Experiences with the use of MERODE in the development of a Web Based Application

Karina Chong[1], Verónica Macías[1], Monique Snoeck[2]
[1] Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral (ESPOL)
Campus Gustavo Galindo, Km. 30.5 vía Perimetral
Apartado 09-01-5863, Guayaquil, Ecuador
[2] Faculty of Business and Economics
Catholic University of Leuven
Naamsestraat 69
B-3000, Leuven, Belgium
kchong@fiec.espol.edu.ec, vmacias@espol.edu.ec, Monique.Snoeck@econ.kuleuven.be

Resumen

Este artículo presenta el reporte de la experiencia de usar MERODE como método de modelamiento del negocio para el desarrollo de una aplicación web. MERODE posee algunas ventajas tales como el incremento de los atributos de flexibilidad y mantenibilidad de las aplicaciones construidas siguiendo el método y la posibilidad de realizar la verificación y validación interna del modelo de la aplicación de una forma automatizada. La aplicación desarrollada tiene como funcionalidades generales la administración de la organización de eventos y la administración de la información de un grupo de investigación. La aplicación fue monitoreada para verificar sus atributos de flexibilidad y mantenibilidad y se verificó la factibilidad de usar el método en el proceso de desarrollo. Los resultados obtenidos mostraron que la aplicación desarrollada tenía unos atributos de flexibilidad y mantenibilidad muy satisfactorios.

Palabras Claves: *MERODE, modelamiento, dominio, negocio, MDD.*

Abstract

This article presents an experience report on using MERODE as the business modeling method for the development of a web application. MERODE has several advantages as improving the flexibility and maintainability of applications and the possibility of doing automated verification and validation on the internal consistency of the model. The application's main functionalities were managing the organisation of events and managing the general information of a research group. The developed application was monitored in order to check its flexibility and maintainability and also to verify the feasibility of using the method. The results show that in fact the flexibility and maintainability of the application were satisfactory.

1. Introduction

A domain model (aka. business model) is a representation of a business domain from a particular point of view. As any model, the purpose of a domain model is tackling the complexity of the whole by means of showing its relevant components and hiding unimportant issues. The importance of a component of the domain depends on the perspective that we are using to analyze it; and that is why we can have several views of the same domain.

Model Driven Development (MDD) is an approach to develop software applications, based on domain models. There are several methods and methodologies that follow these principles. One of them is MERODE [1], a method whose application is the main focus of this article.

MERODE stands for *Model driven, Existence-dependency Relationship, Object oriented Development*. It is an Object Oriented Enterprise (business or domain) Modeling method, and it was designed by Monique Snoeck, Guido Dedene, Maurice Verhelst and Anne-Marie Depuydt at the Department of Applied Economic Sciences of the Catholic University of Leuven, Belgium [1].

MERODE has been applied and tested in a large number of companies in Belgium and Netherlands; but it is not very known in Latin America.

One of the disadvantages of some MDD methods is the use of UML as modeling language, because it has a lack of well-defined semantics [2]. MERODE on the other hand proposes a formally defined modeling syntax that follows the *single model principle*, based on the conception of a single model, for which different views are constructed. It also defines consistency rules between views [3], and provides means for implementing automatic verification and validation of consistency between the views. MERODE also follows a natural layered architecture, grouping specifications according to the aspect they originate from [4]. Layered architectures have been employed for improving system modularity before, but MERODE goes a step further defining what kind of objects, should be in each layer. Figure 1.

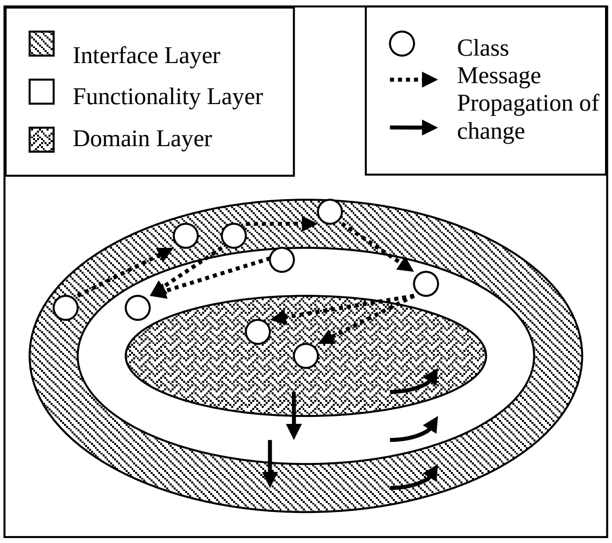

Figure 1. MERODE architecture Layers and change propagation

In the inner layer called the *Enterprise Layer*, domain objects are defined. They are independent from those in the higher layers, and communication is constrained to same-layer objects.

The middle layer is called the *Functionality Layer*. Objects in this layer represent the input and output services that the application will offer. They communicate between them and those in the Enterprise Layer. Input services are requirements of entering, or changing information; and output services are information requirements.

The User Interface Layer, is the outer layer. Objects defined here represent the Graphical User Interface (GUI) components of the application, and communicate between them and with objects in the Functionality Layer.

The order of the layers have the purpose of improving the flexibility and maintainability of the system. This is achieved by locating the objects that are less subject to changes, in the inner layer. Therefore, changes to objects prone to change (usually GUI objects) propagate less than changes to objects that remain constant during their life-cycle (usually domain objects) [4].

Another relevant characteristic of MERODE is the use of events as a connection point between Enterprise (Bussiness Layer) and Functionality Layers (Information System Layer). Other object-oriented methods usually subordinate the events to the objects, and represent the object interaction by means of sequence or collaboration diagrams that usually have long sequences of methods invocation [5], in a *message-passing* like approach. The problem with this procedure is that in the presence of very long sequences of methods, the flexibility and maintainability of applications decrease. The *event-driven* approach used by MERODE, gives to events the same relevance than an object, and recognizes them as a fundamental part of the structure or experience [5].

An object business event is an atomic unit that represents a particular action that happens in the real world; so object interactions now is modeled by defining which objects are concurrently involved in a given event [5]. The order the different objects execute an event is determined by an *event-dispatcher*. This element is responsible for sending events notifications to all participating components, coordinates the responses (success or failure) and provides the triggering component with feedback about the execution status [6]. As a result the long sequences of method invocations are replaced by standardized interaction patterns.

Given the aforementioned advantages we decided to implement an application using the MERODE method, and evaluate the final product and the process to develop it.

This article is organized as follows. The second section introduces the application developed using MERODE for the analysis phase. Section 3 presents the metrics that we defined in order to evaluate the final product and the development process. Results of our measurements are shown and discussed in section 4. Finally, conclusions and recommendations of this work are presented in section 5.

2. The application

In order to apply the method, we developed a web application for our use in organizing conferences and publishing the research work carried out within our group.

As part of the activities of our research group we organize conferences, workshops and courses for students and industrial partners. Hence, the application to be developed should provide us with the required functionality for publishing relevant data, registering assistants in specific events, for sending email notifications of incoming events to the registered users, for sending email notifications of deadlines for registering or paying, for the reporting and controlling of payments, for managing documents presented by attendants in order to have discounts (student card, IEEE member card, etc.), for checking the attendance to the events and for registering the material delivered to the assistants.

As a research group, we also need to publish general information on the group (objectives, members, etc.) the ongoing projects, past projects and articles produced.

We decided to develop it as a web application, because we want the general public to access the information. The use of roles allows giving some users the responsibility of managing the information, and allows having a general public role, which is intended for users who can see the information and register to events. Roles can be overlapping, so there is the possibility that an administrator user sees the same information than the general public can see.

The application was developed using an object oriented programming language, but according to the authors, MERODE can be used to model the domain for applications that would be implemented using other programming paradigms [4]. We choose to use Java because it is a multi-platform and very robust language, and because we thought it would be more natural to use an object oriented approach in the implementation as we used the same approach for the domain model.

3. Application and Process Development Metrics used

The following metrics were taken during the development and beta-releases of the application:

- Development time
- Differences between estimated and actual development time
- Number of changes to the application
- Number of versions of the Software Requirements Specification (SRS) Document
- Number of versions of the different "views" in the model

The development time, and the differences between the estimated time and the actual time were taken in order to verify if MERODE affects the scheduled time and also to study the feasibility of using it in a commercial environment where time to develop an application is usually very restricted. The methodologies that were used to estimate the development time were Function Points [7] and COCOMO [8].

The number of changes made in the application were collected in order to estimate the quality of our domain specification, and to show the frequency of each type of change.

Changes were classified as: Conceptual changes, Form changes and Error Changes. Conceptual changes were the ones that cause a modification in the Business layer. Form changes affect the Functionality layer and the User Interface layer, but were requested by the final user. Error changes were demanded after an application fault, but did not affect the Business layer.

Numbers of versions of SRS were taken to measure how much the requirements were changing as the application was being developed.

Finally version numbers of the different views in the model were taken to show how much the domain model changed while the application was being developed.

The views of the model defined by MERODE are the Existence-Dependency Graph (EDG), the Object-Event Table (OET) and the Finite State Machine (FSM) graph. The EDG is a data model that shows the domain objects and their relationships, the OET shows the relations between objects and events, and how an event affects (creating, modifying or ending) each object, and the FSM shows the life-cycle states of an object, and the sequence constraints that an object imposes on events[4].

4. Measurement Results

Table 1 shows the estimated and the actual time for each application's component in working days, the difference between the actual and the estimated time, and the proportion of that difference against the estimated time.

Table 1. Estimated and Actual Developing Time

Com-ponent	Estima-ted time (working days)	Actual time (working days)	Differen-ce between actual and estimated time	% of the diffe-rence
Subs-cription	50	74	24	48%
Events	40	54	14	35%
Publi-cations	20	31	11	55%
Mana-gement	20	25	5	25%
Total	130	184	54	42%

The data shows a considerable difference between the estimated and the actual development time for each one of the components. The larger differences correspond to components that were developed first (Publications and Subscriptions). This effect could be the result of the developer inexperience both in estimation as in the use of the method. The software was developed by an undergraduate thesis student who also was in charge of performing time development estimation. She had taken a course on MERODE, but did not have much experience with the use of the programming language and the platform. As a result the last component that was developed had less difference between the real and the estimated development time.

The difference also may be explained by the time that a developer has to devote in order to follow the MERODE architecture strictly. For example if you have 2 domain objects that have a existence-dependency relationship between them and just the basic creating and ending events for each of them, in order to follow the MERODE layered architecture the implementation would have 2 classes for each object, 4 classes for the business events, 4 classes for the functionality layer and at least 1 class for the GUI layer. Our project had 29 objects, so it represents a considerable development effort.

Then, as the estimation-time was calculated by someone with little experience in doing this and who also had but little experience with developing web applications, it is too difficult to know in how far a bad estimation and in how far a possible effect of the use of the methodology, were the cause of the difference between the estimated developing time and the actual developing time, and therefore we can't say anything about how the application of MERODE affects the developing time.

Table 2 shows the amount of changes that were introduced in the beta-releases of the model, grouped by type.

Table 2. Number of changes

Components	**Change Type**		
	Conceptual	**Form**	**Error**
Suscriptions	0	20	4
Events	1	10	6
Publications	0	4	2
Management	0	2	2
Total	1	36	14
Proportion (%)	1.96%	70.59%	27.45%

One of the advantages of MERODE is the improvement in the quality of the domain specification. If we have a domain model that is changing very frequently, it could mean either you are working in a very difficult business domain, or that your domain model did not reflect the real business domain. Our application was not in a complex business domain, and as we can see in Table 2, we have just 1 change in the domain model during the 7 beta-releases; so we can say that the domain model we constructed was acceptably stable. According to our results functionality and user interface objects have a higher rate (70.59% between both) of change than business objects (1.96%). Error changes have also a lower rate (27.45%) than form changes, and that also is a good indicator of the high quality of the modeling specifications. Figure 2.

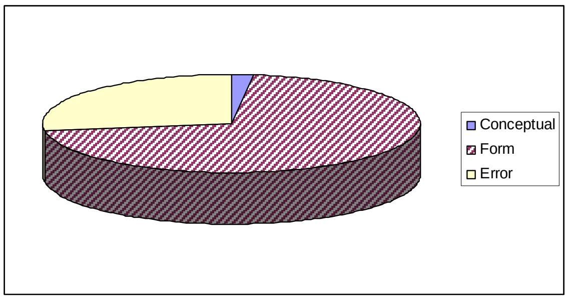

Figure 2. Changes types Proportion.

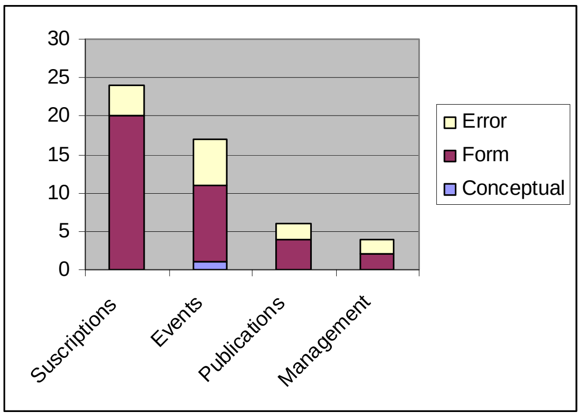

Figure 3. Changes types by components

As depicted in Figure 3, the Subscription component had more total changes, which is logical since it was the first component developed. The last components show fewer changes; particularly form changes were greatly reduced mostly because the application's GUI was already established and the developer had experienced with the GUI programming libraries used in the project.

In Table 3 we present the numbers of versions of SRS document, EDG, OET and FSM views produced during the 7 beta-releases.

Table 3. Number of versions by document and model views

	Number of versions
SRS document	4
Existence Dependency Graph EDG	15
Object Event Table OET	13
Finite State Machine FSM	2

The EDG view had more versions than the rest, which might appear as inconsistent with the results showed in Table 2, because we said that we have just 1 change in the model; but we need to say that these different versions of the EDG were the result of adding object attributes, and do not represent adding or deleting business objects, or changes in the objects relationships; consequently they were not considered in the number of conceptual changes.

5. Conclusions

The experience we had in using MERODE to develop a web base application produces the following conclusions:

- It was feasible to use MERODE as the analysis method to develop a web application.
- The attributes of flexibility and maintainability of the product were satisfactory, even more than initially expected.
- The development process to follow MERODE architecture in a manual way takes considerable effort in man-hours.

These conclusions are in line with earlier evaluations of the method [9].

In order to use MERODE in a commercial environment would be necessary the use of a generation code tool based on the model. Currently, a code generation tool is available to generate a prototype application in Java [10]. The code is generated by using the AndroMDA environment and uses the Hibernate Framework for the persistence layer and the session beans implementing the event handling layer. In the future, we plan the extension of the code generator with an option for a web-based user interface.

12. References

[1] MERODE web site. Last visited 28/09/2007. Available in http://merode.econ.kuleuven.ac.be/

[2] France R., Ghosh S., and Dinh-Trong T. "Model-Driven Development Using UML 2.0: Promises and Pitfalls", *IEEE Computer Magazine* 39, no 2, 2006, pp. 59-66

[3] Snoeck M., Michels C., and Dedene G., "Consistency by Construction: the case of MERODE", *Conceptual Modeling for Novel Application Domains, ER 2003 Workshops ECOMO, IWCMQ, AOIS, and XSDM Proceedings*, October 2003, pp. 105-117

[4] Snoeck M., Dedene G., Verhelst M. and Anne-Marie Depuydt, *Object Oriented Enterprise Modelling with MERODE*, Leuven University Press, 1999, pp. 3-21

[5] Michiels C., Snoeck M., Lemahieu W., Goethals F., and Dedene G., "A Layered Architecture Sustaining Model Driven and Event Driven Software Development", *Andrei Ershov International Conference "Perspectives of Systems Informatics" Proceedings*, Lecture Notes in Computer Science, 2003

[6] Snoeck M, Lemahieu W., Michiels C., and Dedene G., "Event-based Software Architectures", *Object-Oriented Information Systems, 9th International Conference, OOIS 2003 Proceedings*, September 2003, Lecture Notes in Computer Science, pp. 107-117

[7] Garmus D., and Herron D., *Function Point Analysis: Measurement Practices for Successful Software Projects*, Addison-Wesley, 2000

[8] Boehm B., Abts C., Winsor A., Chulani S., Bradford K., Horowitz E., Madachy R., Reifer D., and Steece B., *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000

[9] Snoeck Monique, Dedene Guido, Experiences with Object-Oriented Model-driven development, *Proceedings of the STEP'97 conference*, London, July 1997.

[10] Monsieur G, Snoeck M, Haesen R, Lemahieu W, 2006, PIM to PSM transformations for an event driven architecture in an educational tool, *European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA)*, Bilbao (Spain), July 11, pp. 49 - 64.