19

A Structured Object-Oriented View on Systems Modeling

G. Wu and G. Dedene

Section of Information Systems, Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium

Abstract

Interest in object-oriented methods has been rapidly increasing, as software developers and management demand on producing high quality and productivity softwares. Special features of object-oriented developments can really help to achieve high quality softwares. While this does not mean that object-oriented approaches will automatically obtain high quality software, the special features of object-oriented methods can only guarantee a certain degree of quality. To fully explore the benefit of object-oriented paradigm in software development, extra concepts should be introduced in order to achieve higher quality and productivity. The *structured object-oriented view*(SOOV) on systems modeling to be discussed in this paper is such a concept that can be used to further exploit the power of object-oriented methods to ensure higher quality, especially for large and complex systems. The structured view defines an overall inherent structure on objects and systems. Real world objects and systems are structurally decomposed into several logically and inherently related parts. After decomposition common parts could be discovered and factored out early in the modeling stage. The whole software development process is guided by this view in a quite neat and consistent manner.

Keyword Codes: D.2.1;D.2.2;H.1.0
Keywords: Model-driven, abstract model, real world object, model object, object main, object aspect, structured object-oriented view, object deposition and factoring.

1. INTRODUCTION

The structured object-oriented view on system modeling is part of M.E.R.O.DE. (Model-driven Entity Relationship Object-oriented DEvelopment), a methodology being developed at the faculty of applied economics science of K.U.Leuven. The methodology in fact covers the whole software development cycle from modeling to implementation[1-7]. The objective of this paper is to show an important part of the M.E.R.O.DE. method, that is, the structured object-oriented view on systems modeling. Discussions are based on this method, but applicable to general OOAD. Rather than starting from information requirements (Requirements-driven approach) for systems development, which is indeed a very volatile starting point, development starts with modeling business reality (Model-driven approach) of that part of the real world for the system under development. It turns out to be a much more stable starting point than the system information requirements[4].

The M.E.R.O.DE. development emphasizes on creating an *abstract or business model* of reality. Later on an information model is built, in the design stage, around this abstract model of reality. The abstract model is the basis and source to supply information to end users. Business modeling in M.E.R.O.DE. models the business reality by *business objects* or *model objects*, *business events* and *business rules* concerning the sequence of these events[4],[5]. Business oriented terms or names are used so that end-users can understand and participate in

M. Lee et al. (eds.), *Software Quality and Productivity*

the modeling process. To construct abstract model, modelers first find out relevant *real world objects* in systems under development (problem domain). Real world objects are denoted by R objects. Relationships between R objects are described by ER model. A set of attributes can be identified for each R object and represents the object's static state.

Concerned business events in real world system need also to be identified. For example, "manager gets promoted" and "employee starts training period" are two events in personnel management system. The happening of an event usually causes changes to R objects in a system, for instance, the happening of event "employee joins a project" will cause change to employee's state, i.e., the number of projects that the employee is currently participating in. Similarly, the event affects also project's state, i.e., the number of employees who are participating in the project. When the happening of an event causes changes to the state of a R object, it is said that the event *involves* this R object and an event method is defined in it.

In general, an event may involve several R objects, for instance, the "employee joins a project" event involves both employee and project objects. Events actually represent the *behavior* of R objects and an event sometimes represents the behavior of several R objects, for example, "employee joins a project" event is the behavior of both employee and project objects. A system's behavior is then represented by the behavior of R objects in the system. Behavior of R objects in reality usually can not happen freely. Business rules concerning the sequence or order of these objects' behavior exist, such as that "employee becomes a regular employee" event must follow the event "employee passes trail use period". These business rules are also called *behavior constraints.*

2. BEHAVIOR COMPLEXITY OF REAL WORLD OBJECTS

A simple personnel management example in this paper is used to illustrate the main ideas. Three R objects in this system are of interest to modelers, they are Manager, Employee and Project. Relationships between these R objects are shown in figure 1.

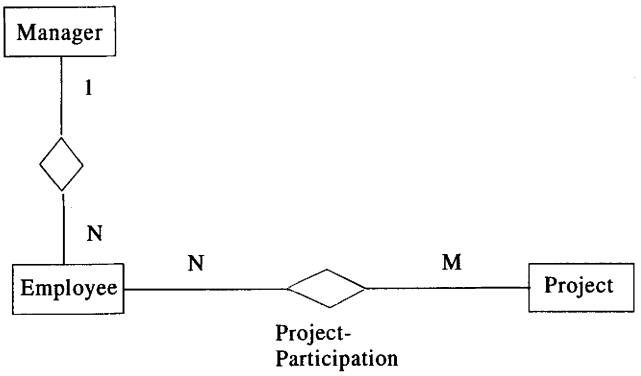

Figure 1. ER model of personnel management example

Concerned events and R objects are given in table 1, the R object/event table. A "*" in the table indicates that the event in the row involves the R object in the corresponding column. An object in reality tends to have a lot of behavior. The employee object in the

example has the following behavior: "e-crt", e-start-trial", "e-end-trial", "e-regular", "e-trial-fail", "e-promote", "e-leave", "pp-join", "pp-start-train", "pp-end-train", "pp-disjoin", "e-join-m" and "e-leave-m". Of course employee may have a lot of other behavior which is not concerned or not currently considered in this system. Now considering all behavior of a R object, not only that in a particular system, could we get an inherent structure on it? Simply putting events to R objects can not help modelers to better understand objects' internal structure.

Table 1. R object/event table for personnel management example

Event types	R Object types Manager	Employee	Project
m-crt (new manager)	*		
m-promote (manager gets promoted)	*		
m-end (manager ends)	*		
e-crt (new employee)		*	
e-start-trial (employee starts trial use)		*	
e-end-trial (employee ends trial use)		*	
e-regular (employee becomesregular employee)		*	
e-trial-fail(employee trial use failed)		*	
e-promote(employee gets promoted)		*	
e-leave(employee leaves company)		*	
p-crt (new project)			*
p-end(project ends)			*
pp-join(employee joins project)		*	*
pp-start-train(employee starts training)		*	*
pp-end-train (employee ends training)		*	*
pp-disjoin(employee leaves project)		*	*
e-join-m(employee becomes subordinate)	*	*	
e-leave-m(employee leaves manager)	*	*	

3. STRUCTURED OBJECT-ORIENTED VIEW ON R OBJECTS

This section describes the R object's internal structure and the inter-R objects' structure impact on the state and behavior of objects. First the *R objects' internal structure* is examined. In fact the behavior and states of a R object can be classified or grouped into several logical related groups based on its relationships with other R objects in the ER model, each group is characterized by a relationship. An employee, for instance, has behavior group related to manager (i.e., "e-join-m" and "e-leave-m"), behavior group related to project (i.e., "pp-join", "pp-start-train. "pp-end-train" and "pp-disjoin") and behavior group related to no relationship (i.e., "e-crt", "e-start-trail", "e-end-trail", "e-regular", "e-trail-fail", "e-promote" and "e-leave"). The behavior of a R object now can be partitioned into several disjoined event sets. If the system under development is interested in all potential behavior of a R object in reality, a complete partition of the object's behavior can be obtained. The R object in this case is exactly the same as the one in reality. Similarly a R object's state (attributes) can also be partitioned into several disjoined attribute sets based on relationships.

The *inter-R objects' structure*, a structured view on system, is defined as follows: a R object's behavior and state related to a relationship are actually common to all R objects participating in this relationship, they are the *common part* of these R objects. For example, an employee object's behavior and state related to "project-participation" relationship are common to both employee and project objects, they represent the common part of employee and project objects. To describe these structures, the following concepts are introduced.

Definition 1. A R object's aspect: those parts of a real world object's behavior (events) and state (attributes) which are related to the same relationship, are grouped together and called one aspect of the real world object.

Definition 2. A R object's main: those parts of a real world object's behavior and state which do not concern any relationship in which the object participates are grouped together and called the main of the object.

Note that a R object may have many aspects and a specific system is interested in only some of them. Each aspect contains a set of inherently and logically related events and attributes of a R object and represents a logical unit within the R object. It can be seen that this analysis and study of real world objects' behavior and states can assist modelers to obtain the structure within a R object and the structure among the different related R objects.

4. MODELING REAL WORLD OBJECTS AND SYSTEMS

Real world objects and systems have to be specified by M objects in object model of M.E.R.O.DE., the specification domain objects. M objects stand for model objects. Two approaches can be used to model a R object in object model. One is to model a R object by one M object, the other is to model it by several M objects. The main problem of the first approach is the redundancy caused by specifying the common part (aspect) of two related R objects separately in both of them. This could cause problems of inflexibility and possible inconsistency. The basic problem of the second approach is the loss of R objects in object model, that is, modelers can no longer find R objects in object model. The structured view on R objects can help us to solve these problems, namely to build a bridge between R objects and M objects, and at the same time provide a way to factor out the common parts of R objects in a system to reduce the redundancy in specification. The second approach is better in this sense. The bridge will be discussed in section 7.

Based on the structured view on R objects discussed above, *common parts of R objects are represented by their aspects.* Now factoring out common parts means to factor out aspects of R objects. To do so, each aspect is separated from its related R objects, and is considered or treated as one M object in object model. The mains of R objects are treated also as M objects. The second approach is used here. Two definitions are introduced to describe these more formally.

Definition 3. Aspect object (common part): the model (M) object used to describe an aspect of a R object is called an aspect object.

Definition 4. Main object (non-common part): the model (M) object used to describe the main of a R object is called a main object.

It is clear that there is only one main object for each R object and vice versa. The main object represents the corresponding R object's "identity" in object model, but not completely because some aspects have been taken out and specified in aspect objects. It is also obvious that a R object may have several aspects, therefore the main object is associated with several aspect objects in object model. Aspect objects, in this case, correspond to relationship objects. The choice to use the name "aspect object" is simply because the name itself carries the desired meaning, that is, it represents the aspects of R objects, while the name "relationship object" does not.

All of the events involving R objects need to be transformed into events involving M objects, which can be described by a M object/event table similar to the above discussed R object/event table. M objects, in this case, are listed in columns instead of R objects. There are five M objects in the personnel management example, three main objects, i.e., employee, manager and project main objects, and two aspect objects, i.e., "manager-employee" and "project-participation" aspect objects.

It is important to notice that an event involving a M or R object can actually be classified into two types based on its functions in M or R objects. They are action and response events.

Informal definitions are given below. More formal definitions can be found in [7]. Action and response events of R objects will be discussed in section 7.

Definition 5. An event belongs to a M object: an event E_i is said to belong to an M object o_j if it is existence dependent on M object o_j and there does not exist another M object o_k such that E_i is also existence dependent on o_k and o_k is existence dependent on o_j.

Definition 6. Action event (behavior) of a M object: an event E_i is considered as an action event or the behavior of a M object when it belongs to the object. Example: "pp-join" is an action event of "project-participation" aspect object.

Definition 7. Response event of a M object: an event E_i is said to be a response event of a M object when it involves the object but does not belong to it. Example: "pp-join" is a response event in "employee" main object.

The objective to distinguish between these two types of events is that only the action events of a M object are considered to be its behavior, response events reflect only the impact of other M objects' action events (behavior) on this M object. The behavior constraints to be specified in object model are only specified on the action events of M objects. Definition of "an attribute belongs to a M object" is also given in [7].

5. GRAPHICAL REPRESENTATION OF THE STRUCTURES IN OBJECT MODEL

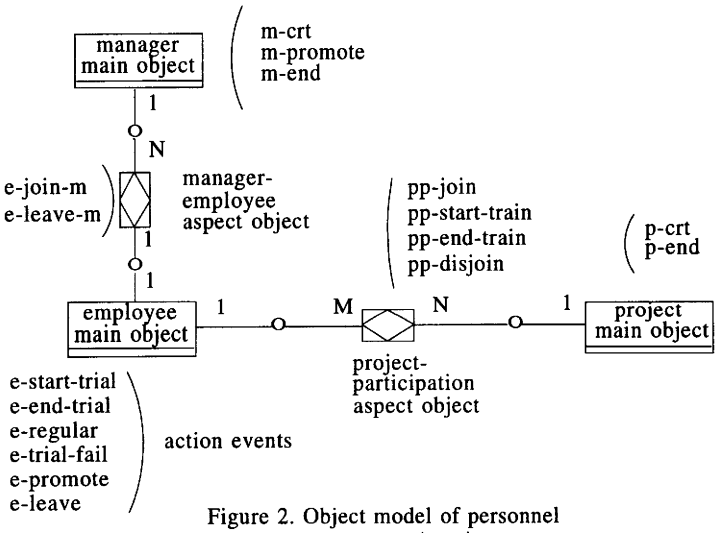

Figure 2. Object model of personnel management system

Object model can be obtained directly from ER model by a concrete procedure. Figure 2 is the object model derived from the ER model of personnel management system. The small circle in the diagram represents the relationship between model objects. There are five M objects in this example. A main object is represented by a box with double lines at its bottom side to show that it is different from the corresponding R object. For example, the employee main model object is different from the R employee object because part of R employee object is taken out and specified in project-participation aspect model object. An aspect object is represented by a box with a diamond within it. In object model, an aspect object is always existence dependent on the related main objects, for example, the "project-participation" aspect object is always existence dependent on main objects "project" and "employee".

Cardinalities of relationships between model objects are represented by the numbers on both sides of the small circles. Action events of the five M objects are assigned to and listed next to them based on definition 6. They are considered as the behavior of these M objects. Response events can also be added to these M objects based on definition 7 [7], they are not shown here in figure 2. Similarly attributes can be assigned to these M objects[7].

In figure 2 "project-participation" aspect object contains the common part of both employee and project objects in ER model, it is one aspect of employee and one aspect of project object as well. "manager-employee" aspect object is then common to both R manager and employee objects. Now it can be seen how the structures can help us to get more information from object model, which could otherwise be ignored.

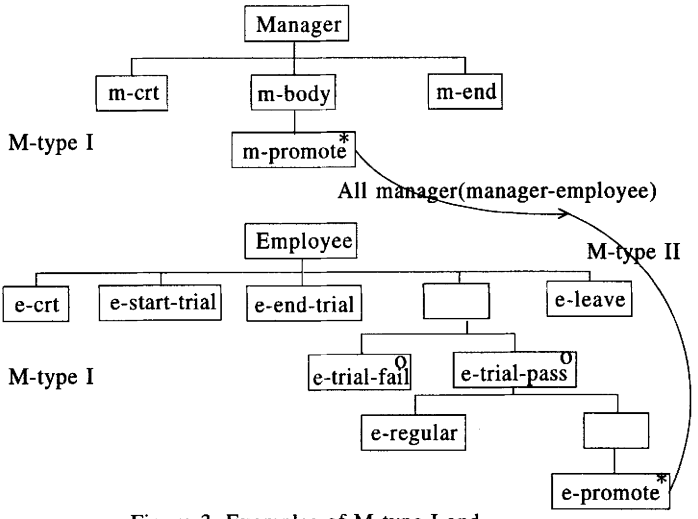

Figure 3. Examples of M-type I and M-type II constraints specification

6. BUSINESS RULES (BEHAVIOR CONSTRAINTS) SPECIFICATION

Besides of attributes and events specified for a model object, business rules in system should also be specified in object model. Recall that business rules are the sequence or order constraints on events, but specified only on action events of M objects in object model. There are two types of constraints, one is the *M object internal behavior constraints* (*M-type I*) which is specified on action events within a model object. This type of behavior constraints can be specified by techniques such as F.S.M., Petri-net, Action diagrams, J.S.D. entity life cycle diagrams and so on. In M.E.R.O.DE, the J.S.D. entity life cycle diagram approach[8] is used. The other is the *inter-M objects behavior constraints* (*M-type II*) which are placed on action events of different model objects. Specification language and graphical means are currently used in M.E.R.O.DE. to specify this type of constraints. Examples of employee and manager main objects' M-type I constraints and M-type II constraints between them are shown in figure 3.

For M object internal behavior constraints, sequence order of events is graphically represented by the left-to-right order of their appearances on the diagrams. "*" means that the event can happen zero or more times and "O" means that only one of the events with "O" can happen[6]. The employee's life cycle can be explained as the follows: an employee is created

when he or she enters the company, then starts trial use. If the employee passes the trial use, he or she becomes a regular employee and can get zero or more times of promotions till he or she finally leaves the company. If the employee fails the trial use, he or she has to leave the company. The language specification of the inter-M object behavior constraint "an employee's promotion must follow all of his or her managers' promotion" shown in figure 3 is: ALL Manager (manager-employee).m-promote $\rightarrow_{II}$ Employee.e-promote. It means that the promotion events of all managers related to an employee through relationship "manager-employee" must happen before the employee's promotion events. M object classes now include the following elements: State (attributes), action events (behavior) methods, response events (impact of other M objects) methods, behavior constraints on action events (M-type I and M-type II).

7. RECOVERY OF R OBJECTS IN OBJECT MODEL

To recover a R object is to connect all of its components, i.e., M objects, specified in object model to the R object. Therefore a link or a bridge between R objects and M objects should be established. This link is clearly given by the structures on R objects, that is, a R object is represented by several model objects, its main object and aspect objects, in object model. The big circles in figure 4 actually give the R objects "manager", "employee" and "project". Definitions of R objects' states, behavior and behavior constraints in terms of their related M objects' states, behavior and behavior constraints are given below.

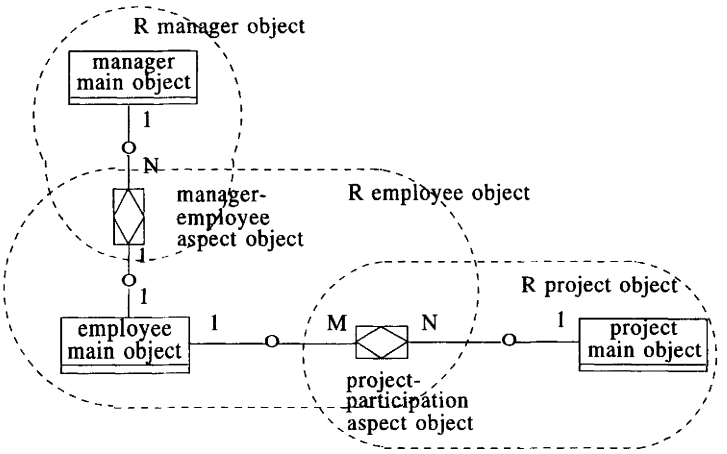

Figure 4. R objects in object model

Definition 8. A R object's state, behavior(or action events) and behavior constraints: a R object's state consists of its main and aspect objects' states(attributes), its behavior consists of its main and aspect objects' behavior(action events), and its behavior constraints consist of its main and aspect objects' internal behavior constraints (M-type I), and inter-M objects' behavior constraints (M-type II) between its related model objects (its main and aspect objects).

Definition 9. A R object's response events: an event is a response event of a R object when it involves the object but is not the behavior of the object.

A R object is now decomposed into several logically related model objects. Object model in this case can support two levels of objects, the real world objects and model objects, thus more information about reality can be provided by the object model. Usually end-users are interested in real world objects and software developers are interested in model objects. In fact end-users can understand object model without difficulties and hence can participate in business modeling.

Properties of a R object can be obtained from the related M objects. The gap between real world objects and model objects is now closed. Arbitrary decompositions could also work, but produce less fruitful and structured specifications. This is the case where modelers arbitrarily or non-systematically introduce M objects to object model without clear ideas about the relationships between the introduced M objects and R objects. Concepts about object, such as, inheritance, reuse, objects' states, behavior and behavior constraints etc., are now applicable to both levels of objects[7].

Finally it should be mentioned that the flexibility, extendibility, maintainability and reusability of the object models obtained based on the structured object-oriented view can be improved[7]. For example, adding (or deleting) a relationship in ER model needs only to add (or delete) one aspect object and the inter-M objects behavior constraints with other model objects. In addition, this structured view in fact provides a way to incrementally specify a real world object and system[7].

8. CONCLUSIONS

This paper introduces a structured object-oriented view (SOOV) on R objects and system, and a way to systematically decompose R objects of a system into M objects of object model. The view can not only solve the redundancy and inconsistency problems and recover the R objects in the specification at the same time, but also increase the software quality and productivity by improved flexibility, extendibility, maintainability and reusability.

REFERENCES

1. Dedene G. and Snoeck M., "M.E.R.O.DE.: A Model-driven Entity-Relationship Object-oriented DEvelopment method", research paper, K.U.Leuven, 1994.
2. Dedene G., " On the mathematical foundations of an object-oriented Business modelling approach ", L.I.R.I.S. K.U.Leuven research note 1990.
3. Dedene G., Snoeck M. and Depuydt A., "On generalisation/Specialisation Hierarchies in M.E.R.O.DE. object-oriented business modelling", research report, K.U.Leuven, 1992.
4. Dedene G., " M.E.R.O.DE. by examples. A tutorial on object-oriented Business modeling.", object technology 93, Cambridge U.K., March 1993.
5. Verhelst M., " Software Development ", Course-notes K.U.Leuven 1991.
6. Wu G., " Structure of objects based on their behavior and the optimized behavior constraint checking in M.E.R.O.D.E. ". Research note Oct., 1992, K.U.Leuven.
7. Wu G., "A systematic approach to M.E.R.O.DE. systems specification", doctoral seminar, April, 1994.
8. Jackson M., " System Development ", Prentice-Hall 1983.