MODELS TRANSFORMATIONS: FROM MAPPING TO MEDIATION

Bernard Morand
GREYC UMR CNRS 6072, Université et ISMRA de Caen
IUT Département Informatique, Rue Anton Tchékhov, B.P. 53, 14123 IFS CEDEX
Phone: (33) 02 31 52 55 24
Fax: (33) 02 31 52 55 22
Email: Bernard.Morand@iutc3.unicaen.fr

Abstract

This paper recalls some aspects of the transformation problem between models pertaining to different levels of abstraction, as it was solved in the database engineering field. We argue that resorting to model mapping techniques in this field has been possible only in part and also with regard to special circumstances. These circumstances were the standardisation set *de facto* by the Relational Data Model and mapping techniques were only successful with the help of transferring specific constraints from the logical model into the domain model. We then look closely at the new characteristics of the object oriented development process to show that these favourable circumstances have changed and suppose to modify the approach of model transformation and linking. We suggest a solution principle based on model transformation by mediation and some consequences of this approach for the MDA meta-model description.

1. Introduction

The development of OO applications meets today with a problem known for a long time in the database field: the definition of the relationship between a domain (or conceptual) model and its physical implementation. The generalisation of generic and standardised platforms is nowadays a good indication of the fact that OO technology has acquired a social autonomous behaviour, independent from the models and tasks of analysis as well as those of design and implementation. Object Orientation faces a similar problem as database engineering searching for an independent level from specific Data Base Management Systems used for the implementation of data. Database designers used to pass by an intermediary step, or a third level called logical model, independent of both the domain model and the physical model. From a methodological point of view, OO platforms play the same role than DBMS and this is how we understand the aim of the MDA project:

> *"The Model Driven Architecture defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on specific technology platform. To this end, the MDA defines an architecture for models that provides a set of guidelines for structuring specifications expressed as models"* Model Driven Architecture, [OMG Document 2001]

It is at first convenient to note that several years of know-how in object oriented software development end by the correction of a founding principle. It was thought at the beginning that the homogenous body of concepts available from requirements analysis to implementation would imply but one model. Then the software design process would simply increase progressively this unique basic model, from domain to software components. The advantage that was hoped for was a software architecture in the image of the domain objects structure, thus simplifying the understanding and allowing the extensibility of OO systems. But software industrialisation and interoperability between systems have favoured the rise of generic software platforms (CORBA, J2EE, etc.) which appear to be *de facto* standards for the so called "middleware" level. These platforms obey to their own rules and models the reason of which seems to come from the relative autonomy of the available technology at a given time. In particular, the object oriented development process was led to take into account these specific technical aspects by recommending their modelling on an equal footing with the domain objects (the right upper branch of the Y development process). Thus, in order to design a software architecture it is

requisite to relate different types of models at different levels: domain, technical aspects and tools, and software component organization (the lower branch of the Y).

The MDA view recommends two related models of the system under construction: a Platform Independent Model (PIM) and a Platform Specific Model (PSM) linked by a *Mapping* relationship and its reverse relationship of *Refactoring* (Figure 1)[1]:

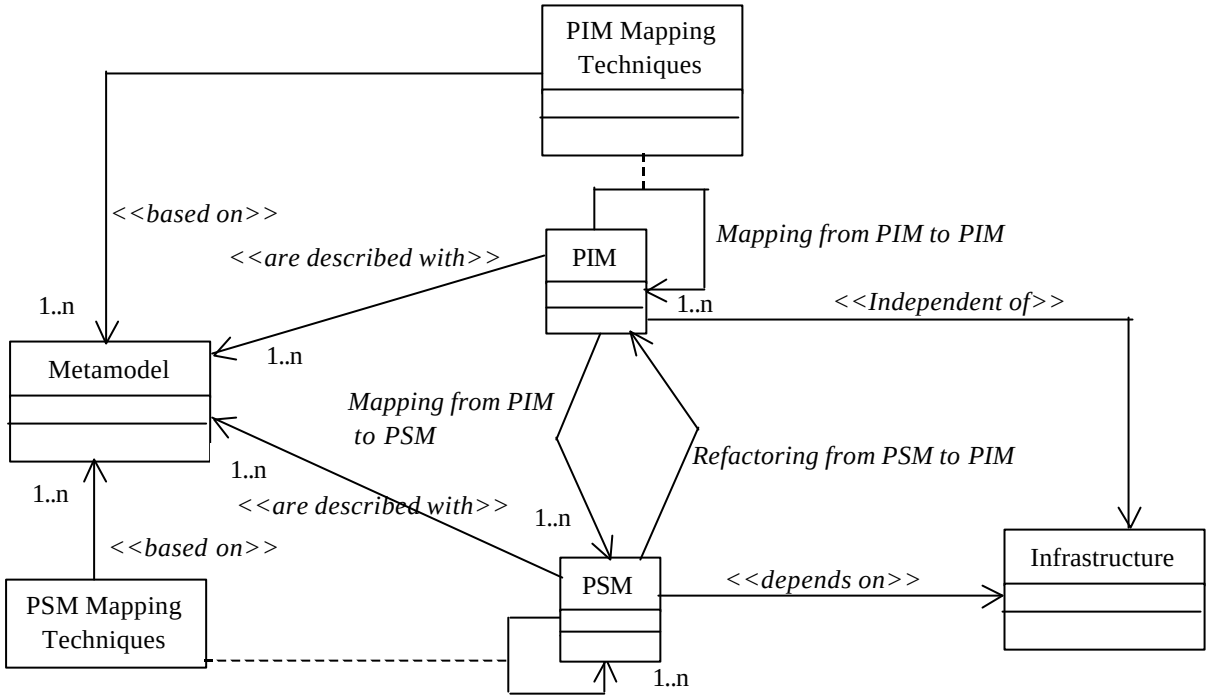

Figure 1: MDA Meta-model Description (extract)

Although MDA only addresses explicitly the computer system specification, it is assumed that the domain model can be integrated consistently in this architecture[2]. This approach by levels of inter-related models is very similar to database engineering techniques used in the 1970's. In a first part of the paper, we recall these model linking techniques, their benefits and limits, aiming to draw lessons for today. In the second part, we identify key points which lead to modify the approach of model mapping based on meta-modelling. We then suggest an alternative approach for model transformation based on a triadic relationship called intermediation.

2. What database engineering can teach us

Considered as a matter of software architecture, databases offered a solution to the drawbacks resulting from independent applications development, built as things came and for the required functionalities mapped to specific needs of each application. Databases allowed to take benefit of information coherence (avoiding data redundancy), of information sharing between applications and users, of data persistency as well as transactions security. However, two types of questions were immediately asked. The first was the design of a software that would be independent of the immediate applicative requirements. The answer consisted in introducing the concept of modelling for analysis, a concept which was already common in software realization (languages and algorithms) but of little use in domain analysis. The other side of the answer consisted in a principle of separation between information structuring (database schema) and manipulation of this structure (SQL with some procedural lan-

[1] The status of the Class *Infrastructure* in the figure shows the problem that we are addressing in this paper. The indirect specification of this class lies on the links labelled *Independent of* and *Depends on* coming from their respective sources, PIM and PSM. But such labels bear no meaning at all.

[2] "*This computation-independent description is sometimes referred to as domain model. While it need not be explicitly present in a particular usage of the MDA scheme, MDA accommodates it consistently in the same overall architecture*" (op. cit., note 7 pp. 7)

guage). The second type of question was the relation between different levels of models since the domain model (often conforming to the Entity-Relationship Model) was clearly distinguished from the logical model (which quickly imposed standard was the Relational Model[3]). The answer to the second question was clearly set by the [ANSI-SPARC] report in 1975. It prevailed implicitly or explicitly until the arrival of object oriented approaches. This report recommended for the first time to distinguish different levels of modelling: a ***conceptual*** level independent of technology (**M_1**), a ***logical*** level (**M_2**) independent of the physical implementation (**M_3**)[4] and finally an ***external*** level (**M_4**) representing the users' view of the system. MDA does not take this external aspect into account but it is clear that we are facing the same question, namely the relationship to be defined between different models.

The chosen solution already was that of mapping between models that we define as follows: each modelling element used in model M_i (the definition of which is eventually stored in a meta-model) is linked to one or more modelling elements in model M_j. It is then possible to transform a source model M_i in a target model M_j by means of correspondence rules (or functions) applying to each element (or group of elements) in M_i. These rules that can be seen as logical deductions, can be automated and many software engineering tools deduce from any Entity-Relationship Model (M_1) the corresponding relational database schema (M_2). The question we are concerned with is to describe the conditions in which this has been made possible and the limits of this technique. The basic models concerned, Entity-Relationship and Relational, come from different contexts: the first one relates to semantic networks while the other is founded on the mathematical theory of sets. The people implied were also in charge of different responsibilities: analysis *versus* development. The first ones tended to consider their work as essential (modelling the domain) while the others' a subsidiary task limited to encoding. On the other hand, developers tended to consider the domain model as a data (set by a specification) which was out of their field of responsibility and focussed on technical implementation problems.

The consensual cohabitation of populations taking part in the projects, as well as the cohabitation between modelling techniques, was made to the cost of reducing the model M_1 to the principles required by model M_2. It has only been possible to industrialize the mapping process from M_1 to M_2 when it was acknowledged that the conceptual model M_1 would be in 3rd Normal Form. If this prerequisite is legitimate regarding 2nd and 3rd Normal Forms which express semantic constraints on relations and objects, it is different for the 1st Normal Form. The latter requires the attributes to be mono-valued and it is a syntactic constraint imposed by the set theory foundation of the relational model[5]. Thus, for the mapping process to be operational, some constraints were introduced in the conceptual model that were so foreign to it that its proper function was disputed: if the difference between M_1 and M_2 is reduced to a difference of vocabulary (entity vs. table, identifier vs. primary key, relationship vs. foreign key), why not directly express M_1 in the vocabulary of M_2? In fact, the use of M_1 was stabilized for practical reasons: communication between the designers and the end users of the system. It is worth to note that the rise of M_2 specific constraints to M_1 has only been successful because M_2, the relational model, imposed itself as a standard. It has not been practically necessary to transform from M_1 to other types of logical models M_2[6]. This is an additional difficulty to take into account in PIM / PSM transformation, which will be addressed in the next section. Last, the transformation process that was generalised in databases presents a drawback that can become essential today: the M_1 -> M_2 function is not always reversible. It is not always possible to match an element from M_2 with its corresponding structure in M_1, because the relation source-target is not reflexive in general. Concerning the second transformation, from the logical model M_2 to the physical model M_3 the mapping techniques can hardly be used. It is a matter of optimisation process that often requires a de-normalisation of the relational schema and the reasons of which are unrelated to M_2 itself (a reason of data access frequencies rather than of formal data structures).

---

[3] Let us recall however that two other concurrent logical models have coexisted for a while, the hierarchical model and the network model, thus creating a similar situation to that of platform independent model, susceptible of deployment on specific platforms (DBMS).

[4] For example an abstract relational model as independent of a physical model, the latter depending on a specific DBMS.

[5] It was overcome by later models but we shall not discuss this matter here.

[6] Conversely, works on logical models conforming to the Entity-Relationship Model have rapidly lost any reason of being, as soon as the Relational Model became a standard for the logical model.

The model transformation know-how in database engineering shows that the mapping process works partly fine but to the cost of very strong constraints imposed on the source models to be transformed. It produces, at best, a paraphrase of a source model into a target model, not always reversible and sometimes impossible; we are far from a real translation in the sense of natural languages. We now examine if the actual software engineering context allows to continue on this track, a technique which is presupposed in the Model Driven Architecture with its independent links *Mapping* and *Refactoring* between PIM and PSM.

3. What has changed in the models functions and the relations between models

Three essential characteristics of software engineering have deeply modified the way models are used and consequently lead to re-evaluate the modes of connecting models. The first characteristic relates to the development process which results from the Object Oriented approach. The second one relates to technological evolutions as well as the toolbox available for software engineering and it is at the origin of the new concept of Software Architecture. The third characteristic concerns the nature of the logical level itself and its role in articulating different models, in particular the place "in the middle" between the domain model (or Business Model) and the physical components model.

3.1 The shift in the methods paradigm

Before the generalisation of object oriented approaches, numerous methods had been developed and each had specific features but all presented a family likeness. They were based on a two-pass modelling, data modelling on one hand and process modelling on the other. This splitting of the information system at the domain analysis level as well as for the software implementation was in phase with the database founding principle. Methods such as Merise, SSADM or SADT for example, are representative prototypes of this family of methods. The design in two separated parts of an information system has made possible the *a priori* models validation, before any effective software development. In fact, having two sets of models, applying to two distinct views of the information system, it was possible to proceed to an inter-models comparison by a cross-referencing process (mapping). Typically, the designer could make sure that the data access needs for a given functional service constituted a coherent sub-schema of the data model. Inversely, the designer could make sure the data items specified by the data model were used by the functions asked for by the users. From the analysis process point of view, the essential role of models was the models validation in terms of exhaustiveness and coherence[7]. From another point of view, analysis was med in a global way over the whole domain, which made possible a general description of modelling as prior stage for implementation.

The Object Oriented approach takes place oppositely on all these points. First, it implies the reunions of both data and activity aspects under the unique concept of object, which allows resorting to the cross-referencing principle only at the local space of a class. Then, the incremental development process by successive iterations allows managing in parallel both functional requirements analysis and software objects organization but it does not allow any more this *a priori* validation of a global model for the project. Both these fundamental changes explain that no previous generation method, to our knowledge, has succeeded in transforming itself to an object oriented method from its owns foundations. But, at the same time, an essential reason to resorting to modelling techniques (*a priori* models validation) has disappeared. Even if models can serve locally to verify certain correspondences between internal elements of their structure, they can validate a project only by its efficiency, that is, its execution. **It thus seems to us that the mapping technique between structural components of models becomes insufficient if it is not accompanied by a possibility of simulation.** As an example, it is possible to ensure that a UML dynamic diagram relative to a class instance bears features which are described by the attributes and the methods of its class static diagram. However, it would be useful to make sure that the expected properties are preserved during the execution of an objects subset, belonging eventually to several classes (management of transition trigger events). Such a prerequisite seems transposable to larger extent models such as the PIM / PSM relations of *Mapping* and *Refactoring*.

[7] Let us note however that this validation was not worth a formal proof: the corresponding algorithm may not terminate. Note as well that the cross-referencing process of a function with its data sub-schema involves derived data the status of which has remained undefined by the modelling techniques.

3.2 Making technical tools usage autonomous and general

The family of methods we have just described no doubt had as target a specific software system, Data Base Management Systems, even if the conceptual models were meant to be independent (c.f. the role of the 1st Normal Form, section 2). In parallel, other families of methods were developed for other specific software systems (such as SART). Inside each of these great application domains, the existence of a unique technical solution constituted an implicit context for the construction of models. In a way, if one was freed from material architecture constraints dedicated to some types of applications, the situation was fairly different for software architectures. Each domain (data processing, real-time) had its own implicit software architecture. The maturation of generic software components, defined on the basis of generic services they can supply, and not on the bases of usage context, modifies in turn the nature of the logical model itself. In a data processing oriented domain, for example, the logical domain was determined by a unique solution, that of the relational model. Today, the database has become a common place, in charge of maintaining data persistency and access, taking an equal part with other components in a more complex software organisation. Such an organization, previously dictated by the available and predefined technical platforms (DBMS), now becomes a modelling subject as such [Morand 2001]. The logical model aiming at platform independence represents an architectural choice of software components, itself subject to different implementations. Furthermore, the distributed aspect of computerized systems contributes to make the logical model a first class model, by difference with a simple model derived from conceptual domain models with the constraint of a predefined toolbox (cf. 3.3).

Another major change factor for logical models deserves to be noted. A specific aspect of Software Engineering with regard to other disciplines is not only to integrate its tools in its own product but also to integrate its products in its own raw material. The previous models and methods targeted implicitly a computing system starting from scratch for which domain objects were effectively independent of any previous implementation consideration. The situation is not similar today since new developments must cohabit as harmoniously as possible with prior software products. This proposal accounts for the appearance of the concept of **reusability** and it can be rephrased in the following words: domain objects are also and from the beginning software objects. This is to say that the time-frame validity of logical models has changed in scale. Whereas this frame used to spread over the limited time of a project development period, occasionally increased by anticipations on maintenance tasks, it is now necessary to take into account the constraints due to pre-existing software (reverse engineering) as well as, on the other side, the constraints linked to future evolutions (extensibility). The important consequence is that the PIM / PSM relation, whichever the way it is considered, is neither a static nor a universal relation but a temporally indexed relation. Taking these properties into account seems to us likely to modify importantly the postulated independence between the two links « *Mapping from PIM to PSM* » and « *Refactoring from PSM to PIM* » in the MDA meta-model (Figure 1).

3.3 From Middleware to the relations system implied by the logical model

We summarise the two previous points to show that acknowledging an independent and explicit modelling level of technological platforms in Object Oriented design brings the old approach to logical models, but that it also imposes generalisation and extension. In fact, the essential difference resides in the nature of the relationship between models (Figure 2: on the left, successive mapping linkages; on the right, transformations by mediation). Instead of specifying the result of a functional correspondence between two correlates, a triadic relation specifies a mediation process between them. Hence, the Business Model is an intermediary that stands on one hand to domain objects for a logical model, and on the other hand the logical model constitutes a mediator between the implemented software and a specific implementation platform.

The triadic character of this relationship has essential implications for a representation theory that not be discussed here. Borrowed from C. S. Peirce's semiotics [Morand 1998], the important properties of a triadic relation deserve however to be recalled:

- as a logical relation, it is susceptible of a characterisation independent of the correlates existence (this explains why it can be found twice in figure 2, right hand side, with different correlates: as representation relation but also as implementation relation)

- it can not be reduced without loss into two binary relations. This property is a consequence of the mediation concept. More precisely, mediation is explicitly linking between two things or subjects (A and B) by means of a mediator (M). Its reduction to the pairs A-M and M-B looses the linking property of M (a mediator that would ignore itself, in a way)
- since it is logical, the orientation of the relationship is not set in its definition. As a principle, both subjects of the mediation have interchangeable roles without affecting the mediator. In a customer-provider conceptual relationship, for example, the service works as a mediator that is affected neither by the customer role nor by the service provider role. However, the instantiation of this relation, by affecting correlates to roles (i.e., by individualising with identification) can degenerate it in a pair of oriented binary relations that express the fact that the customer acquires the service and the provider supplies it. But the advantage of making the underlying triadic logical relation explicit is that the acquirer and the supplier can change roles without the service being affected.

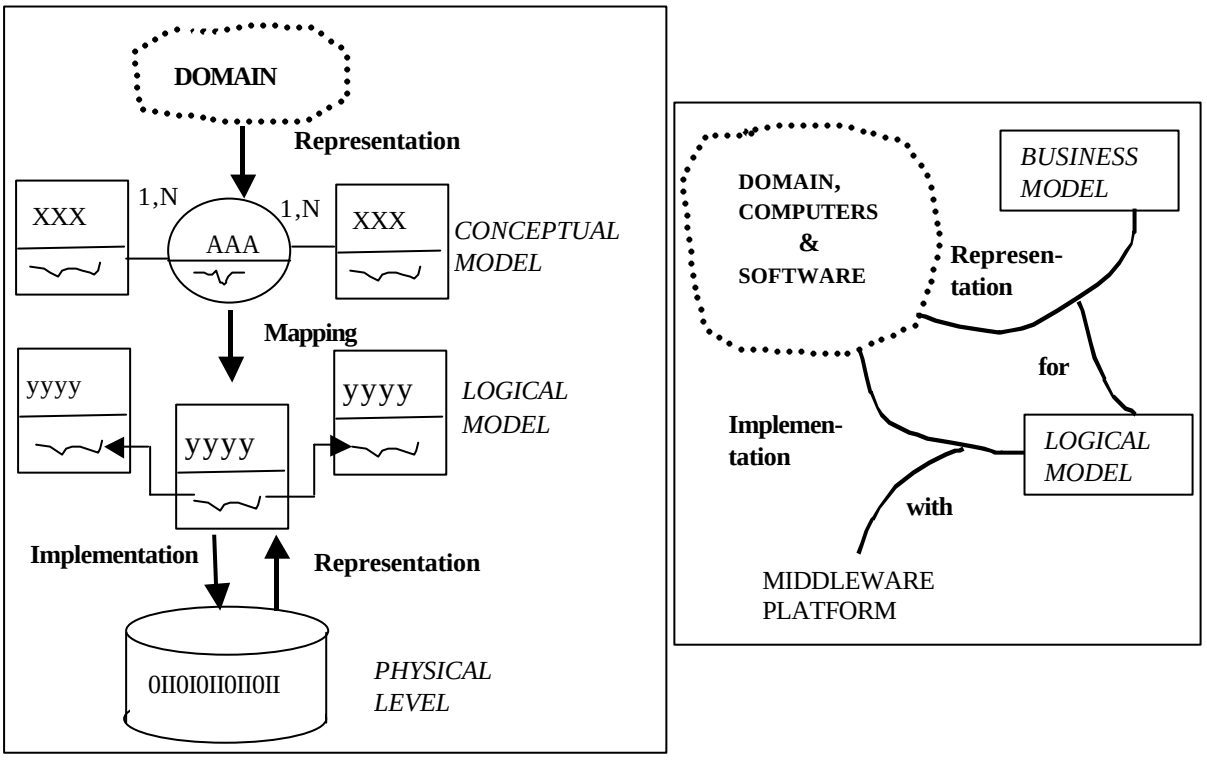

Figure 2: Change of roles and relationships of the logical model in DB and OO modelling

An important consequence of the previous properties lies in the fact that the mediation relationship authorises two working modes, forward and backward. Hence, in figure 2 (right hand side), it is conceptually possible to look at the logical model as a specification for an implemented software system by means of an implementation platform. This forward movement is "accretive" in the sense that implementation adds information to the logical model and it is called **synthesis process**. A backward working of the same relation is also possible: from the implemented system, the logical model can be deduced by abstracting the technical platform characteristics. This movement does not add any information to the knowledge of the system but makes it possible to apply a validation technique for the implementation. This backward movement as an **analysis process**. Doing and undoing models is only possible by means of the properties of such a triadic relation. We have shown, on the contrary, how the logical model of older approaches (figure 2, left hand side) could reach this only to the price of very restrictive conditions on the content of the models themselves.

If we compare now this conceptual approach with the MDA description in figure 1 (leaving aside the role of meta-models), the proposal consists to:

- define a unique node in order to link simultaneously the PIM, PSM and Infrastructure components and,

- give this unique node the operators described as static links between instances: *Mapping from PIM to PSM* and *Refactoring from PSM to PIM*.

The node thus defined seems likely:

- to avoid the two associations, respectively *Mapping from PIM to PIM* and *Mapping from PSM to PSM*, which operational semantics seems limited to model version management;
- to supply real semantics for the two links *PIM "independent of" Infrastructure* and *PSM "depends on" Infrastructure*;
- to consider the mapping techniques as a case of transformation operators among others. This is similar to considering the node in a triadic relation as a specification by means of operations abstraction (and not a data structure anymore). We thus get a generic mediation function that is independent of the mediated objects.
- to set the foundations of a transformation between models that would be executable by simulation, thanks to the operators specification.

4. Towards an approach by models mediated co-operation

The approach of model transformation by mediation needs further works. In particular, we have not discussed works based on a meta-modelling principle that constitute the OMG reference framework (with the *Profiles*). In fact a meta-model is a model's model. According to the proposed triadic relation definition, a meta-model stands to a model for itself. It is but a special case where its object is considered as momentarily stabilised, that is, dealt with as a constant. The advantage of such a static abstraction is the attribution of properties to the object model and their explicit description into the meta-model. Oppositely, this static characteristic does not allow to account for the dynamic nature of modelling processes where the object of the model changes with time. Hence, MDA description (figure 1) is constrained to envisage two "*are described with*" associations between the PIM / PSM classes and the *Metamodel* class. An association links several instances of the class *Metamodel* to one specific model object instance (from the PIM class, for example). But, as in the case of mapping techniques, the internal links between the former several instances in the class *Metamodel* remains undetermined excepted that they relate to the same instance in PIM. One can thus fear that the meta-modelling approach may meet the same difficulties to account for model transformations as those met by the first level modelling.

Another direction of requisite and future work for the mediation approach is its own logical and implementation models. However partial solutions already exist because Object Oriented software engineering has met them experimentally in other contexts. The essence of our proposal relies on an observation that is as simple as trivial: every model is an informational object just as much as any software or domain object; it must thus be dealt with as such. In particular, we consider the node that represents mediation between two models as logically fitting into the *Mediator* Design Pattern as it was already presented in 1995 [Gamma and al.]. Similarly, the solution centred on the *Broker* concept in the CORBA model belongs to the same family of discoveries in software development. The only difference lies in the context of these discoveries, but the solutions they bring belong to our opinion to the same approach: mediated co-operation between Objects. We just have suggested to regard Models as Objects.

References

[ANSI-SPARC] *Report*, ACM SIGMOD Newsletter, Vol.7, N°2, 1975

[Gamma and al.] *Design Patterns*, Addison Wesley Publishing Company, 1995

[Morand 1998] *Modeling: Is it turning Informal into Formal?* <<UML>>'98 : Beyond the Notation, J. Bezivin & P.A. Muller (eds). Springer Verlag LNCS 1618, (1999).

[Morand 2001] *Cinq approches pour définir le concept d'architecture logicielle*. Journée du PRC GdR I3, Lyon, 13 décembre 2001.

[OMG Document 2001] *Model Driven Architecture*, ormsc 2001-07-01