MODEL DRIVEN ENGINEERING : TWO APPROACHES THROUGH THE SAME CASE STUDY

V. Ribaud, P. Saliou, M. Kerboeuf

EA3883, LISyC, Université de Bretagne Occidentale, C.S. 93837 29238 Brest Cedex 3, France
E-mail: {Mickael.Kerboeuf, Vincent.Ribaud, Philippe.Saliou} @univ-brest.fr

Abstract

The work described in this article presents two model-driven engineering approaches through the same case study. The case study is an shared electronic agenda used as a representative from medium and small sized Information Systems. The first approach relies on the Unified Process, supplied with IBM/Rational Rose. The second approach relies on CADM (CASE Application Development Method), a waterfall process belonging to the family of systemic methods, supplied with Oracle CASE Designer.

1 Introduction

Methods and tools are essential for the achievement of a project. Formerly, the CASE tools are often confined to the analysis and design phases while the programming environments are used for the implementation. Presently, tools vendors promote tool suites which should be able to support an integrated development process driven through a model driven approach. We present a case study, built into two different methods and environments : CADM/Designer and UP/Rose.

1.1 The development cycle

Approaches presentation follows the development cycle that especially defines role and progress of project phases. 7 phases are hold :

0 - Project set-up from a statement of work

1 - Requirement capture

2 - Analysis

3 - Design

4 - Coding and unit testing

5 - Integration and integration testing

6 - Validation

Phases 0 and 6 relies on common documents for both approaches : a statement of work is provided to define the case study and a validation plan is used to evaluate software at the end of the development process to ensure compliance with the statement of work.

1.2 Case study : statement of work (excerpts)

Each user freely creates his/her artifact as a person in the system. Agenda let users to create meetings, to register/cancel for a meeting and to consult existing meetings along several criteria : participant, meeting room, subject, temporal interval.

...

A meeting is attended by one to many persons and is located in an unique room that must be available at meeting time. A room can successively hold up different meetings. A person can take part to several disjoint meetings.

...

Some operations are right-restricted. Generally, an user can perform any update on objects that he/she created. An administrator is provided with the agenda in order to manage common data ; management right is transmissible and irrevocable.

...

1.3 Validation plan

The statement of work is not structured in order to have no influence on the approach used. It does not contain detailed requirements but rather requirements titles, presented within a numbered list ordered alphabetically. This list order indexes the verification plan too; this presents no sense with the real verification plan issued from each approach, but should permit the comparison of compliances through the indexed list.

N.	Requirement	N.	Requirement
1	Cancel participation to a meeting	10	Participate in a meeting
2	Create a meeting	11	Retrieve meetings by attendee
3	Create a person	12	Retrieve meetings by date
4	Create a room	13	Retrieve meetings by meeting room
5	Connect [Register]	14	Retrieve meetings by subject
6	Delete a meeting	15	Update a meeting
7	Delete a person	16	Update a person
8	Delete a room	17	Update a meeting
9	Deregister		

2 First approach : the Unified Process with Rational Rose

The development process belongs to the unified processes family. « The unified process is first and foremost a software development process ... The unified process uses the Unified Modelling Language (UML) in order to create elaboration and building plans of the software system ... Nevertheless, the truly specific features of the unified process are as follows: use case driven, architecture-centered, iterative and incremental » [5].

2.1 Requirement capture

Requirement key point is to establish the importance of requirements as a contract between client and provider. A requirement is defined from IEEE Std 729-1983 « Condition or capacity which a system or subsystem must exhibit to satisfy a contract, standard, specification, or other obligatory formal document ».

Use case model is employed. A use case captures a contract between the stakeholders of a system about its behaviour. The use case as a contract for behaviour relies on two models described by Alistair Cockburn [2] : -1- the system provides interactions between actors with goals, -2- the system has the responsibility to protect the interests of all the stakeholders.

2.2 Analysis

The unified process presents analysis as a transition between an external view of the system (written in the client's language) to an internal view (written in the developer's language). The analysis structure is as follows : package, classifier (class, association, interface), element.

The goal is to get a static analysis of the system (in the form of a class and/or an object diagram) and a dynamic analysis (in the form of collaboration[1] diagrams : one for each use case). Each analysis class belongs to one of the base stereotypes : « boundary », « entity », « control ». The analysis class diagram is structured (through packages) in sub-diagrams, overlaying several use cases. Each use case is a collaboration into the analysis model that describes the way a given use case is realized and executed in term of analysis classes and interactions between instances of these classes. Technical requirements that are not defined with use cases (sometimes called non-functional requirements) are analysed in a common collaboration if several use cases are concerned else in a peculiar collaboration. Collaboration diagrams are structured into packages independently of class diagrams structure. Services packages can be used at a lower level in order to structure the system from the services that it provides; this is an important step in the development process because it gives the system its initial structure which is subject to further refinement in the design phase.

2.3 Design

Product design consists of two broad phases that may overlap in execution: preliminary and detailed design. Preliminary design establishes product capabilities and the product architecture, including product partitions, product-component identifications, system states and modes, major intercomponent interfaces, and external product interfaces. Detailed design fully defines the structure and capabilities of the product components [3].

The unified process presents design as a shaping activity in order to give a form and an architecture to the system that meet requirements. As a fundamental basis for design, the analysis model assigns a system structure that we should try to keep. However, the design model is an object model which is an abstract vision of the system implementation ; this model is depicted in a hierarchical system with subsystems and design classes.

There are so many design class stereotypes that the implementation language and its architecture are providing with. The architectural model used is MVC (Model-View-Controller).

---

[1] Collaboration diagrams are called communication diagrams in UML 2.0 and moved from the structural diagrams to the behaviour diagrams.

A « View » design class models the interaction of the system with the actors and often represents abstractions of forms, windows... mostly derived from « boundary » analysis classes. A « Model » design class is used to represent information and behaviour of a phenomena or a concept; there are mostly stemming from « entity » and « control » analysis classes. A « Control » design class represents coordination, scheduling, transactions and other objects control, as also complex processing that cannot be linked to a given « Model » class.

The design model establishes an obvious mapping between design artifacts and implementation constructs of used tools : « Views » are OCX controls and VB forms, « Models » are relational tables (possibly with object/relational persistence and query service), «Controllers» are SQL stored procedures or queries as also VB procedures. The specification of a design artifact uses the same language as the implementation tool ; then operations, parameters, types, ... are specified in the tools syntax.

Component diagrams are used to represent design classes and their dependencies. Interaction overview diagrams[2] are used to depict high-level control (referring to design classes rather than interaction diagrams).

2.4 Coding and unit testing

Once the design has been completed, it is implemented as a product component. The characteristics of that implementation depend on the type of product component. Then, unit testing of the product component is performed as appropriate. Unit testing involves the testing of individual software units or groups of related items prior to integration of those items.

Coding uses relational tables; an ODBC-like driver to access to data sources; SQL queries and stored procedures, VB procedures and OCX controls and forms.

An environment must be established to enable unit testing to take place. Testing units incrementally promotes early detection of problems and can result in the early removal of defects.

2.5 Integration and integration testing

The purpose of Product Integration is to assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product. A critical aspect of product integration is the management of internal and external interfaces of the products and product components to ensure compatibility among the interfaces [3].

Product integration can be conducted incrementally, using an iterative process of assembling product components, testing them, and then assembling more product components. Testing consists in verifying that a component conforms to its baseline and satisfies all specified requirements. Tests define :

- nominal input data,
- foreseen operator’s actions,
- expected results,
- functions scheduling.

[2] Novelty in UML 2.0

2.6 Software testing

The purpose of Validation is to demonstrate that a product or product component fulfils its intended use when placed in its intended environment [3].

Tests organization relies on validation plan, structured as follows :

- Validation is divided into operations.
- A validation operation consists in verifying a set of functionalities, services, documents or system constraints. An operation is structured in stages.
- A stage is decomposed in actions. Actions define functionalities to verify at each stage.
- Each action is constituted with trials. Trials should verify results conformance with requirements.

3 Second approach : the CADM method with Oracle Designer

3.1 Designer and the associated CADM method

Oracle Designer is an extremely powerful integrated CASE tool. It allows the whole building of an Information System all along the phases of the software life cycle. To this end, it relies on a unique common repository stored in an Oracle database.

The development approach relies on CADM (CASE Application Development Method), a waterfall process (Analysis->Design->Build->Implement->Production)) by Paul Dorsey and Peter Koletzke [4]. This approach is a derivative of the Case*Method by Richard Barker [1]. CADM revise and expand the Case*Method in order to use Designer. CADM does not describe how to build systems or how to use Designer, but how to build systems using Designer.

CADM belongs to the family of systemic methods. The data and processing have first to be separately modelized, and then coupled to constitute a unique and integrated system. The building of the system gets through different abstraction levels: analysis, design and implementation.

3.2 Requirements capture

There is no requirements capture model in Designer. During the preliminary analysis phase, the requirements capture materializes mostly in a textual form casually laid out or structured by a requirements plan. During the general analysis phase, the requirements capture becomes elaborate, detailed and reshaped through a function hierarchy and an Entity/Relationship data model.

The requirements capture materializes as a function hierarchy. Through this approach, the point of view is the one of the system: what are the functions that the system must offer to fulfil the final user needs ? In the use cases, the point of view is the one of the system users: what do the various users expect from the system, what are their aims ?

We wish to use a “light” use case model in order to describe this external point of view ; so we use a short textual form : use cases summaries (resumes). It is then possible to transform automatically an external point of view (use cases) into an internal point of view (function hierarchy), by relying on the organization of use cases in terms of relations as well as functional grouping in packages.

Below, an incomplete list of summaries related to meetings.

N.	Actor	Goal	Summary
1	User	Cancel participation to a meeting	A person is removed from the meeting attendees' list.
2	User	Create a meeting	A new meeting is created with its own characteristics as well as an available meeting room.
...			
6	User/ Grantee	Delete a meeting	The user/grantee deletes its own/an existing meeting. Attendees' participation to this meeting is cancelled.
...			
10	User	Participate in a meeting	A person is added to the meeting attendees' list only if he/she is available at the meeting time.
...			
17	Grantee	Update a meeting	Some characteristics of an existing room are updated. Updating the capacity requires that meetings' needs are guaranteed.

Depending on the complexity of the business area and the level of knowledge of the project team and users, it may or may not be appropriate to include an Entity-relationship Diagram (ERD) in this phase; in this case, it is called a Strategy ERD and it should identify the key entities and their relationships to provide an overall perspective of the business area data.

3.3 Analysis

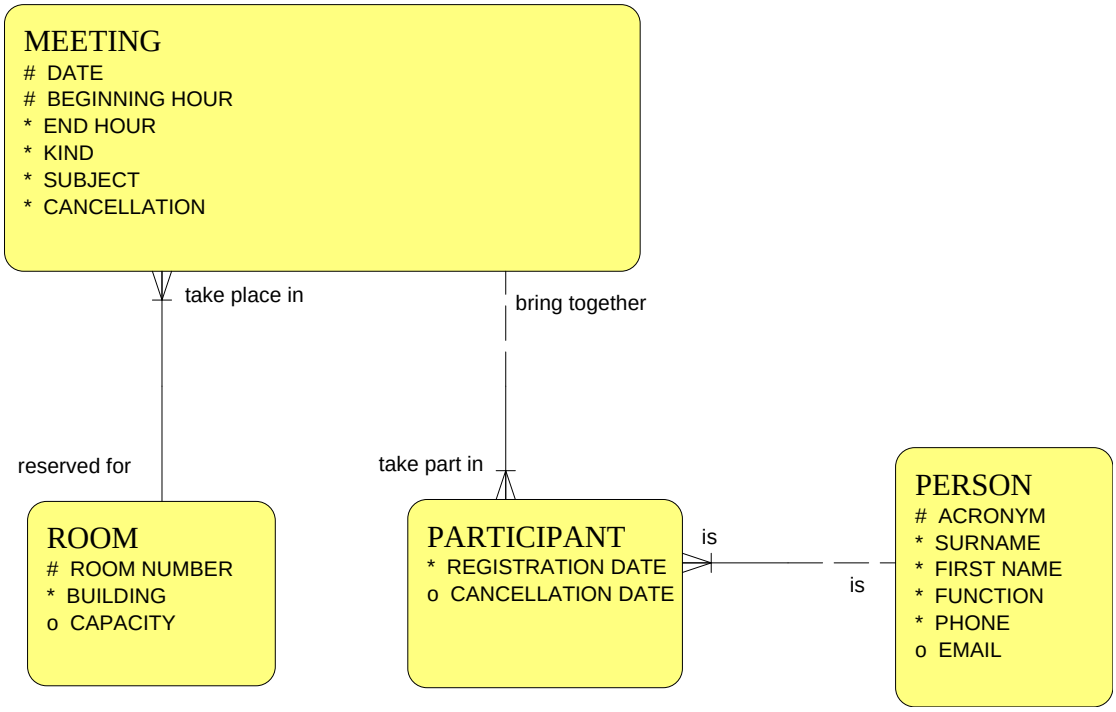
If not still defined, the Entity-relationship Diagram (ERD) should be established. It is a communication tool as well as an analysis tool. The Analysis ERD attempts to capture as many of the data-related business rules as possible in a diagram. No consideration is given to performance or to the feasibility of implementation of a rule. The goal is to represent business requirements. Rules that cannot be implemented in the ERD as stated as text.

The function hierarchy is the model proposed by Designer to analyse processing. Processing in the information system are hierarchically divided into a set of activities known as functions. Therefore a function is a more or less important activity which can be automatized or manual. Some functions can be shared, in that case they appear several times in the hierarchy with a distinctive sign.

3.3.1 General analysis

The general analysis phase produces a function hierarchy and an Entity/Relationship Data model.

The ERD model for the agenda is given below.



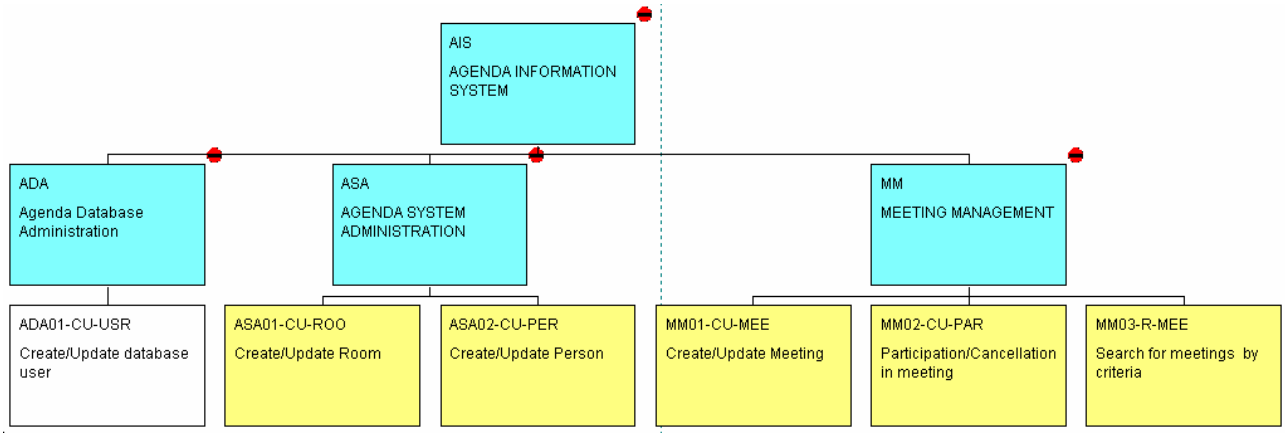
Attention should be paid to domains. A domain is a set of business validation rules, format constraints, pre-defined values that apply to a set of attributes. Domains are used to standadize characteristics of attributes and are used later by Designer generators.

For example, the kind of meetings can define a domain static (all values are defined) or dynamic (values can be added during the agenda use). The domain D_KIND_MEETING is given below

Allowable Domain Values

equenc	Value	h Va	bbreviatio	Meaning
1	GM		GM	General Meeting
2	CDMB		CDMB	Computer Department Management Board
3	DIB		DIB	Departement Improvement Board
4	DMC		DMC	Departmental Management Committee
5	OTH		OTH	Other

From the requirement list, we will get a function hierarchy given below.



Little attempt should be made at this point to identify functions that will map to application modules. The requirements capture is quite rarely structured, so it is not possible to map the

functions and entities obtained throughout this phase. Very often the function hierarchy and the E/R model are used as elements of work and discussion with the users to validate and approve requirements.

3.3.2 Detailed analysis

During the detailed analysis phase, the function hierarchy is refined and completed: entities usages are defined for every function as well as attributes usages while cross-reference controls are performed between data and functions. Let us tell more about the important feature of the definition of usages : which entities (tables) are used by functions (modules) and how functions (modules) use entities (tables) i.e. does the function (module) Create, Retrieve, Update, or Delete instances of the entity (table) ? The CRUD matrix is a two-dimensional chart that summarizes usages between functions (modules) and entities (tables). Defining usages is a part of security policy, because it defines data access control inside application modules.

As an example, the CRUD matrix for the Create/Update Meeting is given below

Entity usages

Entity	Create	Retrieve	Update	Delete
MEETING	☑	☑	☑	☑
ROOM	☐	☑	☐	☐

Meeting attribute usages

Attribute	Insert	Retrieve	Update	Nullify
BEGINNING HOUR	☑	☑	☑	☐
CANCELLATION	☑	☑	☑	☑
DATE	☑	☑	☑	☐
END HOUR	☑	☑	☑	☐
KIND	☑	☑	☑	☐
SUBJECT	☑	☑	☑	☐

Room attribute usages

Attribute	Insert	Retrieve	Update	Nullify
BUILDING	☐	☑	☐	☐
CAPACITY	☐	☑	☐	☐
ROOM NUMBER	☐	☑	☐	☐

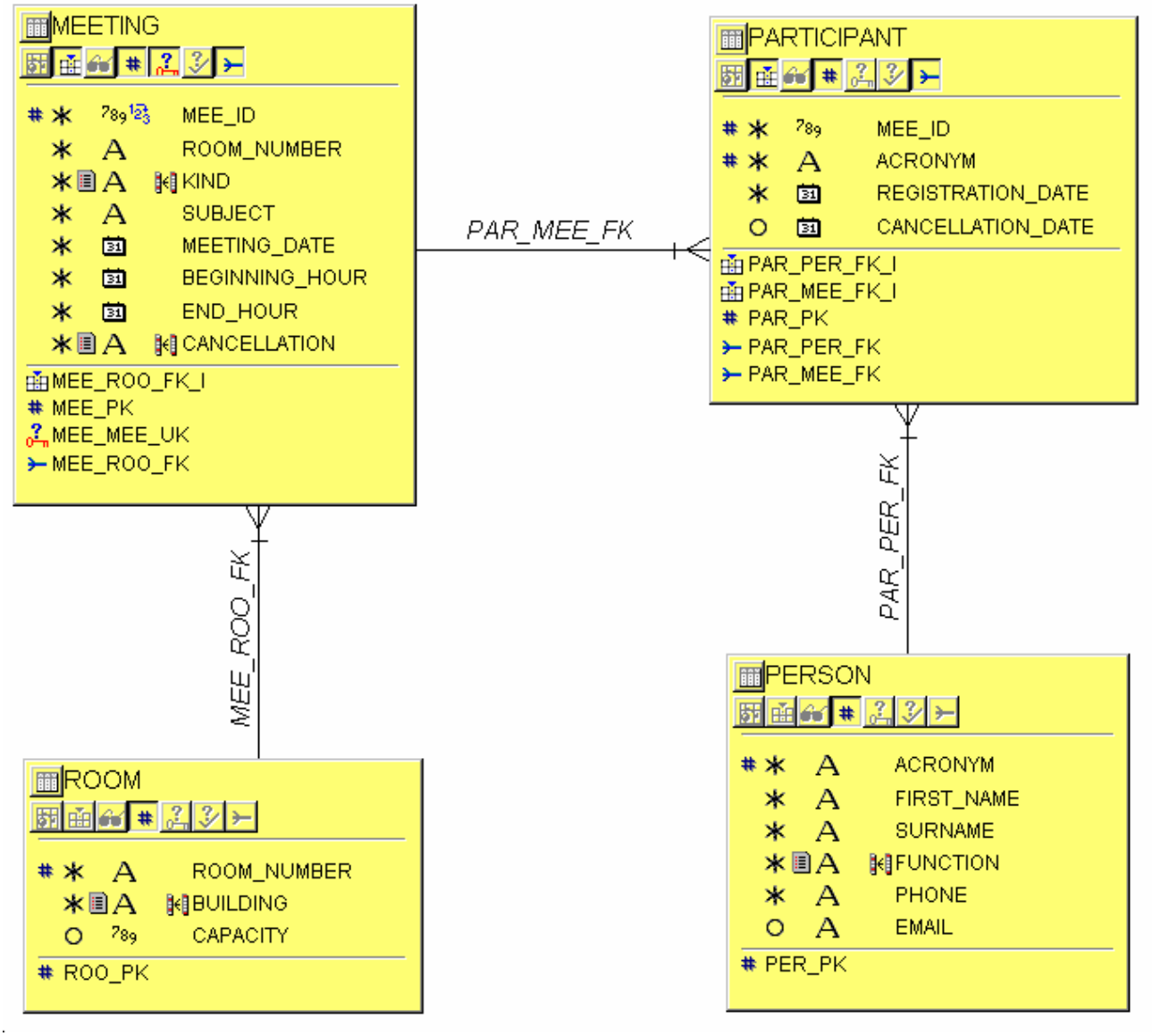
3.4 Design

3.4.1 Model transformation

At the beginning of the design phase, functions are mapped to the application modules and the E/R model is mapped to a relational model. A module is structured into modules components, which can be *in fine* translated either Oracle 4GL constructs (Forms block), either in a package of Java classes, or a set of Web pages. In the Pre-Design phase, the various design standards, including GUI standards, coding standards, and design naming conventions are determined along with the ways in which Designer will support these standards.

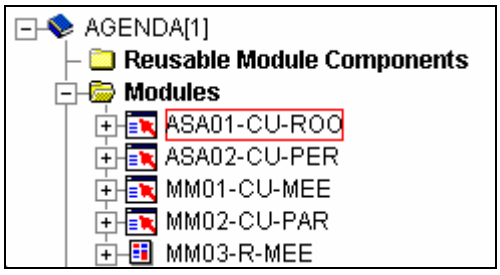
3.4.2 Server Model Diagram

Server Model diagrams present a picture of various physical structures in the Oracle database (e.g., tables, views). The logical relational model for the agenda is given below.



3.4.3 Application modules

For each atomic function we obtain an application module.



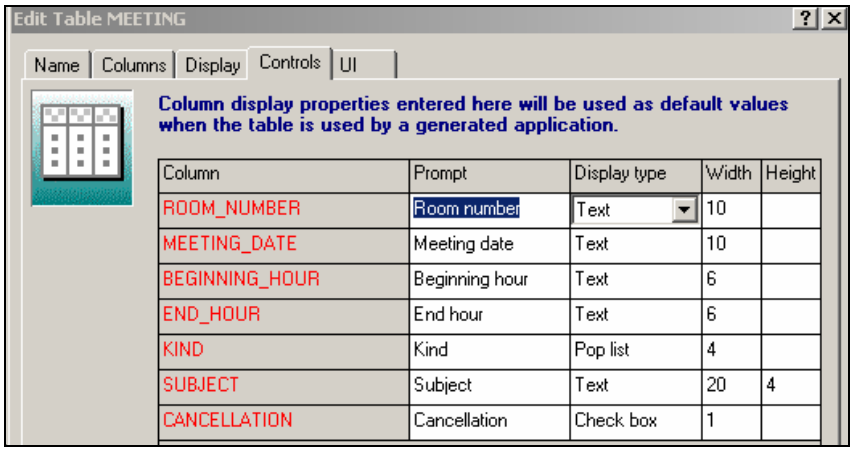
There is a traceability link between design modules and analysis functions.

3.4.4 Model refining

During the design phase, theses modules and database objects are refined and completed: tables usages as well as columns usages are defined for each module while cross-reference controls are still performed. The Design phase includes two broad sub phases :

- physical design of the database (there are already a complete ERD from the Analysis phase and a conceptual design of the applications from the Pre-Design phase);
- physical design of applications : to specify in detail how the application will interface with every field of the database.

As an example, the column display properties of the MEETING table are given below.



Column	Prompt	Display type	Width	Height
ROOM_NUMBER	Room number	Text	10	
MEETING_DATE	Meeting date	Text	10	
BEGINNING_HOUR	Beginning hour	Text	6	
END_HOUR	End hour	Text	6	
KIND	Kind	Pop list	4	
SUBJECT	Subject	Text	20	4
CANCELLATION	Cancellation	Check box	1	

Security and access control is designed during this phase.

3.5 Coding and unit testing

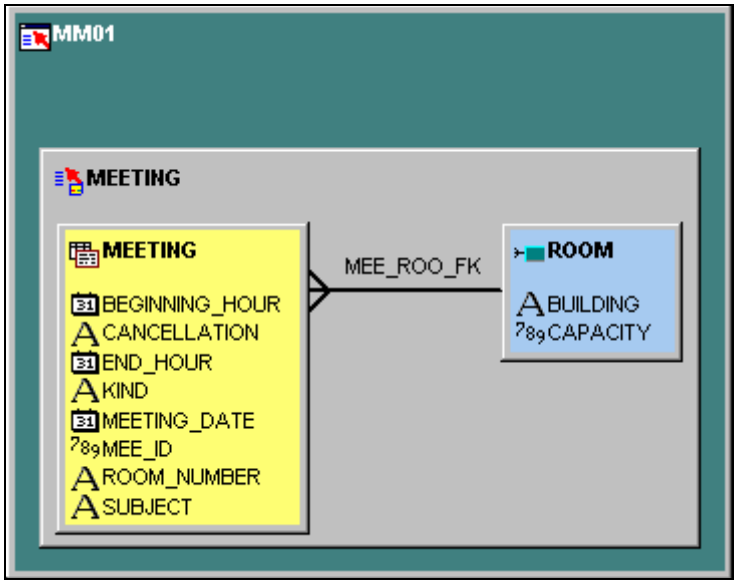
The build phase involves two areas : the database and applications.

Database building is a straightforward SQL generation operation, including physically configuration and building a quantity of test data or and/or data migration.

Application (module) building is a generative process, except for the design and implementation of stored procedures (in this particular case, Designer facilitates modules editing, code generation and ensures the consistency of the repository). Module generation is an iterative process : generate the modules and assess how different they are from the desired modules. Completing the internal control system can take different approaches : making change to the design and regenerating the module; otherwise editing code modifications and performing reverse engineering where possible.

3.5.1 Data View Model

Each module has his own data view model, like below for the “Create/Update Meeting” module.



3.5.2 Display View Model

Each module has his own display view model, like below for the “Create/Update Meeting” module.

3.6 Integration and integration testing

The application and the database cannot be tested completely separately. Both components must be tested and tuned together. The unit testing procedure should proceed as follows : to generate the application; to develop the application using dummy test data sets; to work with the application until satisfied; to test applications and reports using a database populated with a sample of realistic data; to run the application using a production-sized test database that will ensure adequate performance. The next step is performance tuning, which will not be discussed in this paper.

3.7 Software testing

Some of the activities and control executed during earlier phases contribute significantly to the success of the software testing phase, especially a formal acceptance of the testing process with the users (through a validation plan); the mapping of requirements to functions and modules; column level usages carefully peer-reviewed. One of the most important features of CADM is that as moving into the software testing phase, there is a complete audit trail allowing to logically follow a system requirement (gathered in the Analysis phase) all the way through the completed system. Theoretically, all that is left in the final test phase is system-level performance tests and user acceptance testing for the application. However, many phases are subject to changes and will require change control, but the discussion is out of the scope of this paper.

4 Conclusion

Model-driven approaches to systems development move the focus from programming language (3GL or 4GL) to models. The key challenge of model-driven development is in transforming higher-level models to so-called platform-specific models[3] that can be used to generate code.

For the model-driven vision to become reality, tools must be able to support the automation of model transformation. Many tools are now mature : this article has presented two approaches using model transformation and code generation.

We believe that some further questions need to be issued.

The complexity of a system description can only be described from different viewpoints, hence through the use of multiple models. Models can also to be decomposed into other models. Thus, models are used either in a horizontal manner (different system aspects) or in a vertical manner (from higher to lower levels of abstraction) [8]. This 2-dimensional space squares problems and tools complexity.

The two major approaches of system development are transformation and elaboration. The transformation makes an explicit distinction between abstraction levels (for example, between conceptual and logical level [7]) and advocates a transformation process between levels. The elaboration approach is based on a unique level and advocates a process by successive refinements [6]. UML/UP moves progressively from elaboration to transformation (through the MDA), while CADM/Designer relies on a transformation paradigm through an elaboration process. There is a kind of confusion of genders, may be detrimental to learning and mastering any approach.

[3] In the MDA terminology sense

5 References

[1] Richard Barker, Case Method: Tasks and Deliverables, Addison-Wesley Longman, 1990.

[2] Alistair Cockburn, Writing Effective Use Cases, Addison-Wesley Longman, 2001.

[3] CMMI for Systems Engineering and Software Engineering (CMMI-SE/SW, V1.1) Continuous Representation, http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr011.pdf

[4] Paul Dorsey and Peter Koletzke, Designer/2000 Handbook, Oracle Press, 1997.

[5] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison-Wesley Longman, 1999

[6] James Rumbaugh and al., Object-oriented Modeling and Design, Prentice Hall, 1991

[7] Sally Schlaer and al., A deeper look at the transition from analysis to design, JOOP, Feb. 1993

[8] Shane Sendall and Wojtek Kozaczynski, Model Transformation – the Heart and Soul of Model-Driven Software Development, IEEE Software, Special Issue on Model Driven Software Development, Sept/Oct 2003