# The MetaModel: A Generic Approach to Data Warehousing

We, at the South American Data Warehousing practice, have been working on DSS projects for the last three and a half years. During this time we developed a concept that helped us make a difference in terms of added value and project effort, significantly increasing the former while decreasing the latter. This is the MetaModel, a generic approach to a Data Warehousing solution.

The MetaModel was used and evolved in several different industries and business processes as a practical and efficient way of organizing, storing and retrieving information for decision support. It has proved its value in retail, consumer goods, telecommunications and other industries, in areas such as marketing, sales, accounts receivable, executive management and combinations of these.

The goal of this document is to explain, in a format understandable by people not necessarily from an IT background, what the MetaModel is, its benefits and the ways we have been using it to leverage our Data Warehousing practice.

This is the first document of a set about the MetaModel. The following papers will be “The MetaModel : Technical Description”, that will detail MetaModel’s data structures and processes, and “The MetaModel : Implementing and Managing” , that will describe the whole process of MetaModel’s implementation in a data warehousing project.

## Introduction to the MetaModel

Conceptually, the MetaModel is a Data Model developed for achieving a generic way of processing, storing and retrieving information for decision support. Generic means it can be applied with minor, if any, customizations to several business process needs on varied industries, both for summary and detailed data. Together with the model go several applications and routines that handle, also in a generic way, the issues of loading, managing and summarizing of this information.

One important thing to have in mind is that the MetaModel is **not** and industry template. It can be parameterised to fit most templates, because it is essentially a structure waiting for names to be assigned to its parts. Once it is associated with a specific template, it takes little time for it to be up and running a DSS for that industry.

## The MetaModel and the DW Architecture

The MetaModel is a generic schema for implementing dimensional models. As such, it can be used in several places in the Knowledge Management technical architecture. In this section we will show how we have been using it.

### Data Structures

#### Operational Data Store (ODS)

If we take Inmon’s definition of an ODS, it would not be the case of using the MetaModel as such, since it is dimensional and holds historical, redundant data. That’s not compatible with synchronous replication of the transaction systems. Actually, when we need and ODS, we prefer to develop it completely from scratch.

But, by PW methodology’s definition, the MetaModel can be used as an ODS. We have some cases where the most detailed data was brought from the transactional systems into the MetaModel, appropriately transformed and cleansed, and used the MetaModel as the single, integrated source of data for all the data marts. In such cases, it can be viewed as the ODS. Actually, this has been the most common use for the MetaModel up to this date.

#### Dimensional Data Store (DDS)

Most of the DW practice in South America consists of building a central data store and generating cubes on multidimensional engines from this store. The MetaModel has been used as a DDS in situations where it was not advisable to use data marts, or when the users needed both the summary and very detailed data, and that would represent too large a volume for multidimensional engines to handle. In these cases, we would query the MetaModel directly (through custom screens) or use it for generating ad-hoc reports that the user could later work on with some query tool.

One next step on the development of the MetaModel is to use the information it already has for building a semantic layer that looks like stars for end-user access. Through this, we would isolate these users from the natural complexity of having a generic schema, without losing the benefits for the administrator.

#### Data Mart

Up to this date, we have not used the MetaModel as a tool for building data marts. Multidimensional engines have met most of the data mart needs we have come across. If we were to use the MetaModel as a data mart architecture, it would have the benefit of being 100% integrated with the data source (supposing we are using the MetaModel also as either an ODS or DDS).

#### Metadata

The main structures of the MetaModel store the definitions that are necessary to build the logical data model within the generic structures, actually forming a metadata layer. A few samples of the metadata stored in the MetaModel:

##### DDS Structures:

* Dimensions, including :
* Hierarchical structures
* Attributes

##### Performance measures, including:

* Granularity
* Aggregation methods

##### Extraction / Transformation Process:

* Data sources (for each measure)

##### Access/Analysis Layer:

* Data distribution through Data Marts (representing each Data Mart data structure)

### Processes

#### 

*Figure 1 - MetaModel and the DW Architecture - Flow of Processes*

#### Transformation

By transformation we understand the process of receiving transactional, relational data and translating this into dimensional data suitable for analysis. The MetaModel does not address this.

This part of the Data Warehousing process has been handled up to now by custom development in a 3GL or 4GL language, usually the one the client is already most comfortable with. There is no technical reason for not using an extraction/transformation tool for this process, but the cost/benefit relation of this approach has not been worth it up to this moment.

#### Cleansing

The MetaModel has some built-in cleansing capabilities, basically validation and rejection of unknown data. That would be checking the source system codes for the dimensions and accepting only those that are meaningful as were input on the MetaModel. This, together with the capability for importing translation tables, meets most demands from our clients, after the initial loading of data. Cleansing of historical data has been dealt in a case-by-case basis, usually resulting in the detection of errors in the source systems.

Some versions of the MetaModel have the capability of correction of input data by the administrator. That would be mostly substitution of a wrong code for the right one, followed by reprocessing of the rejected records.

One interesting feature is that when part of a lot of data has been accepted, it will wait in a staging area until the rest of the lot has either been permanently deleted or fixed by the administrator. In that way, we avoid the receiving of potentially wrong data by the end-user.

#### Key Substitution

From the first version, the MetaModel has key substitution features, together with its data cleansing capabilities. It allows for mapping of several different source codes to one dimension member, providing for non-integrated source systems.

Supposing that an extraction tool has already dealt with data cleansing and key substitution, the information can be loaded directly into the staging area, in order to avoid unnecessary processing.

#### Aggregation

The MetaModel has built-in aggregation functions, capable of summarization both in time and in level of detail. These functions are fully integrated with the generic partitioning system of the MetaModel, and allow for the most common summarization strategies (sum, average, etc.).

#### Partitioning

The MetaModel handles internally data partitioning based on subject, level of detail and age of data. Thus, the system can be tuned for performance on most used information, without needing huge computational resources or letting fact tables grow out of control.

## Why Use the MetaModel

This section shows what are the main reasons for using the MetaModel as a solution to the ODS or the DDS layer of the Knowledge Management architecture. We must keep in mind that, so far, it only works in an architecture where the access to the Data Warehouse data is provided by a Data Mart built with a multidimensional tool or to custom screens querying directly the MetaModel. Refer to “The Future” chapter to know the next version of the MetaModel, that will allow it to work with ROLAP and Query Tools.

Some of the benefits of using the MetaModel as a component of a DW architecture include:

* Its generic physical structure adapts easily to changing information need, easing the growing process of the DW architecture;
* There is no need to develop specific data loading/distribution/summarization applications, since its generic procedures are prepared to handle any kind of information that fits into its structures. It brings the additional benefit of having only one set of data loading/distribution/summarization applications to monitor, easing the DW administrator’s work;
* It eases the prototyping process, since the physical infrastructure is already built and supports continuous changes; and
* It implements some functions of the DW administration layer without using a specific tool, since it provides DDS metadata and storage structures.

### Project Stages and the MetaModel

Since it provides a generic structure for loading, storing and retrieving information, the MetaModel changes the traditional structure of a data warehousing project, as described below.

#### Phase 0

This phase is technology independent, therefore it is not affected by the MetaModel - except for the fact that its use as a solution must be considered.

#### Analysis

Using the MetaModel as a solution for data warehousing allows most of the efforts originally devoted to technical issues (e.g. Data Design) to be redirected to the Analysis stage. More emphasis is given to the Performance Measurement modeling process.

It also eases the prototype process: in a conventional approach, one must wait until the modeling is done to begin the physical database design. Only after this it is possible to get some data into the database in order to trigger the prototyping process, that will gradually allow the users to validate the information available through the data warehouse architecture.

On the other hand, the usage of the MetaModel provides a ready-to-use database structure, allowing designers to anticipate the prototyping process start. Actually, this process can start during the analysis phase - continuous changes (inherent to this kind of process) are well supported since they can be done through a set of administration applications (not involving database maintenance).

#### Design

Each phase of the design stage is differently affected by the usage of the MetaModel, as shown below:

* The Data Design phase is the most affected one, becoming much shorter. Considering that the dimensional’s database physical structure (either the ODS or the DDS, depending where the MetaModel is being used within the Knowledge Management Architecture) is already built, this phase consists mainly of filling these structures through the administration applications in order to implement the model developed in the Analysis stage. There is no need to develop a Physical Data Model.
* The Data Transformation Design also takes advantage of the MetaModel. In fact, the MetaModel does not address transformation issues: it expects to receive data already cleansed and transformed, ready to be stored in a dimensional database, and formatted in a standard layout. Considering that the interfaces have the same layout regardless the information that it carries, it is possible to start the interface program’s specification just after the end of the analysis stage. As mentioned earlier, in most cases this work is left to client staff.
* The Presentation System Design is usually our main focus in the Design Stage, because it is barely affected by the usage of the MetaModel. Again, as in the Analysis stage, the MetaModel redirects the focus of our work to business issues (what data is going to be analyzed and how this analysis is going to be presented), since most technical issues are already addressed by the generic structures.

#### Construction

The impacts of the MetaModel in the construction are analogue to the ones in the Design stage: more focus on the Presentation System Construction, since client staff usually handles Data Extraction/Transformation and Data Loading/Distribution/Summarization is implemented through MetaModel’s built-in routines.

This approach also addresses more time for testing and tuning the whole system: the data tests can start as soon as the interfaces are built, turning the extraction/transformation system construction into the critical path of the project.

#### Implementation

Not affected by the usage of the MetaModel.

## Main Structures of the MetaModel

#### 

*Figure 2 - MetaModel Kernel Structure*

#### Dimension Table

Through the dimension table we inform the system of all the existing dimensions of the model generated in the analysis phase. This table doesn’t have the actual dimension members - only the dimensions. So, for a communications company, a typical dimension table could be:

|  |  |
| --- | --- |
| **Dimension Code** | **Dimension Name** |
| 1 | Client |
| 2 | Telephone |
| 3 | Service |

That tells the system that there are clients, telephones (brands and models) and services that the company offers besides the lines. With that we can start creating dimension combinations and measures on the MetaModel.

#### Dimension Instance Table

This table has the actual dimension members (the name “dimension instance” comes from object oriented analysis). Members for all dimensions are listed here, and there’s an entry for each one. Hierarchy is represented through a relation between two members of the same dimension. So, for the dimensions we created before, a possible dimension instance table would look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dimension Code** | **Instance Code** | **Instance Name** | **Level** | **Father** |
| 1 | 1 | Total Client | 1 |  |
| 1 | 2 | Northeast | 2 | 1 |
| 1 | 3 | Southeast | 2 | 1 |
| 1 | 4 | John Doe | 3 | 2 |
| 1 | 5 | Jennifer Eight | 3 | 3 |
| 2 | 1 | Total Telephone | 1 |  |
| 2 | 2 | Motorola | 2 | 1 |
| 2 | 3 | Ericsson | 2 | 1 |
| 2 | 4 | Elite | 3 | 2 |
| 2 | 5 | Startac | 3 | 2 |
| 3 | 1 | Total Services | 1 |  |
| 3 | 2 | Conference Call | 2 | 1 |

The level column indicates how high in the hierarchy a given instance is. For example, all totals are level one and the client dimension would be like this:



The numbers in parenthesis are the levels of the dimension instances. The problem of a normalized hierarchical structure is usually addressed in the data-marts, which can be less normalized and receive the dimension structure generated by a process already implemented in the data warehouse. That process takes the dimension tree as it is represented in the MetaModel, i.e., vertically, and translates it to a horizontal representation better suited for on-line aggregation, most common on ROLAP tools. The information generated by this process can also be used for a direct query to the MetaModel.

#### View Table

This table indicates the valid dimension combinations on the Data Warehouse. For performance reasons, this table is denormalized, and, instead of having an n-n relation with dimension, it has space for dimensions from 1 to a fixed maximum (currently 16). This maximum limits the number of dimensions that can be applied simultaneously to a variable, time not included. Time is assumed to be always a dimension and has its own treatment.

So, following the previous example, the view for the variables dimensioned by client and service would correspond to a record like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **View Code** | **View Name** | **Dim. 1** | **Dim. 2** | **Dim. 3** | **...** | **Dim. 16** |
| 1 | Client x Service | 1 | 3 | 0 | ... | 0 |

The zeros mean that that dimension slot is unused. Nulls could not be used because the entire row has a unique constraint, which doesn’t allow null columns. What the row above means is that all variables of view one are dimensioned by client first then service.

#### Variable Table (Measures)

This table holds the description and any other metadata needed for the definition and calculation of measures. As such, it holds information as the source of the data (together with other tables that log data imports), how long it should be stored, summarization rules and a few others. Suppose that the example we have been building up to now has a measure of the sales by client and service. It could be represented in the variable table like this:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Var Id** | **Var Name** | **Description** | **View Id** | **Time Cons** | **Hier. Cons** | **Time Detail** | **Source** |
| 1 | Sales | Amount of sales of services for each client | 1 | SUM | SUM | Monthly | Sales System |

So, the variables Sales, the first created, is dimensioned by client and service, aggregates on time by sum, which is also the case for the hierarchical summarization, has a maximum granularity of monthly data (that will be used for data cleansing) and comes from the Sales System. The last field usually is a code referencing a table of data sources. That table holds information about the kind of source (system, external, etc.), the frequency of data generation from that source, the way data will be received from that table (flat files, gateway, etc) and who is responsible for information from that source.

#### Time Series Tables (Facts)

Finally, the actual data is stored in these tables. The time series tables are actually physical partitions of one single logical entity. So, they all have the same general structure. This structure is optimized for access by time, related variables and the first dimensions in the view that the variable belongs to. That means that some care should be taken when defining views, so that most-used dimensions come first. So, the data for Sales on a typical time series table looks like this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **View Id** | **Var Id** | **Date** | **Dim. 1** | **Dim. 2** | **Dim. 3** | **...** | **Dim. 16** | **Value** |
| 1 | 1 | 04/97 | 5 | 2 | 0 | ... | 0 | 40 |

That line means a sale (var. id 1) to customer “Jennifer Eight” (dimension 1 is client on view 1, and client 5 is “Jennifer Eight”) of the service “Conference Call” (dimension 2 is service, and service 2 is “Conference Call”). All variable values can be stored in this format. That’s its main advantage, and it also has a fairly good execution plan for retrieving data, much like those generated for traditional star schemas.

The main drawbacks of this format are:

* It doesn’t work well for comparing two measures. That’s because each row on the table holds a single measure. Actually, the traditional star schemas don’t perform well either when comparing measures from two different stars. For the proposed solution to this problem, refer to the “The Future” chapter, under the “New Features” section.
* It’s more complicated for the end-user. The same features that make the MetaModel easier to administrate, i.e., generality, make it much harder to understand for the end-user. We have usually addressed that by hiding the queries under custom screens or through the use of data marts. The proposed solution for this is also described in “The Future”, under “New Features”.

#### Other Tables

There are several other tables on the MetaModel, for dealing with mapping data sources, controlling data marts and handling internal processes. Also, not all the fields of the main tables are described here. Another document, “The MetaModel - Technical Description” will describe all the tables and processes in detail. That document is intended for persons with a good knowledge of IT, specifically database architecture, and will not address any project-related issues.

#### Technology Issues

The architecture described above is fully implemented in Oracle 7.3 on a Digital AlphaServer, with the batch processes running partly in PL/SQL and partly in Pro\*C. The administration tool is currently written in Visual Basic with Oracle Objects for OLE, and we have a PowerBuilder 5.0 version under development. It manages all the metadata tables, but there is no screen for updating directly the time series tables. That’s on purpose, so that no one will touch the data itself. That should be done either automatically by extraction tools and programs or by a specific data-entry application, designed for other users.

### The MetaModel and the Star Schema

The MetaModel was created for reasons similar to the Star Schema, namely storing data in a format better suited to decision support analysis. But, while the MetaModel was structured for use in any situation, growing with the same structure, the Star Schema has to be customized for each case.

That is both an advantage and a disadvantage. The benefit of this is that the Star Schema, being specifically adapted to the problem it is addressing, is easier for the user to understand. However, it is harder to maintain and grow.

That is not because the Star Schema is more complex, which it isn’t. Actually, it is incredibly simple. It is basically because there is no general star schema solution, over which can be created reusable algorithms, like the ones for loading and summarizing data on the MetaModel. So, there is a big effort on coding these algorithms for every new system built. Also, as there is no standard for dimension and fact tables, the administrator has to keep system scope growth under tight control, whereas with the MetaModel the very design enforces and documents these standards.

The cost of these standards is the added level of conceptual complexity, which is being addressed by the suggested improvements on the model, described in the chapter “The Future”.

## Success Cases

### Retail (MLD)

This project objective was to provide the management of a large department store with a practical tool to analyze Oracle GL’s data. The information model was straightly derived from the Oracle GL’s structure, turning accounts into performance measures and components of the account key into dimensions.

In this case, the MetaModel’s metadata structure related to the extraction/transformation process stored the relationship between data warehouse’s measures and GL’s accounts. This information was used by the interface programs (built in PL/SQL) to retrieve the transactions from GL and store them into the MetaModel on a daily basis. It means that a simple change in these metadata (made through a set of administration tools built in SQL\*Forms 4.0) was enough to change the data loaded into the data warehouse (e.g. to add a new measure).

Considering that the information was initially stored in the transaction level, it was necessary to use the generic summarization routines that managed to aggregate these transactions to daily and monthly totals and also created the higher levels of dimensions’ hierarchies.

A pretty flexible application was built to display the data warehouse data to end-user, dynamically creating the SQL necessary to query the MetaModel (in this case, no multidimensional tool was used).

### Health Holding (Golden Cross)

This project was for the holding of a group of health-related activities (hospitals, labs, insurance etc). As a holding, they had several performance measures (the first lists had around 700) from several different sources. These sources generated data in several different formats with non-unified codes for the information, and a high level of dirty data.

The loading application for the MetaModel had to be custom-rebuilt for this case, adding several validation features and a complete front-end that allowed the system administrator to correct all most-common errors. That permitted the system to keep receiving data while the extraction programs were being fixed, because the administrator had flexibility enough to correct minor mistakes. When you have several people from several different companies writing extraction programs (as was the case here), this can be the difference between a system that delivers the information and another that does not.

This system was built over a Digital AlphaServer 1000 running Sybase. The data sources ranged from Unisys mainframes to DBF files and the end-user access to the information was made through custom applications that worked on a multidimensional database built with Pilot Analysis Server.

### Consumer Goods (Kaiser)

This project was the reconstruction of an EIS they had developed some years before. The company had its production environment in a package that ran on Progress. Most of the information came from this package, with some data also coming on a Nielsen database (mostly market-share information). These were being distributed through DBF files accessed by pre-formatted screens.

As an added complication, the company was in the process of selecting the enterprise standard for RDBMS in preparation for a SAP implementation. So, we had to develop a solution in a low-cost platform and guarantee it would be portable later.

After some months working in the analysis phase, we came out with a set of measures and dimensions that had to be extracted from the source systems (in Unix), transported to the Data Warehouse environment (originally a PC running Windows and an application developed in Powerbuilder with Personal Sybase), where the data would be validated and loaded. This information would later be transported to a multidimensional database, which was used for presentation. This database had to be replicated across seven plants throughout the country. This was implemented using a snapshot capability of the database coupled with a satellite link for transmission of the information.

This structure allowed for efficient presentation on every site, and the whole process had to be finished daily by 8:30 AM, when the users started accessing the information. This left a 2-hour processing window, which required lots of efforts in programming for efficiency.

Currently, the system is under migration to a HP-UX/Informix platform, using Powerbuilder 5.0 as a developing tool for administration. The end-user access to the information was made through custom applications that worked on a multidimensional database built with Pilot Analysis Server.

### Heavy Industry (CSN)

This project was the first to include several business areas in Brazil’s largest steel company. The first step aimed to develop the Data Warehouse’s infrastructure and spread the new culture through the company, so it included a small volume of summarized data that were extracted from Excel spreadsheets, stored into the MetaModel (built in Oracle) and queried through a multidimensional database (built in Pilot Analysis Server) by the high executives.

The project’s success triggered other two projects : one in the credit control area and another in the production area. Both of this projects dealed with large amounts of data, with a very small processing window available to the loading process. The detailed data loaded during these two projects gradually replaced the original spreadsheet data, in a way that transactional systems could become the real source of data to the corporate DSS.

An ODS was developed to hold detailed information about client’s orders, and the front-end application (custom screens developed with Pilot LightShip) managed the navigation between the multidimensional database an the ODS (implementing a drill-through mechanism).

### Telecommunications (Miniphone)

This project was the first to stress test the MetaModel with big dimensions and huge volumes of data. It was focused on Marketing and Sales for a cellular phone company. As is normal in such companies and areas, the dimensional model had several dimensions for defining client profiles, along with a huge client dimension. Given the fact that daily data was requested, the average load is 100000 records a day, with weekly and monthly peaks of a few million records.

For dealing with that amount of data, several changes had to be made in the MetaModel. For the first time we used development in C, together with a complete rewriting of the data distribution code. It allowed for a very efficient and flexible process, which we used to better partition the information received.

All this information was also being passed to several multidimensional databases, used as presentation data marts. As it would be impossible to load the full detail of information received directly into the data marts, a summarized version of the information was passed, and a drill-through mechanism was implemented. This mechanism identified where inside the data the user was and created a query running over the Data Warehouse to generate the detail of that information. As the results of this query were saved on database tables in each user’s schema, the analysts could keep searching inside the information received or use it for promotions or customer surveys.

The Data Warehouse runs on a AlphaServer 4100 with 3 processors, 1 Gb RAM and 100 Gb hard disk. The total size now is around 70 Gb. Currently the EIS module is under development, together with other measures of the Sales area, that could not be obtained before. That should bring the Data Warehouse to the 100 Gb size, mostly daily data.

The great benefit of using the MetaModel on this client was that, as we went on adding new information to the system, there was no need to recode the applications for distributing and summarizing data.

## The Future

We are starting to move to a new level of projects now, which will demand more and more on the technical side, as on the management side. These new projects usually have different tools as front-end, both ROLAP and MOLAP, different data sources, such as packages, and much bigger sizes. In order to keep up with these changes, we are planning and implementing several changes on the MetaModel.

### New Features

#### Support for ROLAP

Because of the growing database sizes we are facing, the choice of ROLAP tools for presentation will be more and more common in the next months. Up to this date, the MetaModel has been used either with a custom-developed front-end or with MOLAP tools, that hide its inherent complexity. Also, its internal structure does not lend itself well to SQL-generated DSS queries. Basically, it does not perform well for comparing different measures. From a ROLAP tool point of view, it looks like something where each measure is in a different star. That’s because it has normalized measures, with a foreign key to the Variable table.

The proposed change is to create several value columns in the Time Series table, thus leaving it very similar to a fact table in a star schema. That would be a great benefit for both querying and summarization. The cost is a little more complexity for the distribution process, but surely worth it.

#### MetaStar Management Tool

That is an add-in for helping with ROLAP tools also. A typical ROLAP tool will work better with a star than with any other schema, and the MetaModel is no exception. The MetaStar would use the information inside the MetaModel for generating views that would look exactly like stars for the ROLAP tool. Thus, the administrator has the advantage of a generic schema, without the users having to understand its complexity. The first tests with this concept rely on RDBMS engines that can merge the conditions on a view with the query that is running over that view. Oracle does that, so it doesn’t bring an impossible effort because of the view in the way. Other databases still don’t do that, and this tool would not be viable with them.