

Transforming the OOram Three-Model Architecture into a UML-based Process

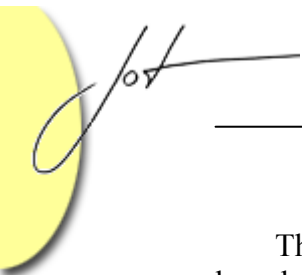
Jesús García Molina, María José Ortín, Begoña Moros, Joaquín Nicolás,
Departamento de Informática y Sistemas,
Universidad de Murcia, 30071 Murcia, Spain

Abstract

Three-model architecture (TMA) is a software process defined for the OOram method, and aimed at developing business information systems. In our experience, TMA is very helpful in building client-server applications using object-oriented and database technology. However, in order to use a standard notation and to take full advantage of the benefits provided by use case-driven processes, it is convenient to transfer TMA to UML. In this paper, we present the translation of TMA into a UML-based process. The enterprise, information and task models of TMA are translated into UML models while preserving their original purpose. An important benefit of the process obtained is to provide guidelines for the elicitation of use cases and domain classes from the enterprise model.

1 INTRODUCTION

OOram [Reenskaug 1996] is a method, based on the concept of role, for performing object-oriented modeling. Three-model architecture (TMA hereinafter) [Reenskaug 1997] is a process defined to support the analysis of information systems by means of OOram. Some years ago, we used this process for modeling a workflow information system in a project aimed at developing the workflow tax system for the Regional Information Systems and Telecommunications Office in the Regional Government [Ortín et al. 1998]. This experience made us to realize that TMA is very suitable for developing business applications involving object-oriented and database technology in a client-server architecture. In addition, we improved TMA by using techniques drawn from the IDEA method [Ceri Fraternali 1997] to undertake the database design. Furthermore, we realized later the usefulness of using UML [Booch et al. 1999], instead of the techniques of OOram and IDEA, since UML is the OMG standard language for object-oriented modeling.



Thus, in this paper we describe how to translate TMA into a use case-driven UML-based process, by showing the mapping from the OOram concepts into those of UML, and the way of expressing the OOram models through UML diagrams.

It is important to remark that we do not deal with the problem of expressing the role concept of OOram in UML. This problem was addressed in a previous paper [Ortín García-Molina 1999], which was presented prior to the interesting discussion on this matter which took place in the UML RTF forum [OMG 2000]. Moreover, role modeling is not essential in the underlying techniques of TMA.

This paper is structured in the following way: TMA process is briefly described in section 2; sections 3, 4 and 5 deal with the mapping from each model of TMA into UML. Finally, in section 6 we set out our conclusions.

2 THE THREE-MODEL ARCHITECTURE OF OORAM

Three-model architecture [Reenskaug 1997] is a software process based on the building of three models: *Enterprise Model*, *Information Model* and *Tool Model*. The first is used for identifying the roles that are played by the workers in the organization, and how they collaborate in order to fulfill the business tasks. The second model describes the information managed by the enterprise. The third model shows the interfaces between the users and the services which provide access to the information (to databases), that is, the software tools used by users in order to perform their tasks by accessing the information services. Figure 1 (extracted from [Reenskaug 1997]) shows the relationships between the three models. We can see that tasks and business information are provided by the enterprise model, and the operations supported by the information model are defined in the tool model.

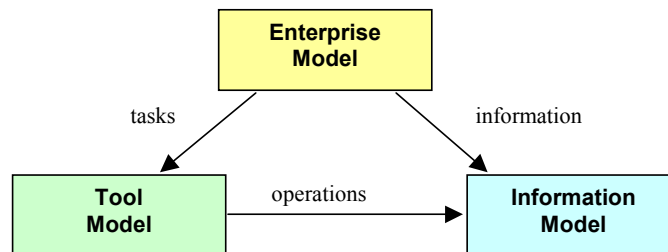


Fig. 1. Three-model architecture

In order to apply the TMA process, first of all the *areas of concern* of the organization are identified, which form a partition of the overall system. Each area of concern is a problem that can be modeled in an independent way, by building the three models.

The approach is both analytical and synthetic: firstly, the existing business processes and data (current situation) are modeled by means of an enterprise model and an information model; secondly, these two models should be refined to include new requirements and improvements (future situation), by adding the tool model. An iterative process is applied until these models are stable.



3 ENTERPRISE MODEL

The TMA enterprise model provides a static and dynamic description of the organization. The first defines the enterprise workers, through the roles they play, establishing for each role its features and the set of roles with which it interacts; on the other hand, the second one defines how the roles interact in order to perform the tasks involved in the area of concern. It focuses on describing and understanding the whole business activity related to the software system which is going to be developed and deployed.

For us, each area of concern is a *business process* of the organization which is characterized by the information that it produces and manages through a collection of tasks in which certain agents take part (as workers or departments), and business process is performed according to a workflow. These business processes are constrained by business rules which determine the policies and the structure of the information of the enterprise.

In this section we will describe the way of translating the TMA enterprise model into UML, after explaining how to identify the business processes.

Business Processes and Actor Identification

First of all, we have to identify the *strategic goals* of the organization under study, in order to obtain the set of business processes. Due to their great complexity, every goal can usually be decomposed into a set of more specific subgoals (which must be fulfilled for the strategic goal to be reached). These could also be subdivided into other subgoals. In this way, a hierarchy of goals of the organization arises. Our experience indicates that two (or a maximum of three) levels of decomposition are enough. The second level goals correspond to business processes, which are described through *business use cases*.

We will use as running example the case study of a company that manufactures products on demand (following a *just in time* scheme). The strategic goals of that company might include *Satisfying a customer order*, *Increasing sales by 25%*, or *Reducing the manufacturing time by 15%*. Thus, the goal *Satisfying a customer order* can be divided into the following subgoals: *Registering the order*, *Manufacturing the product*, *Stock management* and *Generating orders to providers*. These are the subgoals that we will use to identify the business processes.

Now, we have to ascertain which agents are involved in the performance of the identified business use cases. In order to do this, we look at the *external agents* of the organization and the *roles* which they play when they collaborate to carry out some business use case. For each role, a *business actor* is defined. In our example, we have two roles which are clearly external to the organization: *Customer* and *Provider*.

To show the boundary and environment of the organization under study, a *business use case diagram* is created (see figure 2). This is an ordinary *UML use case diagram* but it shows the set of business use cases instead of the system uses cases, along with the business actors involved.

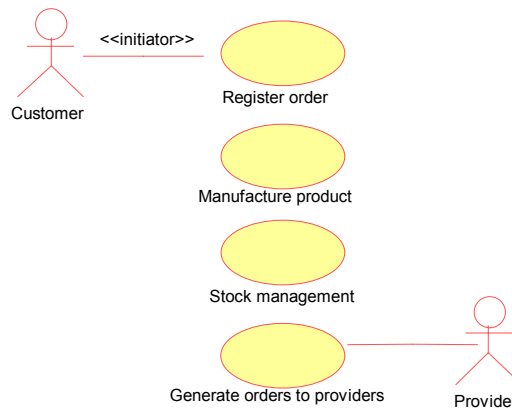
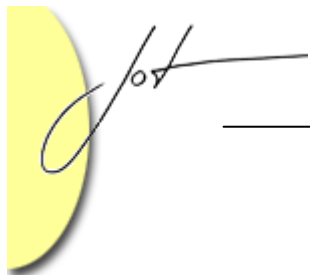


Fig. 2. Business use case diagram for the "just in time" manufacturing system

Business use cases description

In TMA, once the areas of concern have been identified, they are described in detail by identifying first the roles involved. Given this approach, we now focus on describing every business use case. Initially, they are described textually and their whole set of roles played by both internal and external agents is identified. We will take one of the business use cases of our example, namely *Register order* (see figure 3 for description), which can be validated with the users. The roles identified are *Customer*, *Clerk*, *Catalog manager*, and *Manufacturing manager* (with the last three being internal to the system).

1. A customer submits an order, which has to include the order date, customer data and the desired products. A clerk of the sales department might also introduce the order on request of a customer who has placed their order by phone, or has sent it by fax or ordinary mail to the sales department of the company.
2. The clerk checks the order (and completes it, if necessary), and begins its processing by sending it to the catalog manager, who is in charge of its analysis.
3. The catalog manager analyses the viability of each product of the order separately:
 - if the product ordered is in the catalog, its manufacturing is accepted;
 - otherwise, it is considered as a *special product*, and the catalog manager studies its manufacturing:
 - if it is viable, the manufacturing of the special product is accepted;
 - if it is not viable, the product is rejected.
4. Once the whole order has been studied, the catalog manager...
 - informs the sales department if every product ordered is accepted or rejected;
 - in the case that all the products of an order have been accepted, a work order for every product is created, starting from a manufacturing template (the standard one, if the product was in the catalog, or a new one, specifically designed for the product if it was not in the catalog). Every work order is sent to the manufacturing manager, and its launching is considered pending.
5. The clerk informs the customer whether the order has been accepted.

Fig. 3. Description of the *Register order* business use case

In TMA, the *collaboration view* is used for showing the static aspect of the collaboration among the roles within each area of concern. In our approach, a *role diagram* is used



[Ortín García-Molina 1999], which provides the same information as a collaboration view. A role diagram (see figure 4) is a UML class diagram in which every role (a stereotyped UML class) appears linked to the roles with which it can collaborate. Thus, such a diagram allows us to express the knowledge that some roles have about the others as well as the characteristics of the relationships between roles (i.e. multiplicity). In addition, the diagram can serve for defining some characteristics of the roles identified, for example their attributes and responsibilities.

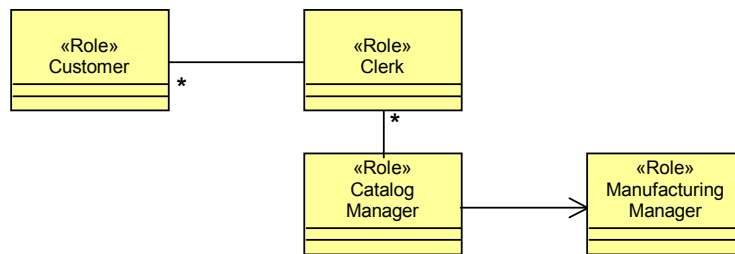


Fig. 4. Role diagram for the *Register order* business use case

On the other hand, TMA uses *scenario views* for describing the dynamic aspect of the collaboration among the previous roles. In a similar way, UML *sequence diagrams* are used in our approach with the aim of describing how the roles collaborate to perform the related business use case (see figure 5). Therefore, the objects that appear in the sequence diagrams are instances of the roles defined in the role diagram for the business use case that is being modeled.

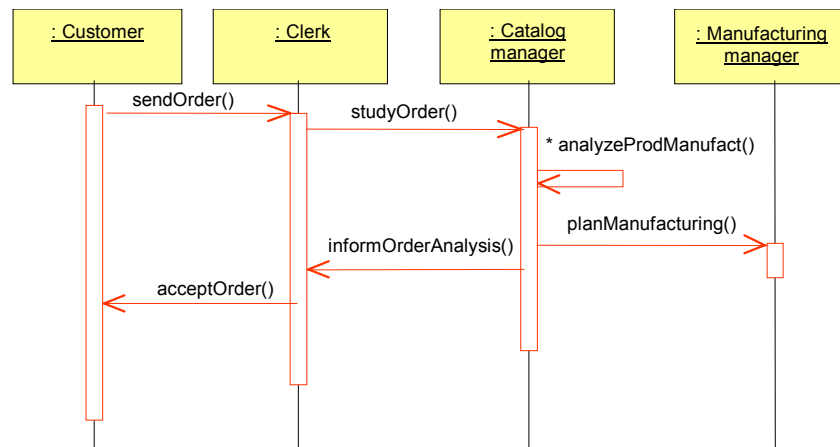


Fig. 5. Sequence diagram for the *Register order* business use case

In every business use case the normal (or basic) interaction flow (*accepted order*, showed in figure 5) has to be distinguished from the possible alternative flows (*accepted* or *canceled order*).

Finally, TMA uses the *process view* (based on the IDEF0 standard [CSL-NIST 1993]) with the aim of describing the workflow performed to reach some goal of the

organization, and indicating the role in charge of every activity, together with the data produced and required by each one. *Activity diagrams with swimlanes* are used to translate this TMA view to UML (see figure 6). We call these *process diagrams*.

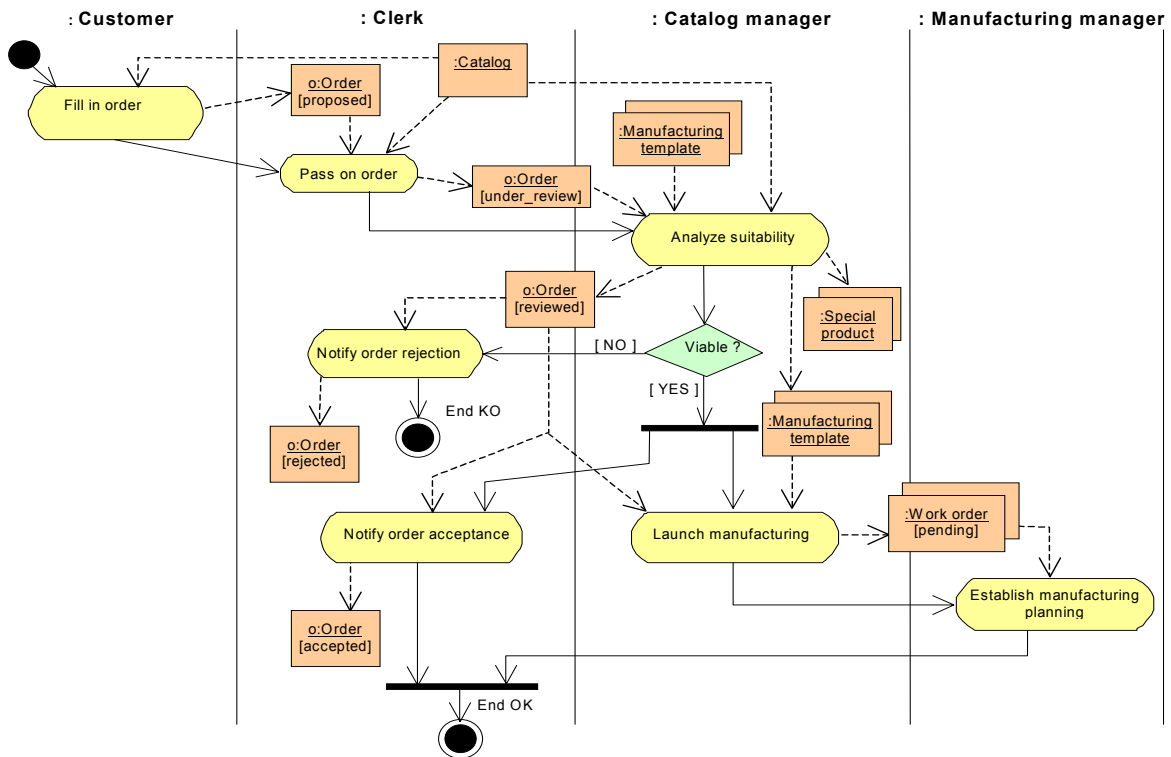
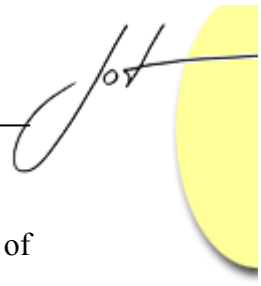


Fig. 6. Process diagram for the *Register order* business use case

A process diagram is built by starting from each sequence diagram. This process diagram consists of a swimlane for every role involved in the related sequence diagram. This swimlane contains all the activities performed by that role. The process diagram also shows the data needed and produced by each activity, and the synchronization required between different activities. Data appear as objects which flow between activities and can be labeled with their state. We refer to these objects as *information objects*.

During the description of a business use case by means of a process diagram, it is possible to find an activity which is complex enough (like the *Analyze suitability* activity in figure 6) to be described in another activity diagram. This new activity diagram will therefore describe a subgoal related to the goal associated with the original business process. In this fashion, business processes can be hierarchically organized.

The glossary (an *Information Resource Dictionary*) includes the description of the activities and information objects that appear in the process diagrams. The semantics of each activity will be illustrated by its *source* (that is, the previous activities), *agent* (who is the responsible for doing the activity), and *pre* and *post conditions* (stating which have to hold before and after the activity). Each information object is described by a set of



attributes together with their integrity constraints (if any). Figure 7 shows an extract of the glossary.

... Information Object: <u>Order</u> Attributes Order code Submission date Maximum delivery date Set of {Products} Customer Total price Current state Constraints - Order code uniquely identifies the order, and has to be assigned automatically by the system - Submission date has to be previous to the maximum delivery date. - An order must contain at least one product, but there is no maximum number. - An order is always processed for one (and only one) customer. - The total price is calculated starting from the price of every desired product of the order.	... Activity: <u>Launch manufacturing</u> Source: Analyze suitability Agent: Catalog manager Precondition: All ordered products are viable and a manufacturing template exists for all of them. Postcondition: A work order for each product has been created, and has been sent to the Manufacturing manager for planning. Use Case: - <i>to be specified</i> - Activity: <u>Notify order acceptance</u> Source: Analyze suitability Agent: Clerk Precondition: All products ordered are viable and have been accepted. Postcondition: The customer is informed that his or her order has been accepted. Order state is 'accepted'. Use Case: - <i>to be specified</i> - ...
--	--

Fig. 7. Glossary: information objects and activities

The *business rules* of the organization [García Molina et al. 2000] are also collected in the glossary, which, moreover, allows us to establish traceability relationships between the different modeling artifacts.

System use cases identification

Most of the main processes currently proposed for UML (such as UP [Jacobson et al. 1999]) are defined as use-case driven. However, the use case concept is not provided by OOram, although TMA implicitly associates a system function to every task in the enterprise model, which will be performed by the software system as described by the tool model.

We propose to extend the enterprise model with a *requirements analysis* phase aimed at identifying and defining the system use cases. Therefore, two different models are obtained when the enterprise model is translated into UML: the *business model* and the *requirements model*. The former corresponds to the TMA enterprise model in a strict sense and has been described in the two previous sections.

The *requirements model* is constructed from the *business model*, which allows us to obtain an initial set of the system use cases (that is, the use cases that describe the functional requirements of the software-intensive system), which are the most important

ones architecturally. We have learned that it is appropriate to define a *system use case* (referred to simply as a *use case* hereinafter) for every activity from the process diagram that has to be supported by the system software, since we believe that both artifacts have a similar level of granularity. The role performing the activity will be the primary actor in the use case. It should be observed that, according to the use case definition, not all the activities in a process diagram will be considered as use cases, but only those which provide some value for an actor.

For instance, figure 8 shows the use case diagram obtained starting from the business use case *Register order* (the process diagram is described in figure 6), assuming that all the activities will be supported by software.

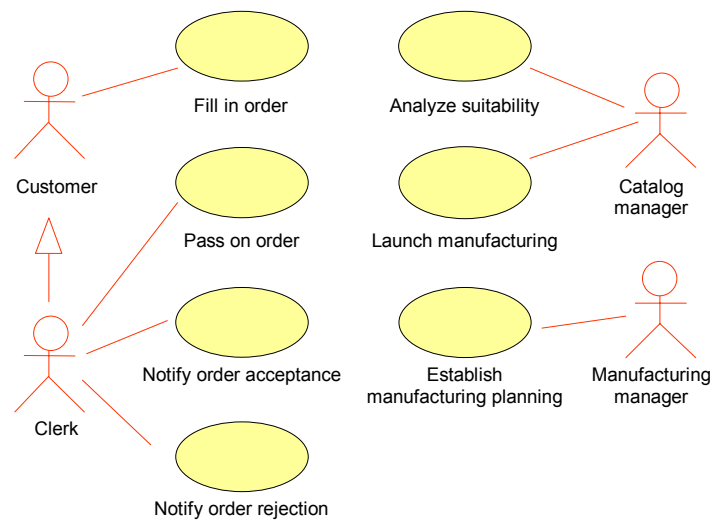
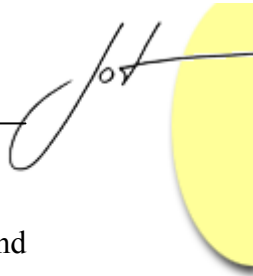


Fig. 8. Initial system use case diagram

Not all use cases will be directly obtained from the process diagrams. Some of them will appear as the result of approving –or modifying– new user requirements, and others will be detected during the iterative process of describing and refining the initial use cases. These new use cases represent functions that the system has to perform in order to reach the goal related to some existing use case. For instance, in our running example, to *Analyze suitability* (see figure 8) it is necessary to look up in the product catalog whether an ordered product exists, therefore this catalog must be kept up to date. Thus, we would have to add the use case *Maintain product catalog*.

Finally, the use cases should be organized into several levels according to the hierarchical decomposition proposed in the business modeling. Relationships between use cases could also be found, such as *include* and *extend*. Nevertheless, we support the recommendations (cf. [Cockburn 2000] [Fowler 1998] [Pols 1997]) about not overusing these relationships and not showing them in the use case diagrams.

Each use case will be described by means of a template which can be filled in starting from the specification of the associated activity included in the glossary. We have



chosen the template proposed by [Coleman 1998] because it combines simplicity and completeness, as is shown in figure 9.

Use Case	Launch manufacturing
Description	Work orders for every product ordered will be created, and will be sent to the manufacturing manager so as to be planned.
Actors	Catalog manager
Assumptions	All products ordered are viable and there exists a manufacturing template for all of them.
Steps	1. REPEAT 1 Get a product from the order. 2 Look for the manufacturing template of this product. 3 Create the work order. 4 Store the work order with 'pending' state.
Variations	-
Non-Functional	-
Issues	-

Fig. 9. *Launch manufacturing* use case description

It is worth remarking that the identification, description and organization of use cases is carried out at the same time as the conceptual model is constructed (described in the next section). In this way, a useful set of use cases, which use the domain vocabulary in the right way, is obtained.

4 INFORMATION MODEL

The purpose of the TMA information model is to describe the information managed by the organization under study. As we already stated in section 2, it consists of building the semantic view (a technique similar to the entity/relationship model) of the roles which represent the data extracted from the process view in the enterprise model (see figure 10). The expressiveness of this view is rather poor since it lacks many concepts needed for the conceptual modeling of information systems with intensive use of data.

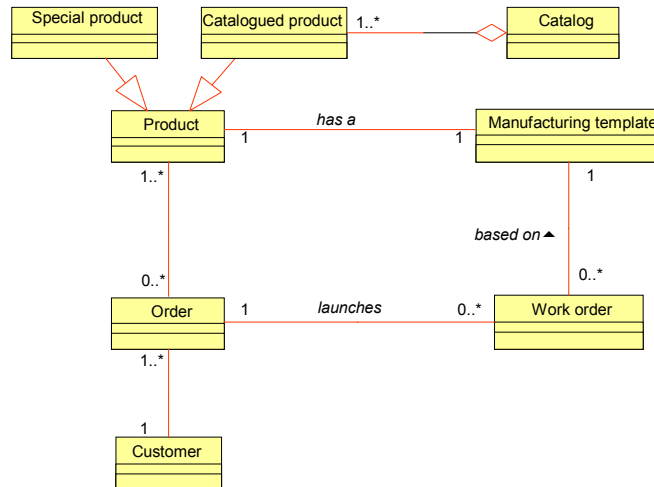
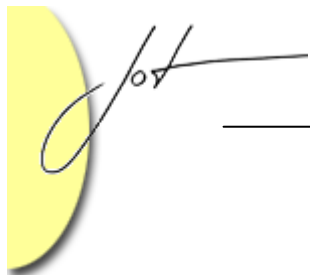


Fig. 10. Initial conceptual model for the *Register order* business use case

This conceptual model will be transformed into a design class model, made up of classes that are closer to the implementation ones. Our proposal extends TMA at this point, since this only addresses the analysis phase. To begin with, all the classes in the conceptual model are considered design classes. Next, new classes and associations could be added as the result of the refinement of the model and the use of design patterns [Gamma et al. 1994], although the building of the tool model determines those classes which will finally become design classes (as described in section 5). Figure 11 shows the design class model that includes classes coming from the conceptual model (for instance, this is the case of *Order*), as well as new classes to model the set of states of an order (by using the *state pattern* [Gamma et al. 1994]).

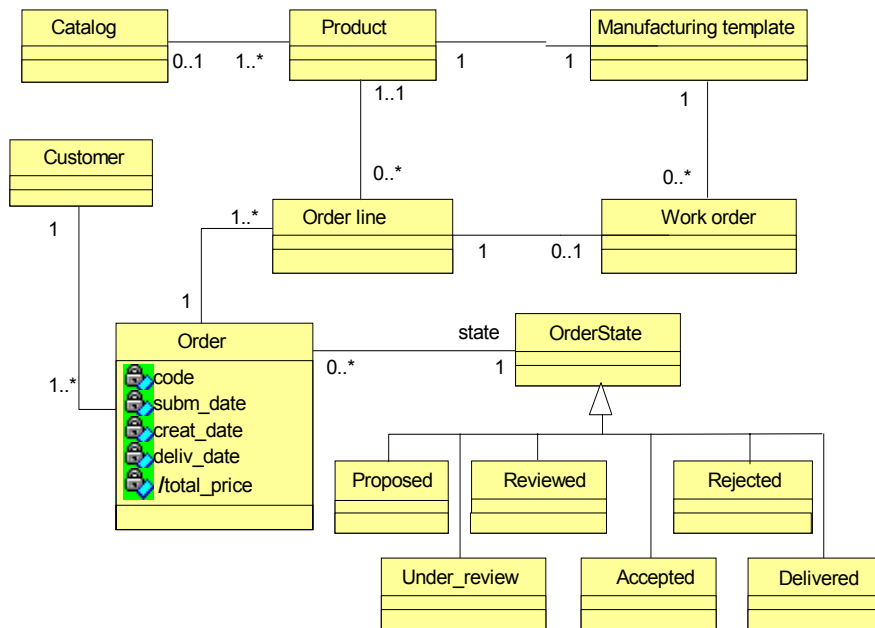
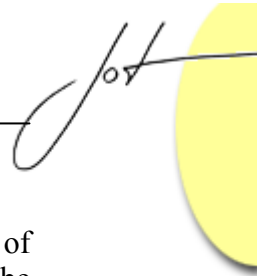


Fig. 11. Design class model



Once the definition of the design class model is stable, we have to identify the subset of persistent classes, which will conform to the *database schema*. This schema can be represented by means of a UML class diagram in which it is possible to include all the specific features of database design. With this aim, we made use of IDEA [Ceri Fraternali 1997], an analysis and design method for information systems with intensive use of data. In particular we used its object diagram, with its semantic expressiveness that can be adapted to the UML context, following the schema given in figure 12.

IDEA Method	UML
Built-in Constraints: <ul style="list-style-type: none">- Primary key (●)- Not null value (NN)- Unique value (U)	A <i>constraint</i> after <i>name:type</i> of attribute: {*} {NN} {U}
Referential Integrity Constraints: <ul style="list-style-type: none">- Cascade Deletion- Restrict (an arrow labeled with <i>CD</i> or <i>R</i>)	A <i>dependency</i> (dashed arrow), from the referencing class to the referenced one, and labeled with a <i>constraint</i> {CD} or {R}
Generic Integrity Constraints: <ul style="list-style-type: none">- <i>Immediate</i> or <i>Deferred</i>,- <i>Static</i> or <i>Dynamic</i>,- <i>Targeted</i> or <i>Untargeted</i>	A <i>note</i> including <i>constraints</i> (<i>immediate</i> or <i>deferred</i> ; <i>dynamic</i> or <i>static</i>) and the text of the integrity constraint, written in either OCL or pseudo-language The note can be linked to a class (<i>targeted</i>) or a group of classes (<i>untargeted</i>)

Fig. 12. Extract of the mapping from IDEA notation to UML

Figure 13 shows the database schema obtained starting from the design class model (see figure 11), which includes some referential integrity constraints (for instance, *when an order is deleted all the order-lines involved have to be deleted too, and a customer cannot be deleted while one of his orders still exists*), as well as other generic integrity constraints.

In contrast to our proposal, the TMA information model only includes those domain classes of the system that will make up the database schema.

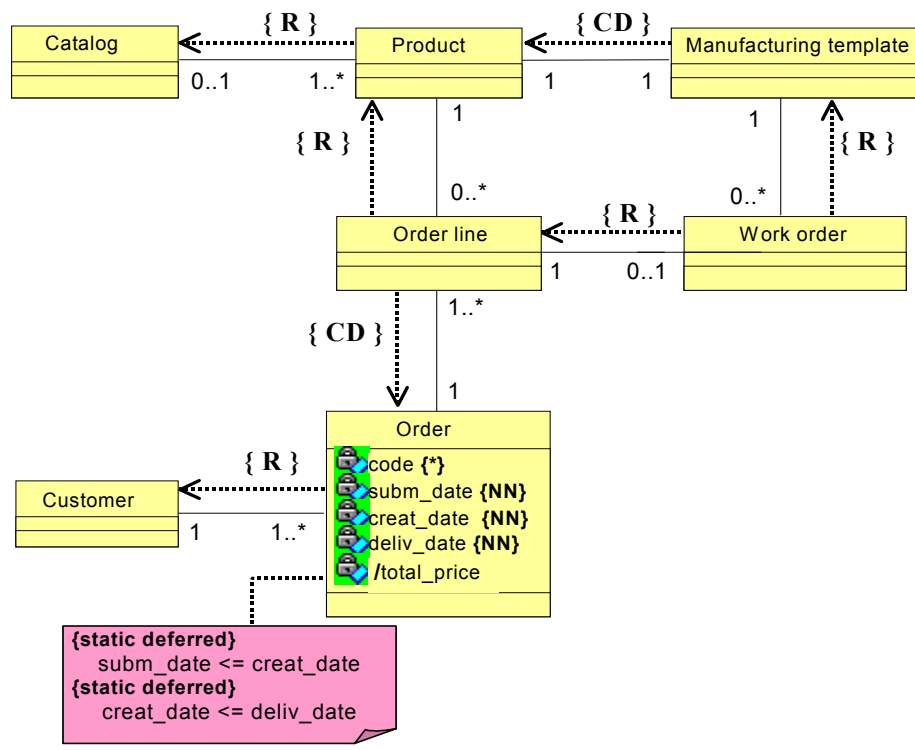
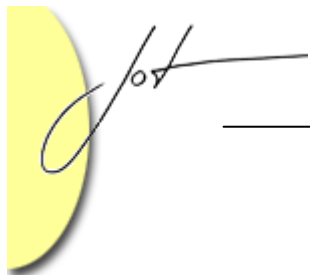
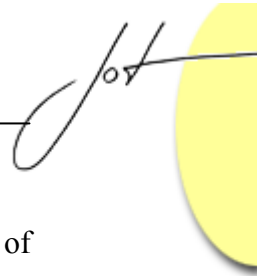


Fig. 13. A partial view of the database schema

From the business model it is also possible to identify some classes whose behavior depends on a rich set of reachable states. In this case, it could be interesting to define a state machine for them, represented by means of a UML *statechart diagram*. These classes are easily detected in the process diagrams since they correspond to information objects labeled with several states. In our running example, *Order* would be a candidate for building a state machine that shows the states of an order (*proposed*, *under_review*, *reviewed*, *accepted* and *rejected*) and the events concerning the changes from one state to another.

5 TOOL MODEL

TMA tool model describes how the users access the information, through a specific software tool (user environment) which supports the tasks of each role. These tasks are extracted from the process view in the enterprise model. First of all, a textual detailed description is written for every task. Next, they are graphically described by means of a scenario view or process view. These diagrams show the interaction between the tools and information services to obtain and modify the information stored in the database. Consequently, a tool model diagram is formed by three types of role: i) one role representing the user of the system (detected in the enterprise model) that will perform the task; ii) one role representing the specific tool that the user will use to carry out the



task, and iii) a set of information service roles (databases). In this way, the interface of the classes that model the information objects is obtained.

The tool model is translated into UML by building a sequence diagram which will show the interaction needed to provide the functionality associated to the system use case. It should be observed that, in general, every activity in the process diagram of the business model is transformed into a use case, as we explained in section 3.3.

In order to build the tool model, both the design classes and the templates of the system use cases are needed. It is an iterative process which, at first, only takes into account the basic flow of each use case that is described through a sequence diagram. Descriptions of the exceptional and alternative paths are postponed until later iterations.

As figure 14 shows, the instances which appear in the sequence diagram match the types of role in the TMA tool model, since this diagram includes an instance of an actor (*Catalog manager*), an instance of the tool used by the actor (*Launch Manufacturing GUI*), and the set of objects involved in the interaction. These objects might be instances of persistent classes, such as *Order* and *Product*, or transient objects such as *OrderManager*. However, in the TMA tool model, only the objects that provide the access to the information services are considered.

At this point it is necessary to apply the heuristics about the assignment of responsibilities to the classes [Larman 1998] and the utilization of design patterns [Gamma et al. 1994] for defining the most appropriate interaction in order to achieve the expected software quality.

The diagram in figure 14 shows the interaction needed to realize the use case *Launch Manufacturing*. As we can see, a work order for every product is created. The tool *Launch Manufacturing GUI* interacts with the system object *OrderManager*, a controller object whose class was not included in the conceptual model. This controller, in turn, interacts with instances of classes already present in the conceptual model.

Of course, the classes of the new identified objects will be included in the design class model, as well as their associations. This is the case of *OrderManager* in our example.

The sequence diagrams of the tool model show the messages exchanged between the system objects to realize the related use case, thus these messages are services which must be provided by the corresponding classes. This allows us to refine the *design class model*, by adding the suitable methods to every class (which has only been characterized by its attributes and constraints, so far).

Moreover, through the creation of the tool model, the specific operations requested by each user (the messages sent to the system) are identified. Thus, it is possible to detect the different commands that should be available for the users. This is very useful in designing the user interfaces.

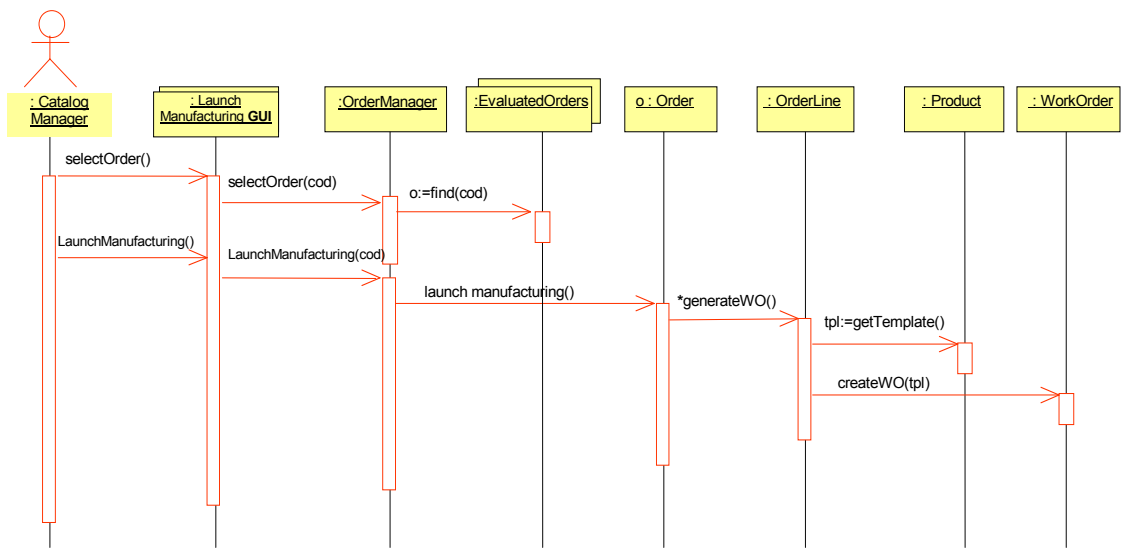
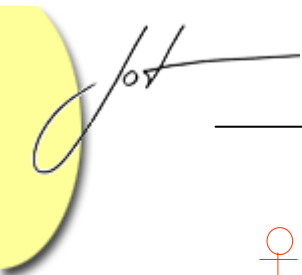


Fig. 14. Sequence diagram for *Launch manufacturing* system use case

6 CONCLUSIONS

In this paper we have presented how three-model architecture, a process for the development of business information systems, defined in the context of the OOram method, can be adapted to UML. Figure 15 shows the mapping between the TMA models and our proposal.

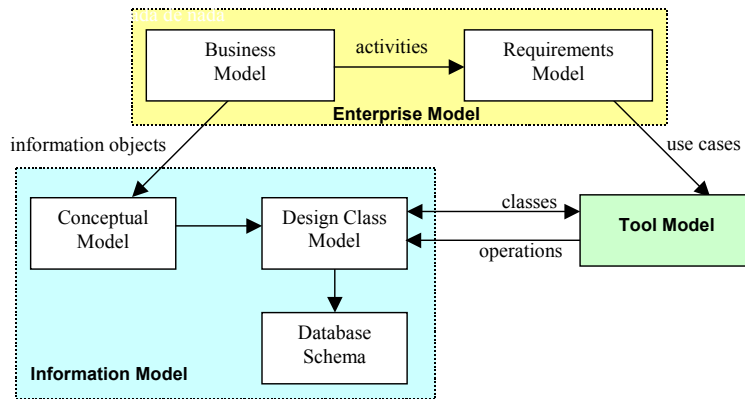
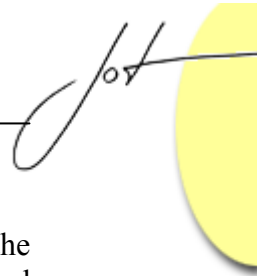


Fig. 15. Three model architecture for UML

- *TMA enterprise model* is translated into UML by means of two models because use cases have to be included in order to elicit the functional requirements of the software system. In this way, we have connected the *business model* (with an identical purpose to the TMA enterprise model) to the *requirements model*, given that each activity of a business process can be the origin of a system use case.



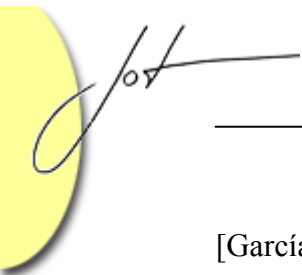
- *TMA information model* is mapped into UML by using three models: i) the *conceptual model*, describing domain concepts, whose initial version is created from the information objects extracted from the *business model*; ii) the *design class model*, including the system classes, and created in parallel with the *tool model*; and iii) the *database schema*, including only persistent classes, and enriched with specific features of database design extracted from the IDEA method.
- *TMA tool model* describes how a system use case, defined in the *requirements model*, is realized by an interaction between instances of classes included in the *design class model*; the protocol of these classes must include methods for each message that appears in the interaction.

The result of the transformation of TMA into UML described in this paper is the definition of an iterative, use case-driven UML process for modeling business information systems organized by a client-server, three-tier architecture.

We think that this process is well-suited to developing object-oriented applications with persistent objects in a database system. The mapping from our proposal into the client-server three-tier architecture is established by identifying the user-interface classes in the tool model as the presentation tier, the classes in the design model as the logical tier, and the classes in the database schema as the persistence tier.

REFERENCES

- [Booch et al.99] G. Booch, J. Rumbaugh, and I. Jacobson: The Unified Modeling Language User Guide, Addison-Wesley Longman, Inc., Reading, Massachusetts, 1999.
- [Ceri Fraternali97] S. Ceri, and P. Fraternali: Designing Database Applications With Objects and Rules. The IDEA Methodology, Addison-Wesley Longman, Inc., Harlow, England, 1997.
- [Cockburn00] A. Cockburn: Writing Effective Use Cases (pre-pub. Draft 3), Addison-Wesley Longman, Inc., Reading, Massachusetts, 2000.
- [Coleman98] D. Coleman: A Use Case Template: Draft for discussion, 1998, http://www.cs.colorado.edu/users/kena/classes/6448/s02/links/hp_use_case_template.pdf
- [Fowler98] M. Fowler: Use and Abuse Cases, Distributed Computing, <http://www.martinfowler.com/articles/abuse.pdf>, April 1998.
- [Gamma et al.94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman, Inc., Reading, Massachusetts, 1994.

- 
-
- [García Molina et al.00] J. García Molina, M.J. Ortín, B. Moros, J. Nicolás, and A. Toval: Towards Use Case and Conceptual Models through Business Modeling, Proceedings of 19th International Conference on Conceptual Modeling (ER2000), Salt Lake City, USA, October 2000.
- [CSL-NIST93] Computer Systems Laboratory, National Institute of Standards and Technology: Integration Definition for Function Modeling, FIPS Pub. 183, December 21, 1993.
- [Jacobson et al.99] I. Jacobson, G. Booch, and J. Rumbaugh: The Unified Software Development Process, Addison-Wesley Longman, Inc., Reading, Massachusetts, 1999.
- [Larman98] C. Larman, Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.
- [OMG00] OMG Revision Issues. Issues for UML 1.4 Revision Task Force. Issue 2837: role concept in UML remains rather vague.
<http://cgi.omg.org/issues/issue2837.txt>
- [Ortín et al.98] M.J. Ortín, J. García Molina, A. Martínez, and A. Pellicer: Combining OOram and IDEA for Information Systems Modeling, Technical Report TR-01-00, Facultad de Informática, Universidad de Murcia, December 1998.
- [Ortín García-Molina99] M.J. Ortín, and J. García Molina: Role-Based Modeling with UML, Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, Cáceres, Spain, 1999.
- [Pols97] A. Pols: Use Case Rules of Thumb: Guidelines and lessons learned, Fusion Newsletter. Hewlett-Packard Laboratories, Vol.5.1, February 1997.
- [Reenskaug96] T. Reenskaug: Working with Objects: the OOram Software Engineering Method, Manning Publications, Greenwich, England, 1996.
- [Reenskaug97] T. Reenskaug: Working with Objects: a Three-Model Architecture for the Analysis of Information Systems, Journal of Object Oriented Programming, vol. 10 no. 2, 1997, pp. 22-30.



About the authors



Jesús García Molina is a professor in the Department of Computer Science at Universidad de Murcia, Spain since 1984. His research interest include object-oriented modeling, requirement specification, software architecture and object databases. He received his Degree and PhD in Chemistry-Physics from the University of Murcia in 1983 and 1987, respectively. He is currently Dean of the Faculty of Computer Science. He can be reached at jmolina@um.es.



María José Ortín is a lecturer in databases at the Universidad de Murcia since 1996, where she obtained his BSc degree in Computer Science in 1994. Her current research interests include object-oriented modeling, software development methods and object databases. She can be reached at mjortin@um.es.



Begoña Moros is an lecturer in object oriented programming at Universidad de Murcia since 1998, where she obtained her BSc degree in Computer Science in 1997. Her research interest include prototyping environment for UML, software development methods and requirements engineering. She is currently developing her PHD in the requirements reuse field. She can be reached at bmoros@um.es.



Joaquín Nicolás is a lecturer in software engineering at the Universidad de Murcia since 1996, where he obtained his BSc degree in Computer Science in 1994. His current research interests include requirements engineering and software reuse at the requirements level. He can be reached at jnr@um.es.