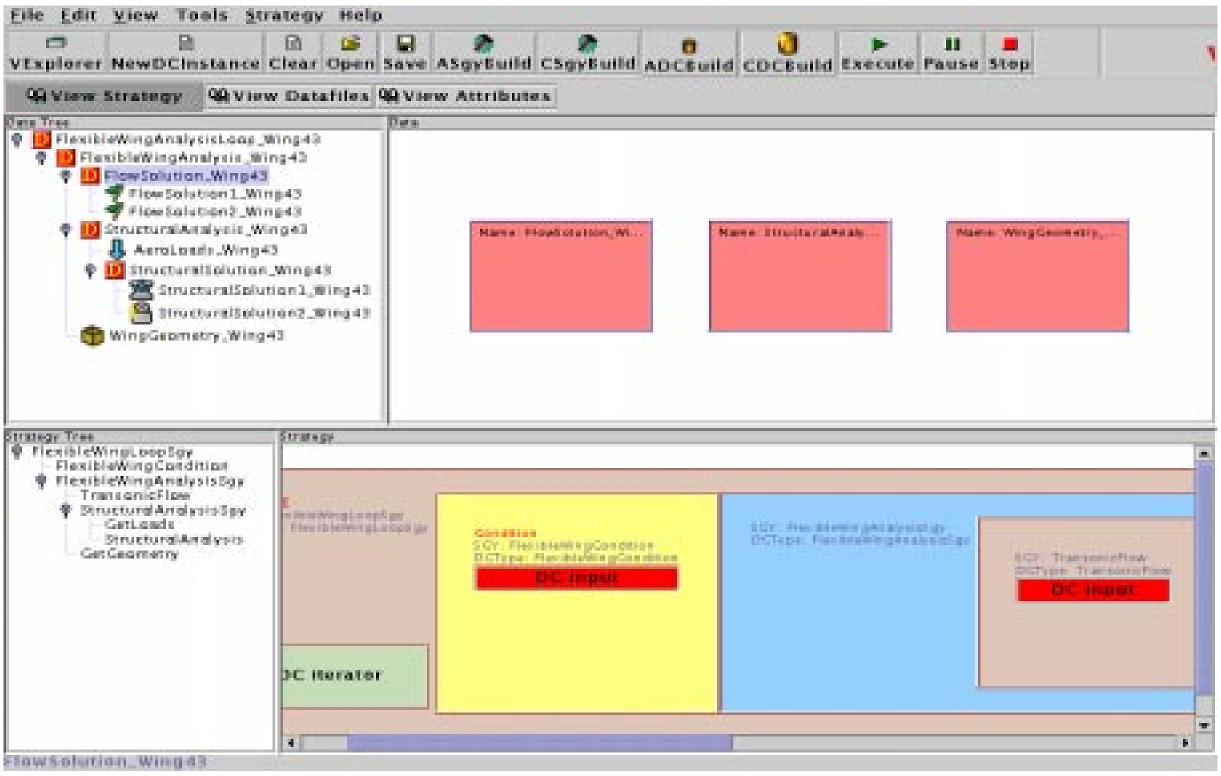

Fig. 2 A view of VADOR GUI

files (potentially large files) will usually reside where they have been created by the application programs.

2.3 Executive Server

The VADOR Executive is a Java server program that manages the execution of *StrategyComponents* to create *DataComponents*. It answers the needs of process automation in a heterogeneous distributed environment. The Executive is a multi-threaded server capable of handling multiple tasks. The Executive interacts with the Librarian to retrieve *DataComponents* to be created and to update the database contents after execution.

2.4 CPU Servers

CPU servers are JAVA programs which wraps legacy programs on specific machines. The CPU Servers have the responsability to run analysis programs with a list of I/O files transmitted by the Executive Server. The CPU Servers also have the responsability to get the input files required for the execution and put the output files to required locations after execution.

3 Data Model

One of the most important characteristics of the VADOR framework is that the framework does not manage detailed engineering data but rather metadata, i.e. references and information about the detailed data. In the present version, the framework manages metadata about datafiles stored on the network. The metadata model includes references to these data files added with a set of attributes for data management and data dependencies.

3.1 Data types

We believe that an effective framework enabling seamless MDA should provide users with a flexible and configurable data model. In order to provide this capability, the VADOR framework is based on user-defined data types. As a first step towards data standardization, data types are defined by the user to encapsulate each type of datafiles that engineers are currently using in their processes. The objects created to encapsulate the data are called in the VADOR framework *DataComponent*, or DC. A *DataComponent* definition table will be used to store in a relational database all the user-defined data types supported by the system (details about the database tables will be given below). The definition of new data types in the system is a simple task requiring only to specify a name for the new type and optionally a description for the data type. The *DataComponent* objects, which are encapsulating one and only one datafile, are called the Atomic DC.

3.1.1 Visualisation tools

As part of the definition of the basic data types, editing and visualisation tools can be defined which will then becomes available for data inspection through the framework.

3.2 Composite data types

Atomic DC in the VADOR system encapsulate one data file of a known type. The description of more complex groups of files is performed by allowing user-defined hierarchical composition of *DataComponents*. Hierarchical compositions are defined and named by the user one level at a time. The definition of Composite *DataComponent*, or Composite DC or CDC, contains a list of Atomic DC types or Composite DC types previously defined in the system. Although the users are free to define Composite DC as they wish, some rules will need to be followed in relation with process definition as discussed below in section 3.4.

3.3 *DataComponent*

In summary, *DataComponents* are objects that encapsulate design-and-analysis data which are usually contained in data files. They comprise an appropriate set of attributes required for data management. A *DataComponent* can be atomic, i.e. only encapsulate one data file, or composite, i.e. encapsulate other composite or atomic *DataComponents*).

3.4 Programs and processes

In addition to the encapsulation of the data at various levels of complexity, the VADOR system provides a model for the encapsulation of the programs and processes used to create the data and provide mechanisms to logically link data and processes at an abstract level. In the system, a program can be any piece of software requiring some data files as input and producing some data files as output. This can be a single executable program, a script, an interactive graphical application, etc. A process is an assembly of programs to be executed in a controlled sequence of operations, based on a known algorithm. Programs and processes can be defined in the system. They are fundamentally defined by the type of DC that they produce and they might require some input data of one or more specific DC type.

3.4.1 Programs

Programs are usually executable legacy software to be executed on a specific machine on the network. The execution time required to run these programs can vary from a few milli-seconds to some days, depending on the specific engineering analysis to perform. Individual programs or script are first defined by the user in the system and encapsulated in objects called Atomic *StrategyComponents*, or Atomic SC. They are defined by giving a name for the Atomic SC, by specifying the Atomic or Composite DC type that they produce and by listing the Atomic and Composite DC type required as input. By convention in the VADOR system, an Atomic SC produce only one DC, which might however be a Composite DC which encapsulates many files. It is thus necessary to define Composite DC type to encapsulate all the output files of every program to be used in the system. We will illustrate the definition of an Atomic SC using a simple example illustrated on fig. 3. In the figure, the program P1 needs two input files of type A and B and produces two output files of type C and D. The description of this process in the VADOR system involves the following steps:

1. Definition of four Atomic DC type encapsulating the data files of type A, B, C and D.

2. Definition of a Composite DC type encapsulating the data created by the program P1, for example a CDC type C-PLUS-D, containing respectively one C and one D type DC.

3. Definition of the Atomic SC encapsulating the program P1 producing a Composite DC of type C-PLUS-D and requiring as input two Atomic DC of types A and B respectively.

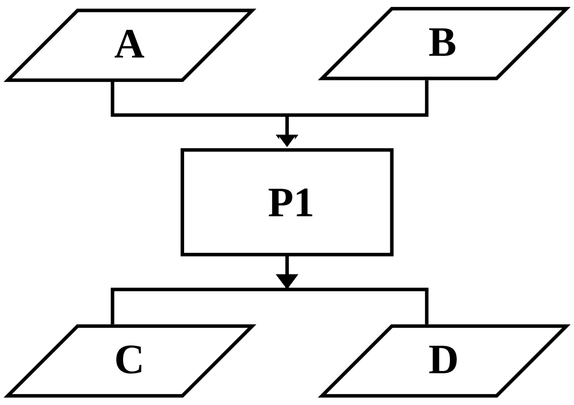

Fig. 3 Example of an Atomic Process

These three steps are performed by the user on the GUI using simple dialog boxes to define the data and the programs.

3.4.2 Processes

A process in the VADOR framework is a controlled sequence of program execution which is used to produce complex data. Processes are encapsulated in Composite *StrategyComponents* objects, or Composite SC, or CSC, which are defined by the users using standard elements of structured procedural programming languages: sequential blocks, parallel blocks, if constructs and controlled loops. Fig. 4 shows a typical process which will be used to illustrate the definition of a Composite SC.

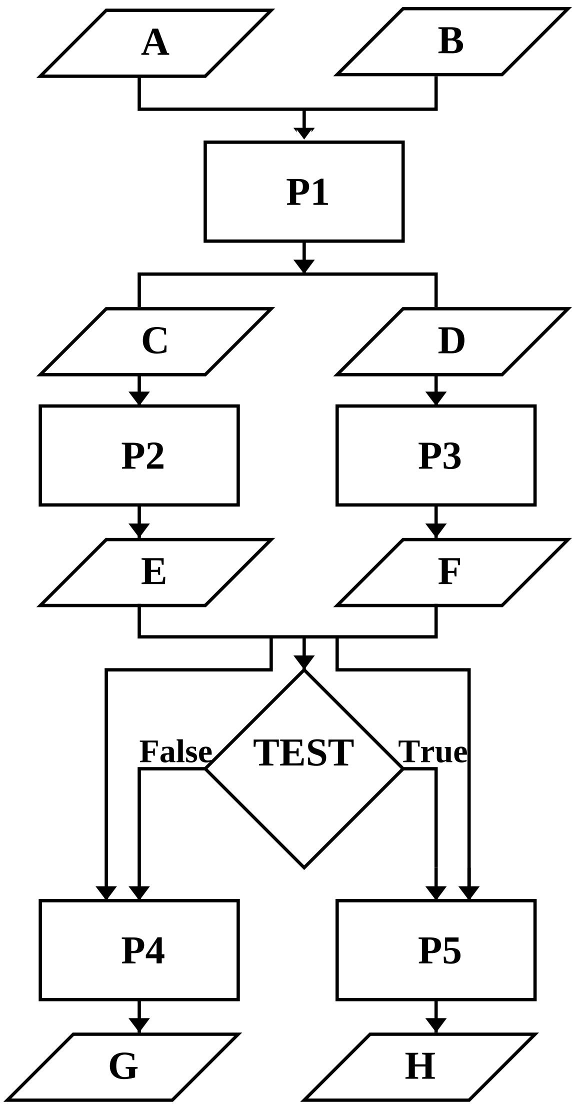

Fig. 4 Example of a Composite Process

First, we will give a description of the data produced by the process. We need to define the DC types A, B, C, D, E, F, G, H, and C-PLUS-D. If we consider that the A and B types are not part of the process but are rather inputs to the process (this choice is left to the user), then we need to define a composite DC type C-TO-H containing one C-PLUS-D, and one E,F,G and H respectively. So the composite DC type C-TO-H encapsulate 6 data files.

Now, let's consider the description of the process itself. Globally, the process is modelled as a sequential process composed of three blocks. In the first block of the sequence, the process P1 produces a C-PLUS-D type DC using an A type DC and a B type DC as input. In the second block of the sequence, the processes P2 and P3 are executed in parallel, the P2 process producing an E type DC using a C type DC as input and the process P3 produces an F type DC using a D type DC as input. The third and last block of the sequence contains an if construct with its associated True and False branches. The test uses data contained in both the E and F type DC to branch. In the FALSE branch, the P4 process produces a G type DC using a E and a F type DC as input while in the TRUE branch, the P5 process produces a H type DC using the same input. It is clear that at the end of the process, either the G or the H type DC will be pointing to an inexisting data file.

Note that there is clear separation in the system between the sequence of execution and the dependencies. For example, the system allows one to describe a process which will use as input a file which will be produced later in the process. This provides flexibility in the description of complex process. However, at execution time, the process may not be able to execute and appropriate messages will be generated.

3.5 *StrategyComponent*

In summary, *StrategyComponents* encapsulate the programs and the analysis methodologies or processes. They represent the basic methods and the data flow required to transform data in a given process. *StrategyComponents* are constructed using the basic elements of a procedural language, enabling the description of complex conditional processes. A *StrategyComponent* has a *type*, which indicates the type of *DataComponent* that the strategy can create.

3.6 Relational Database

In the VADOR framework, a Database Management System (DBMS) is used to store, maintain and provide access information for Components. The DBMS currently used by the framework is the MySQL DBMS [8] which stores information using a relational model - the most popular data model on which DBMSs are built. A view of the various tables and their relations used by the system is reproduced on fig. 5. Divising a schema for a relational data model is not a simple task. The tables contents and their relations currently in used by VADOR are the results of a few iterations and will probably require adjustments in the future. The final schema will be one of the main results of the VADOR project since it defines the core of the system on which everything else is built.

3.6.1 *Data and Process definition tables*

Referring to fig. 5, the first table, called the DCDefinition Table stores the definition of Atomic and Composite DataComponents Types. When the DC type is Composite, the elements of the composition are listed in the Table DCElements.

The DCDefinition table identifies each type of data in the VADOR system by a unique typeID, known as the Primary Key (PK) of the table and associates with this typeID a typeName, a typeIcon, an IsAtomic field and a Description field. The typeName is given by the user and is required to be unique. The typeIcon refers to an icon file managed by the system which will be used by the GUI to provide visual recognition of the *DataComponents* based on their type. The IsAtomic field allows to distinguish between Atomic and Composite DC. The Description field is a string given by the user at the moment of the type definition which will be later accessible through the GUI for information on types. Composite DC require the table DCElements to store their contents. Every element appearing in a composite DC type has a unique dcElementID, has a dcElementName, has a ChildID, has a counter and has a typeID.

Table 1 shows the entries required in the DcDefinition table to describe the type A, B, C, D and C-PLUS-D discussed above. The description should be somewhat longer to fully describe the type. Also, the typeName should be choosen carefully to match engineering practices and an Icon should be designed for visual support. The Atomic types A, B, C and D are completely described by these four lines in the table. The Composite type, C-PLUS-D requires a description of its composition in the DCElements table. The two entries required in the DCElements table to describe the C-PLUS-D composition are shown in table 2. The foreign key (FK) typeID identifies the relation between these two definition tables. The two lines having a ParentId=5 in the DCElements table describe the composition of the typeID=5 in the DCDefinition table. The counter allows to order the elements of the composition. The first element has a typeID 3, i.e. is a C type DC and the second element has a typeID=4, i.e. is a D type DC.

Programs and processes are defined in the Table named StrategyDefinition. A relation is present between the StrategyDefinition Table and the DcDefinition Table, indicating explicitely the DC Type that the strategy can create. A Composite StrategyComponent is described by its elements stored in the StrategyElements Table. The example strategy shown in fig. 4 will be used to illustrate how programs and processes are described using relational tables. The Table 3 shows the entries in the StrategyDefinition table used to describe the process. First, the Atomic programs are described and are given a unique strID. Note the typeID(FK) field, which is indicating the DC type produced by the strategy, for example the typeID=5 (C-PLUS-D) produced by the P1 program. The complete process is described by the composite strategy named P1toP5. The Table 4 completes the description of the process. First, the Table 4 contains the elements of the P1toP5 strategy, which is a sequential strategy with strID=6. Then, the strElementId from 2 to 4 describe the contents of the SEQ1 strategy, which are the P1, the PAR1 and the TEST1 strategies respectively. Then, the contents of the PAR1 and the TEST1 strategies are described respectively.

The above described four table are the core of the data and process definition in the VADOR framework. Note that users will not in general need to deal with the database tables but only with the GUI presented previously.

3.6.2 *MetaData tables*

The main table for metadata storage and management is the DcInst table which stores instances of DataComponents. As described in section 3.1, instances of DC encapsulate single data file or groups of data files and attach to these files metadata information. The table has two fundamental links with the Definition tables. First, the DcInst table has a relation with the DcDefinition table in order to uniquely define the type of data described by the instance. Second, the DcInst table stores the strategy used to create the data through a relation with the StrategyDefinition table. The DcInst table also contains the descriptive information about the instance required for data management. When the DC type is composite, the elements instances of the composition are listed in the DcElementsInst table.

4 Conclusion and Future Work

A data model and framework architecture for MDA have been described in the present paper. The framework has been designed to suits the needs of a large engineering department where engineers are working concurrently on an engineering design project. The framework is highly flexible and configurable and is expected to be adaptable to the needs of every engineer. The framework enforces standardization of the data and methods, which will result in a self-documenting system. The attributes tagged to the data and strategies will provide an easy access to all critical information concerning the data. The object-oriented methodology has been used in the development of the framework. The implementation is done using the JAVA language, a choice motivated primarily by portability and internet capabilities, as well as high-level language capabilities and tools.

Since all the data and processes are being defined by the user using an approach similar to a programming activity, the VADOR software is

typeId(PK)	typeName	typeIcon	IsAtomic	Description
1	A	A.gif	Yes	This is type A
2	B	B.gif	Yes	This is type B
3	C	C.gif	Yes	This is type C
4	D	D.gif	Yes	This is type D
5	C-PLUS-D	C-PLUS-D.gif	No	This is type C-PLUS-D

Table 1 Examples of entries in the DCDefinition table

dcElementId	dcElementName	ParentID	counter	typeID (FK)
1	CinC-PLUS-D	5	1	3
2	DinC-PLUS-D	5	2	4

Table 2 Examples of entries in the DCElements table

strId(PK)	strName	structure	method	typeID(FK)
1	P1	Atomic	P1.exe	5
2	P2	Atomic	P2.exe	6
3	P3	Atomic	P3.exe	7
4	P4	Atomic	P4.exe	8
5	P5	Atomic	P5.exe	9
6	SEQ1	Sequential	none	none
7	PAR1	Parallel	none	none
8	TEST1	TEST	test.exe	boolean
9	P1toP5	Composite	none	10

Table 3 Examples of entries in the StrategyDefinition table

strElementId	StrElementName	strID(FK)	strParentID
1	SeqinP1toP5	6	9
2	P1inSeq	1	6
3	ParinSeq	7	6
4	TestinSeq	8	6
5	P2inPAr	2	7
6	P3inPAR	3	7
7	P4inTest	4	8
8	P5inTest	5	8

Table 4 Examples of entries in the StrategyElements table

neutral relative to a specific application domain and the application of the framework to other domains is straightforward. This genericity is stemming from the configurability of the system.

The formal association between a *DataComponent* and a *StrategyComponent* (which is able to fill an encapsulated data file) provides a link between the behaviours and the services they provide. In addition it provides a tool for documenting the methodologies in use and promote standardization. In addition to data and process description and management, the framework provide capabilities for the automation and integration of various processes used by engineers using modern distributed computing techniques based on client-server architecture and web standarts.

The actions required by the users to define new processes and data in the system are performed during an integration phase, when new processes are interfaced to the system. In a typical every day usage, users will simply execute processes to create data and inspect the data using the visualisation tools accessed through the framework. It is important to note the clear separation between the data and the programs and processes. This separation allows to use many different programs and processes to produce functionnally equivalent data and reinforces the standardization based on the data. This is believed to be a crucial point in the support of seamless multi-fidelity simulation environments.

5 Acknowledgements

The VADOR team would like to acknowledge Bombardier Aerospace and the NSERC of Canada for their financial support of this work. Also, many thanks goes to M. François Pépin and M. David Leblond from the Advanced Aerodynamics Group at Bombardier Aerospace for numerous discussions on the engineering practices in their department.

References

[1] I. Kroo. "Computation-based design – a white paper". http://aero.stanford.edu/ComputationalDesign.html, 1996.

[2] S. Allwright. "Multidisciplinary design, analysis and optimisation of aerospace vehicles - the MDO project". In *Royal Aeronautical Society MDO conference*, 1998.

[3] A. Ndiaye, J.-Y. Trépanier, F. Guibault, B. Ozell, and B. Mahdavi. "Database requirements for an mdo software framework". In *CFD2K, 8e conférence annuelle de la Société canadienne de CFD*, Montréal(QC), Juin 2000.

[4] A. Alzubbi, A. Ndiaye, B. Mahdavi, F. Guibault, B. Ozell, and J.-Y. Trépanier. "On the use of JAVA and RMI in the development of a computer framework for MDO" In *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach(CA), September 2000.

[5] 0.A. Salas and J.C. Townsend. "Framework requirements for MDO application development". In *AIAA-98-4740*, 1998.

[6] J. Sobieszczansk-Sobieski. "Multidisciplinary design optimization MDO methods: their synergy with computer technology in the design process". *The Aeronautical Journal*, 1999.

[7] R.P. Townsend, J.C. Weston and T.M Eidson. "A programming environment for distributed complex computing. An overview of the framework for interdisciplinary design optimization (fido) project". *Technical report, NASA TM 109058*, 1993.

[8] P. Du Bois. "MySQL". *New Riders Publishers*, 1999.

[9] G. Reese "JDBC & JAVA, Database Programming, Java 1.1". *O' Reilly*, 1997.

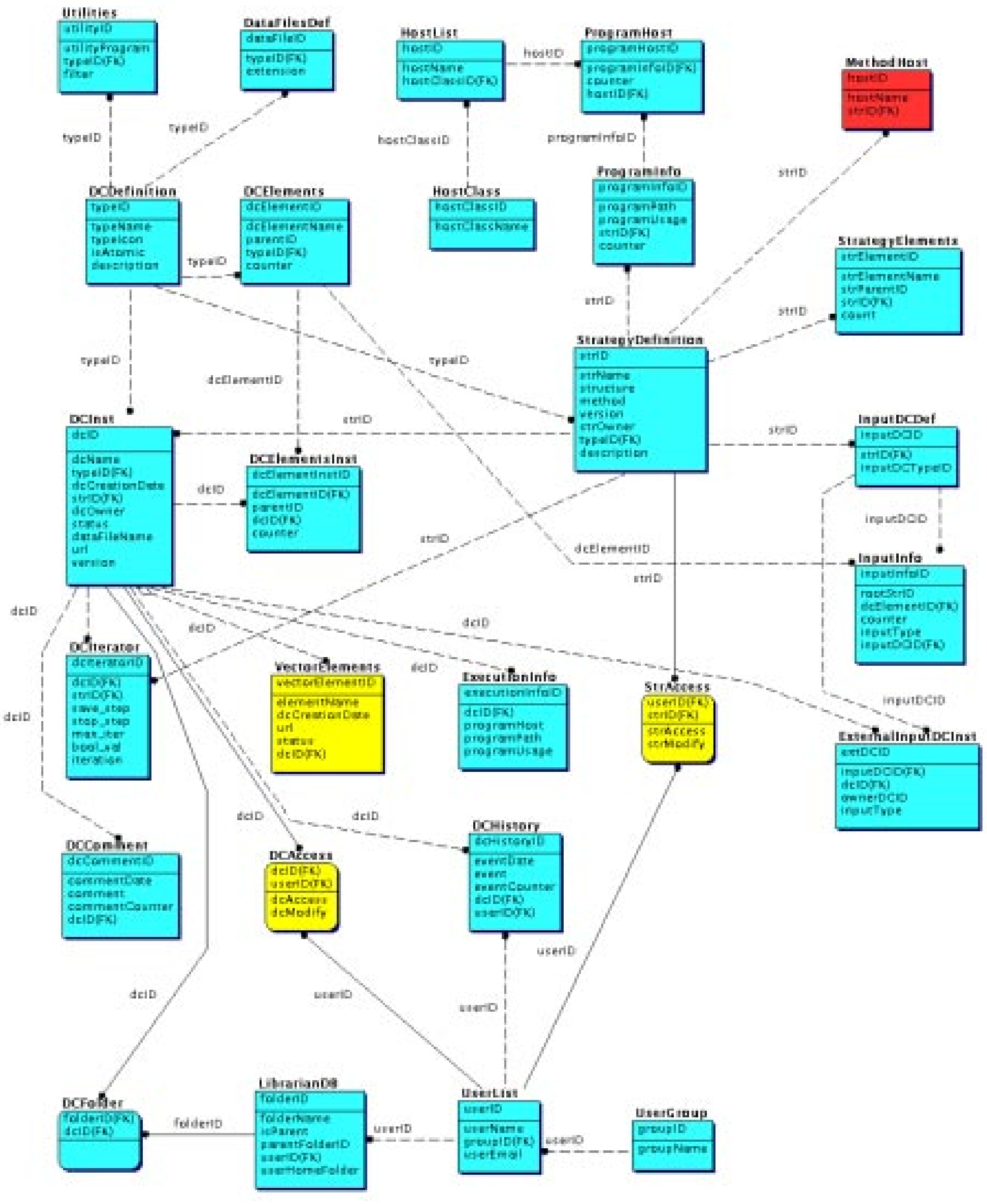

Fig. 5 A view of the VADOR relational database model