Enterprise Model Management Systems

David Sundaram [1] and Elke Wolf [2]

[1] University of Auckland, New Zealand, [2] Dr. Wolf Managementconsulting, Germany
d.sundaram@auckland.ac.nz, wolf@wolf-managementconsulting.de

Abstract. Decision making in the managerial context is often unstructured, ad-hoc, and complex. It involves structuring and solving the problem, and interpreting the results. A computer-based environment that supports these decision making processes is characterised by a modelling approach that enables the representation of the complexity of the problem, access to data and solvers, persistence of problem representation, and the ability to manipulate and integrate problem representations. Lack of a general framework for conceptual modelling and constraints imposed by technology have, in the past, limited the design and implementation of enterprise model management systems (EMMS). This research attempts to overcome key problems in EMMS design and implementation by proposing, implementing, and evaluating a modelling framework that uses Structured Modelling to describe models, object-oriented and relational constructs to represent models, and predicate calculus-based extended Structured Query Language (ESQL) to manipulate models.

Keywords: Decision making, Model Management, Structured Modelling, Object-Relational, Enterprise Model Management Systems.

1 Introduction

Models are an important organisational resource that attempt to capture reality, expert knowledge, and salient features of complex systems [1]. Most of the research effort in the area of organisational decision modelling in the past has concentrated on the discovery and implementation of efficient solutions to mathematical models with little attention given to the management of the modelling life-cycle in an organisational context [2,3]. Some of the phases involved in the modelling life-cycle are the formulation and use of the model, analysis of the results of the model, reformulation of the model if necessary, storage and retrieval of the model, use of the model in conjunction with other models, use the model along with other models as building blocks in the creation of larger, more complex models and finally termination of the model [4,5,6]. Most modelling systems support only some phases of the modelling life-cycle [7,8].

The above problem leads to a variety of systems with different data format requirements, different syntax, etc. being linked together to achieve modelling life-cycle support. Such patchwork systems result in perennial interface problems as well as the need for the end-user to know and keep up to date with the different syntax that

each of the components use. As there is no single common language that could be used throughout the modelling life-cycle, there is a need for a model paradigm-neutral, solver-neutral language for model definition and manipulation. There is also a need for an enterprise model management system (EMMS) where all the activities involved in modelling in an enterprise context are carried out within one environment overcoming problems of data-model and solver-model impedance mismatches.

Enterprise modelling has been a one-shot low productivity exercise, once built and used; the models were rarely used again since the conditions/premises on which the models were built had changed in time [9]. Many modelling systems support the situation where the same model structure is instantiated with different data. But most systems do not allow the easy creation of versions of models where the structure of the model changes. Hence, there is very little support for the evolution of models. In most systems there is re-use of the solver but very little re-use of model or the data used to instantiate the model.

Modellers use multiple representations for developing a model, 'natural' representation for decision makers, 'mathematical' representation for analytical use and 'computer-executable' representation for computers. This leads to redundancy, inconsistency, demand of different skills and hence low productivity [10]. This also leads to decision and policy makers finding it difficult to understand the models built, build models on their own, use the models and modify existing models.

Lack of standardised easy to use interfaces between models and data and between models and solvers lead to time consuming tasks of specially preparing the data to suit the specific model and solver. This again requires specialised skills [9,10] preventing average decision-makers from using the system. Most modelling systems are either very domain specific (like Production Planning domain or Financial Planning domain) or implement only one modelling paradigm (like LP or Simulation). Most modelling systems lack robust model integration support [11,12]. They have some form of consolidating results from different models, but most do not support the integration of models belonging to different model classes, or models belonging to different modelling paradigms, or deep integration where intermediate results of one model affect the selection or execution of another model. Most modelling systems also lack the capability to integrate simple model structures to form complex model structures [13].

2 The SM-OR-SQL Framework

This research attempts to advance core knowledge in the Information Systems (IS) and Management Science/Operations Research (MS/OR) disciplines through the demonstration and testing of ideas and concepts that have been proposed in the area of enterprise model management. The research tries to overcome the problems identified above by proposing, implementing, and evaluating a modelling framework (SM-OR-SQL framework) that uses Structured Modelling to describe models, Object-oriented and Relational constructs to represent models, and extended Structured Query Language (SQL) to manipulate models (Fig 1).

SM was selected as the paradigm to describe models due to its generality and applicability across domains as well as its robust mathematical base. SM's description clearly delineates between models, solvers, and data, which enable model-data independence and model-solver independence [14]. SM's description is also purpose independent, thus allowing the decision maker to use the model for a variety of purposes. The hierarchical nature of SM's modular tree allows decision makers to view the problem at different levels of abstraction, leading to a better understanding of the problem.

OR-DBMS are used to represent the structured models. Using OR-DBMS functionality we are able to naturally represent the SM modular structure made up of modules, primitive entities, compound entities, attributes, tests, and functions. OR-DBMS functionalities also allow us to realise the required characteristics of EMMS. The OR-DBMS also meets the three design challenges posed by Geoffrion [12], namely, 1) it provides an environment that accommodates and allows the implementation of a general conceptual modelling framework such as SM; 2) it provides a modelling language that can implement most of the key functionalities that are required in an EMMS; and 3) it provides an environment where data, models, and solvers could be integrated and managed with ease. The object-relational database management environment provides object-oriented and relational constructs and functionality that allows us to better represent and manage the complexity of the real world [15,16,17].

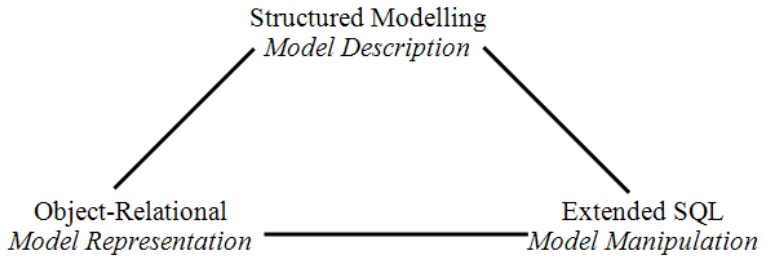

Fig. 1. The SM-OR-SQL Framework

The framework supports the modelling process through the following steps: a) the model is described using SM concepts, constructs, and guidelines b) the structured model is represented using object-oriented and relational constructs c) the Object-Relational representation of the structured model is implemented using an OR-DBMS. d) ESQL is used in the implementation/definition, manipulation, and maintenance of the Object-Relational representation of the structured model. The components of the SM-OR-SQL framework are described in detail in the following sections.

3 The Structured Modelling (SM) Paradigm

The SM paradigm suggested by Geoffrion uses a hierarchically organised, partitioned, and attributed acyclic graph to represent models. SM has three levels: elemental

structure, generic structure, and modular structure. Elemental structure aims at capturing all the definitional detail of a specific model instance. Generic structure aims to capture the natural familial groupings of elements. Modular structure aims to organise generic structure hierarchically according to commonality or semantic relatedness. This enables the complexity of a model to be managed in terms of higher order abstractions.

Every structured model can be represented in terms of five element types: primitive entity, compound entity, attribute, function, and test. A primitive entity is existential in nature and represents a primitive definition concerning a distinctly identifiable entity such as a person, place, thing, action, concept, event, state, etc. A compound entity references other entities (primitive or compound) already defined. Attributes associate a property and a value with an entity or collection of entities. Function elements resemble mathematical equations; they have a value which depends on a rule and the values of called elements. The test element is like a function element except that its value can only be boolean (True/False).

Each element has a calling sequence that identifies other elements directly referenced. The elements can be partitioned into genera based on generic similarity. Every element in a genus must have the same number of calling sequence segments. Genera could be further aggregated into modules in a way that is conceptually meaningful. Modules aggregated together on a similar principle give rise to the modular structure. The modular structure provides a very useful mechanism that allows the user to view the model at different levels of abstraction. This structure could be arbitrary [18] as long as there are no forward references (monotone ordering). Figure 2 shows a generic modular structure. Module names begin with an ampersand (&) to differentiate them from genus.

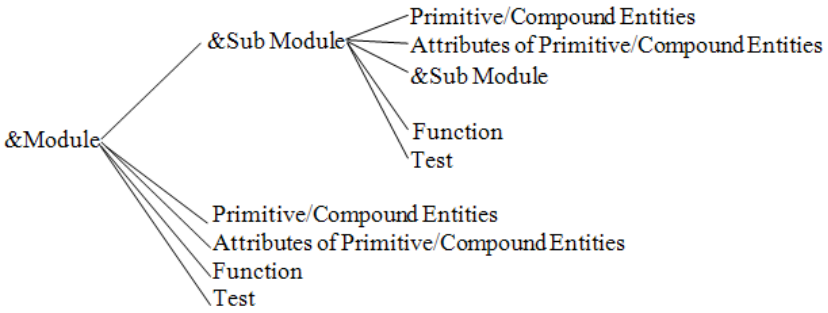

Fig. 2. A Generic Modular Structure

A model schema is used as a tool to formalise the structure of a class of 'similar' structured models as illustrated in Figure 3. A model schema corresponds with the details provided in a modular structure, generic structure, and the intended meaning of structured models. Every line/leaf of the modular structure when expanded with specific details of the model gives rise to the model schema.

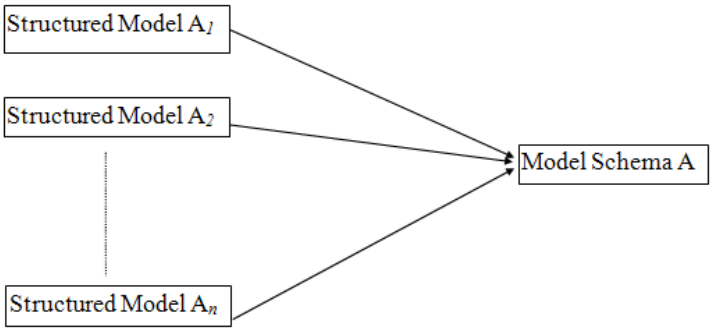

Fig. 3. Similar Structured Models can be formalised into a Model Schema

The details that are given in a model schema for every genus are:

- the genus name
- index, if any, for the genus
- all elements except for primitive entities have a generic calling sequence that identifies all the elements which participate in the definition of the genus, three kinds of calls can be identified in SM [19]:
  - *attribute call* - an attribute element calling a primitive element
  - *association call* - there are two cases of this type of call (a) an attribute element calling two or more primitive elements and (b) a compound entity element calling two or more primitive elements
  - *parameter call* - a function or test calls attribute elements which are used in the calculation
- genus type for attributes (real, integer, etc.)
- index set statement specifying the population of the genus
- range statement defining permissible value for attributes
- a generic rule specifying how values are to be determined for a function or a test genus

We illustrate these concepts by using Geffrion's [20] feedmix example in Figure 4. The element type is indicated as /pe/ (primitive entity), /ce/ (compound entity), /a/ (attribute), /va/ (variable attribute), /f/ (function), and /t/ (test). The genus name in a non-primitive genus is always followed by a generic calling sequence in parenthesis (e.g. (MATERIALm), (NUTRi, MATERIALm)). The type indicator is followed by the index statement in brackets (e.g. {MATERIAL}, {NUTR}). Omission of the index statement implies that every possible element exists. The index statement of a function or test genus is always followed by a semicolon (;) and then a generic rule which specifies how the element values are calculated (e.g. ;NLEVELi> =MINi , ; SUMm(UCOSTm*Qm)). Detailed syntax and semantics of structured models can be found in Geoffrion [20]. The main module is called &FEEDMIX and is made up of sub-modules &NUT_DATA and &MATERIALS and four genera: Q - attribute genus, NLEVEL - function genus, T:NLEVEL - test genus and TOTCOST - function genus.

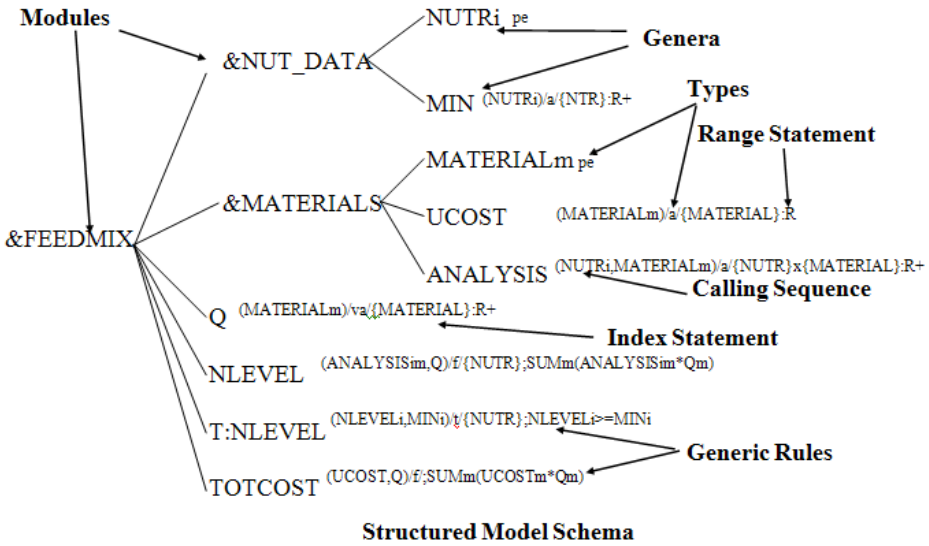

Fig. 4. Modular Structure Schema of the Feedmix problem (Adapted from [20])

Each of the sub-modules are composed of genera that have similar calling sequences. That is the elements that participate in the definition of the genera should be the same or similar so that they could be interpreted as belonging to the same group. This leads to several plausible hierarchical organisations of the modular tree. The &NUT_DATA sub-module is composed of two genera:

- NUTR - primitive entity and
- MIN - attribute genus

The &MATERIALS sub-module is made up of three genera:

- MATERIAL - primitive entity
- UCOST - attribute genus and
- ANALYSIS - attribute genus

Note that ANALYSIS could have been included as part of the &NUT_DATA since its calling sequence has NUTR. Every genus should belong to a type. As can be seen, some are primitive entities, some are attribute genera, one belongs to the function genus, and another belongs to the test genus. All genera apart from primitive entities have a calling sequence. The seven non-primitive entity genera have the following calling sequences:

Genus	*Calling Sequence*
MIN	NUTR
UCOST	MATERIAL
ANALYSIS	NUTR and MATERIAL
Q	MATERIAL
NLEVEL	ANALYSIS and Q
T:NLEVEL	NLEVEL and MIN
TOTCOST	UCOST and Q

The index statement specifies the element population of the genus. For example, Q is indexed on MATERIAL indicating that it has one element for every MATERIAL element. Each attribute genus has a range that specifies the allowable values for the elements of the genus. For example, the attribute genus ANALYSIS can have only positive real values indicated by R+. Every function or test genus has a generic rule that specifies how the element is calculated. For example, the function genus TOTCOST is calculated by the summation of the product of UCOST and Q over all the MATERIAL elements.

We use the modular structure with all the details of every leaf (that is the model schema details) as the structure that will be represented using Object-Relational constructs. Apart from using these constructs we use SM guidelines [4] in the entire model development process. These guidelines include such aspects as a) incorporation of data development processes directly into the model b) documentation of the definitional interdependencies c) the use of stepwise refinement and d) creation of large complex models from validated submodels and e) exploitation of parallel structure.

4 The Object-Relational Framework

The Object-Relational framework is built around the properties and functionalities that the Object-Relational DBMS offer. This section first looks at the key features offered by OR-DBMS and then goes into detail about the features of the Object-Relational framework that we will be using to represent structured models and realise EMMS functionalities.

An OR-DBMS is a DBMS that provides all the services and functionality of traditional relational DBMS such as:

- flexible data access through a standard SQL
- security controls
- server-enforced data integrity
- transaction and recovery
- performance and scalability

as well as key object-oriented features such as:

- open architecture/extensibility - not limited to the predefined built-in data types, and a system that could be tailored and extended to suit requirements
- facility to create user-defined/abstract data types
- facility to create user-defined functions/methods to act on the user-defined/abstract data types as well as existing/built-in data types
- single and multiple inheritance of both structure and behaviour with option for late binding
- polymorphism

as well as features that neither relational nor object-oriented systems possess such as:

- functions/methods that span types
- extensions to the industry standard familiar SQL to support objects

Thus in some ways OR-DBMSs attempt to have the best of both worlds. This is reflected in Stonebraker's [21] classification of DBMS applications illustrated in

Figure 5. Some of the features of OR-DBMS discussed above are used to build a framework that allows us to capture the essence of structured models naturally and to realise EMMS functionality. These features form the core of the Object-Relational component of the SM-OR-SQL framework, and they are discussed in detail in the following sections.

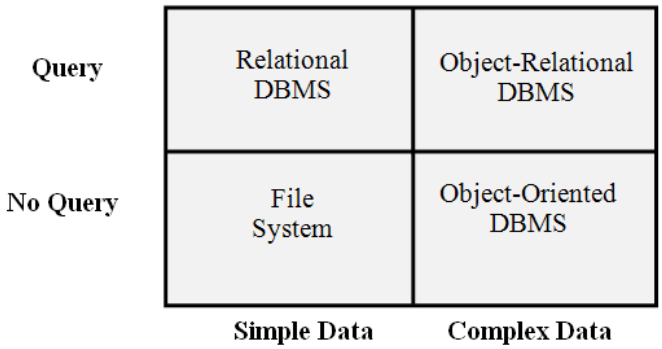

Fig. 5. Classification of DBMS Applications (Source: [21])

Kim [22] has proposed an Object-Relational data model that formalises some of the ideas regarding OR-DBMS. He describes the object-oriented and other extensions to the relational model that results in the Object-Relational data model. We first describe briefly the relational model and then show the extensions to the same. The relational database consists of a set of relations (types). A relation consists of rows (instance of a type) and columns (attributes). A column entry in a row of a relation can have a single value that can belong to a set of system-defined data types (integer, float, string, date, boolean, etc.). There are four key extensions to the relational data model that results in the Object-Relational model:

1. The value of the column of a relation can be a row of any arbitrary user-defined relation, rather than being restricted to the system-defined data types. This implies that the user can specify an arbitrary user-defined relation as the domain of a relation. This leads to nested relations where the value of a row/column entry of a relation can now be a row of another relation.
2. Procedures/methods can be attached to relations and the procedures can operate on the column values in each row. Thus the relation encapsulates the state and behaviour of its rows.
3. Relations can be organised into a hierarchy as a directed acyclic graph such that between a pair of relations P and C, P is made the parent of C, if C is to inherit all columns and procedures defined in P, besides those defined for C. This allows reuse of columns and procedures.
4. The row/column entry of a relation can have a set of values of any data type - system defined or user-defined. This is in contrast to the relational model which allows each row/column entry to have only one value. This reduces the duplication of data as well the creation of a structures that are semantically more meaningful than the traditional data models.

Though these extensions may appear minor individually, collectively the impact on the ease of modelling complex domains is significant. In the following paragraphs we

describe in detail concepts and functionality of OR-DBMS that help us to represent structured models and realise EMMS.

5 Abstraction levels and associated languages

There are four levels that can be identified in the Object-Relational framework in the context of the Illustra OR-DBMS. Each of these levels has a counterpart in the modelling dimension as well as a language associated with it. The model abstraction levels, the equivalent Object-Relational abstraction levels, and the associated language support are shown in tabular form in Table 1.

The first model abstraction level Model Definition and Manipulation is supported through Types, Rules, and Functions by the object-relational abstraction level. This abstraction level is supported through the Extended Structured Query Language (ESQL) which is compatible with SQL3 standard and the procedural language C. The second model abstraction level Model Instance Storage Definition is supported through Tables and Rules by the object-relational abstraction level. This abstraction level is supported purely through the declarative ESQL. The third model abstraction level Model Instantiation and Manipulation is supported through the Instance Level of the object-relational abstraction level. Model instances are created and manipulated through ESQL. All the above-mentioned objects can be manipulated and combined to form complex modelling scenarios via the fourth model abstraction level (Meta-Modelling). This level of abstraction is supported through an icon-based visual modelling environment. The visual modelling environment is based on the box and arrow paradigm. The user connects boxes, which represent objects (like containment hierarchies/composite types, constructed types, base types, and the instantiating data) and methods, using arrows to indicate program control flow.

Table 1. Abstraction Levels and associated languages

Model Abstraction Level	Object-Relational Abstraction Level	Associated Language Support
Model Definition and Manipulation	Types, Rules, and Functions Level	Declarative and Procedural (ESQL and C)
Model Instance Storage Definition	Tables and Rules Level	Declarative (ESQL)
Model Instantiation and Manipulation	Instance Level	Declarative (ESQL)
Meta-Modelling/ complex Modelling Scenarios/Model Integration	Iconic Level	Visual Modelling Environment / *Box and Arrow* paradigm

6 Representation Scheme for Object-Relational Models

We adopt a scheme for representing Structured Model schema using Object-Relational constructs that clearly delineates between the structure and behaviour of

models. Every module/sub-module of the SM schema is shown as a rectangle. This module/sub-module will have structure and behaviour. The rectangle is divided into three regions - the first part shows the name of the module/sub-module, the second shows the structure of the module/sub-module, and the third shows the behaviour of the module/sub-module. Relationships between modules and sub-modules are shown using an arrow with the term Aggregation printed above the arrow and the cardinality shown at the arrowhead. Relationships between modules/sub-modules that inherit structure and/or behaviour from another module/sub-module are shown using a directed arrow with the arrowhead pointing to the module from which the structure and/or behaviour are being inherited and the term Inheritance printed above the arrow. This scheme of representing SM schema using Object-Relational constructs is illustrated in Figure 6.

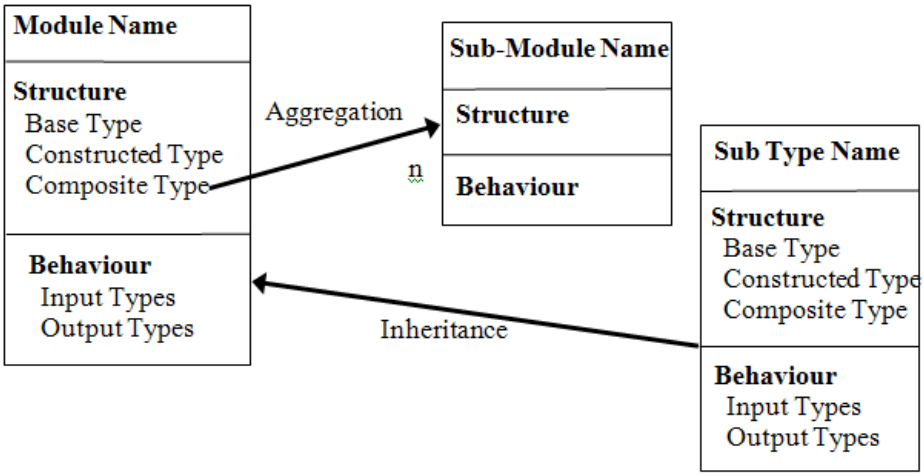

Fig. 6. Representation Scheme for Object-Relational Models

7 Translation of Structured Models into equivalent Object-Relational representations

The SM model schema that we looked at earlier can be thought of as having structure and behaviour. Structure is defined using primitive and compound entities and their attributes. Behaviour is defined by tests and functions. In the Object-Relational framework we have composite types, constructed types, and base types to define the structure of any model and methods/functions to describe the behaviour of any model. In the following paragraphs we look at this translation in detail.

The modular structure along with the model schema of SM can directly be translated into an equivalent Object-Relational representation. The modular structure is equivalent to the containment hierarchy that was discussed earlier. Just as the containment hierarchy uses the concept of relation to aggregate all related entities, the modular structure organises all semantically related genera hierarchically. A module can be viewed as an aggregation of submodules, primitive/compound entities and

their attributes, and functions and tests that operate on the submodules and/or primitive/compound entities. Thus a containment hierarchy that has composite types, constructed types, base types, and functions/methods that operate on the composite and/or base types is equivalent to a modular structure made of submodules, primitive/compound entities and their attributes, and functions and tests that operate on the submodules and/or primitive/compound entities. Figure 7 shows a containment hierarchy that translates the generic modular structure of Figure 2.

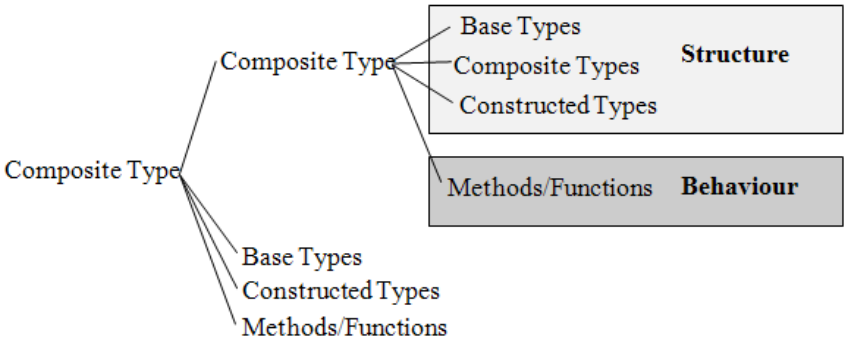

Fig. 7. Containment Hierarchy of the Generic Modular Structure of Figure 2

Primitive entities/Compound entities with their attributes map into composite types. Functions and Tests map into Methods/Functions. Sub Modules and Modules map into composite types. This mapping is illustrated in Table 2.

Table 2. Equivalence between the SM paradigm and the Object-Relational Framework (Source: [23])

Component	SM Paradigm	Object-Relational Framework
Structure	Primitive Entity and its attributes	Composite Type made up of base types
	Compound Entity and its attributes	Composite type made up of other composite types, constructed types, and base types
	Modules	Composite type made up of other composite types, constructed types, and base types
	Calling Sequence - Attribute and Association Calls	Inherent in the type structure and can be deduced by the components of the composite type that represents the primitive or compound entity
	Calling Sequence - Parameter Calls	Inherent in the definition of the function and can be obtained by looking at the input parameters of the function
Behaviour	Functions	ESQL and C Methods/Functions
	Tests	ESQL and C Methods/Functions
	Index	Achieved through primary keys and rules.
	Range	Achieved through table definitions and rules

The different types of sequence calls are implemented in the Object-Relational framework in the following way:

- since attribute calls originate from a primitive entity the attribute call equals the attribute of a type
- association calls by attribute elements are mapped by a composite type whose foreign keys match the primary keys of the referenced types (called primitive entities)
- association calls by compound entities are mapped by a composite type made up of base types (called primitive entities)
- parameter calls by functions or tests are mapped by the input and output types defined by a function

Thus we can map modular structure along with its detailed schema through the definition of containment hierarchies that are made up of base, composite, and constructed types and functions that operate on them.

8 Implementation

The SM-OR-SQL modelling framework was successfully implemented using Illustra, the commercial version of POSTGRES, an Object-Relational database management system prototype developed at the University of California, Berkeley. Illustra's three layers, namely, data management, object management, and knowledge management, provide the basis for dealing with the complexities associated with model representation, model manipulation, ad-hoc modelling, sensitivity analysis, integration of models, etc. While we have chosen commercial products for stability reasons similar functionality is available in open source platforms. For instance PostgreSQL (available at www.postgresql.org) provides excellent support for defining from simple to complex data, models, and solvers [24].

A Hierarchical Manufacturing Planning System made up of three planning levels, namely, forecasting, aggregate production planning (APP), and disaggregate production planning (DPP), was implemented to highlight the characteristics of the EMMS (Figure 8). This real world scenario was chosen as it is rich with models belonging to different paradigms as well as exhibiting different types and levels of integration. Using this implementation as a base we explored and implemented key EMMS functionality such as:

- provision of model-data, model-solver, and model-purpose independence
- development of models belonging to different modelling paradigms such as Forecasting, Linear Programming, etc.
- development of an environment where
  - ⇒ models could be created, instantiated, executed, shared, and terminated
  - ⇒ model instance, structure, and behaviour could be modified
  - ⇒ models could be aggregated, pipelined, or spliced
  - ⇒ models could be integrated in a permanent as well as an ad hoc manner
- provision of modelling languages to support most facets of the modelling life cycle
- implementation and interfacing of model and solver hierarchies

- organisation and management of models
- making model objects (models, data, solvers, parts of models) available for use and reuse in the creation and execution of other models
- provision of What-if /Sensitivity analysis features

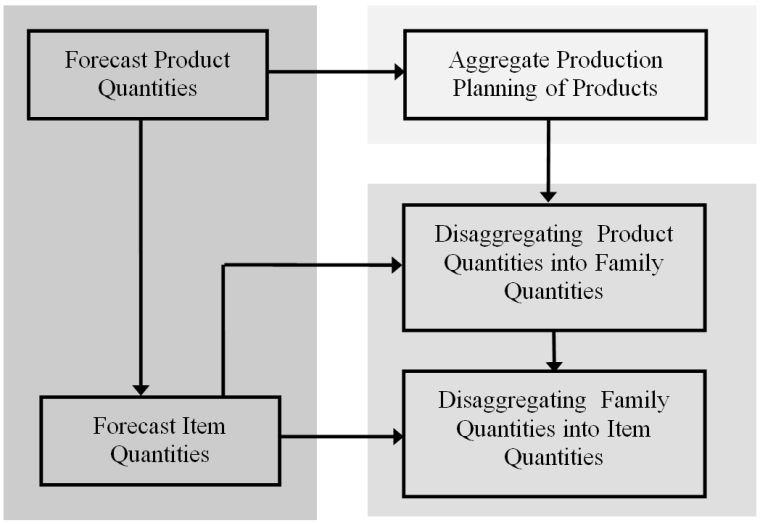

Fig. 8. Hierarchical Manufacturing Planning

Some of the key Structured Modelling concepts that underpin the prototype are:

- clear delineation between model structure (primitive entities, compound entities, and their attributes) and model behaviour (functions and tests)
- clear delineation between models, solvers, and data
- model-data, model-solver, and model-purpose independence inherent in the SM description of the model
- model abstraction through the use of modules made up of sub-modules which are themselves made up of generically related entities

Some of the key constructs of the Object-Relational layer of the SM-OR-SQL framework that helped in representing structured models are:

- definition of simple user-defined objects as well as complex user-defined objects made from simpler objects
- associating behaviour/function to objects
- creation of aggregation hierarchies from simple and complex objects
- creation of inheritance hierarchies

Key features of the ESQL associated with Illustra that helped in delivering EMMS functionality were:

- model, solver, and data definition and manipulation facilities
- extensibility of ESQL through user-defined functions and operators
- flexibility of associating models, solvers, and data in an ad hoc manner
- support for integration of models through nesting of ESQL statements

9 Conclusion

We conclude that the Object-Relational database management system is a flexible and versatile host for the implementation of EMMS. The evaluation of the prototype indicates that key functionalities of EMMS were realised. The definition and management of models, solvers, and data using a single modelling language (ESQL) within one environment (Illustra) seemed to be one of the key factors in the successful implementation of the EMMS. We were able to explore a wide variety of issues because we built the EMMS on existing components such as the versatile Object-Relational DBMS Illustra and public domain solvers, instead of building the EMMS ex nihilo.

The Structured Modelling paradigm was capable of describing the Forecasting, APP, and DPP models with ease. The Object-Relational framework with its rich constructs was able to faithfully capture the complexities of the Forecasting, APP, and DPP structured models. We were able to use ESQL as the modelling language to define the above models, manage the process of modelling, as well as integrate the models in a variety of ways. The SM-OR-SQL framework thus provides a means of describing, representing, and managing enterprise models in a productive manner.

References

1. Iyer, B., Shankaranarayanan, S., Lenard, M.L.: Model management decision environment: a Web service prototype for spreadsheet models, Decision Support Systems v. 40, n.2, pp. 283--304 (2005)
2. Krishnan, R., Chari, K.: Model Management: Survey, Future Research Directions and a Bibliography, The Interactive Transactions of OR/MS, v. 3, n. 1 (2000)
3. Bharadwaj, A., Choobineh, J., A. Lo, A., Shetty, B.:Model Management Systems: A Survey", Annals of Operations Research, v.38, pp.7--67 (1992).
4. Liew, A., Sundaram, D.: Flexible modelling and support of interrelated decisions, Decision Support Systems, v. 46, n. 4, pp. 786--802 (2009)
5. Eriksson, D.M.: A Framework for the Constitution of Modelling Processes: A Proposition, European Journal of Operational Research, v.145,n.1, pp. 202--215 (2003)
6. Wright, G. P., Chaturvedi, A. R., Mookerjee, R.V., Garrod, S.: Integrated Modeling Environments in Organizations: An Empirical Study, Information Systems Research, v.9, n.1, pp. 64--84 (1998)
7. Power, D.J., Sharda, R.: Model-Driven Decision Support Systems: Concepts and Research Directions, Decision Support Systems, v. 43, n. 3, pp. 1044--1061 (2007)
8. Gass S.: Managing the Modelling Process, European Journal of Operational Research, May (1987)
9. Dolk, D. R.: A Generalised Model Management System for Mathematical Programming, ACM Transactions on Mathematical Software, v.12, n.2, pp. 92--126 (1986)
10. Geoffrion, A.: An Introduction to Structured Modeling, Management Science, v.33, n.5, pp.547--588 (1987)
11. Kottemann, J. E., Dolk, D. R.: Model Integration and Modeling Languages: A Process Perspective, Information Systems Research, v.3, n.1, pp. 1--16 (1992)
12. Geoffrion, A.: Integrated Modeling Systems, Computer Science in Economics and Management, v.2, n.1, pp.3--15 (1989)

13. Makowski, M.: A structured modeling technology, European Journal of Operational Research v.166, pp. 615—648 (2005)
14. Lee, K.-W. Huh, S.-Y.: A Model-Solver Integration Framework for Autonomous and Intelligent Model Solution, Decision Support Systems, v.46, n.2, pp. 926--944 (2006)
15. Biswas, S., Narahari, Y., Object Oriented Modeling and Decision Support for Supply Chains, European Journal of Operational Research, v.153, n.3, pp. 704--726 (2004)
16. Liu, D., Stewart, T.J., Integrated Object-Oriented Framework for MCDM and DSS Modelling. Decision Support Systems, v. 38, n.3, pp. 421--434 (2004)
17. Dolk, D.R.: Integrated model management in the data warehouse era, European Journal of Operational Research, v. 122, pp.199--218 (2000)
18. Geoffrion, A.: FW/SM: A Prototype Structured Modeling Environment, Management Science, v.37, n.12, pp.1513--1538 (1991)
19. Lee, K.: Integrated Information Management Architecture: An Object-Oriented Approach for Problem Representation, Model Formulation, and Model Execution, Unpublished Ph.D. Dissertation, University of Wisconsin – Madison. (1992)
20. Geoffrion, A.: An Introduction to Structured Modeling, Management Science, v.33, n.5, pp.547--588 (1987)
21. Stonebraker, M., with D. Moore: Object-Relational DBMSs: The Next Great Wave, Morgan Kaufmann Publishers (1996)
22. Kim, W.: Object-Relational Database Technology, A UniSQL Whitepaper available at http://www.unisql.com/technology/papers/kim /whtpaper.html (1996)
23. Srinivasan, A., Sundaram, D.: 'An Object Relational approach for the design of Decision Support Systems', European Journal of Operations Research, 127(3): 128--144 (2000)
24. Douglas, K.: PostgreSQL, Second Edition, Sams (2005)