Understanding Redundancy in UML Models for Object-Oriented Analysis

Dolors Costal, Maria-Ribera Sancho, and Ernest Teniente

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
Jordi Girona 1-3, 08034 Barcelona (Catalonia)
{dolors,ribera,teniente}@lsi.upc.es

Abstract. A phenomenon that frequently appears when designers define analysis specifications is that of redundancy between models. A correct and deep understanding of this phenomenon is necessary to help the task of the designer. In this paper, we study the problem of redundancy in UML Models for Object-Oriented Analysis. In this context, we identify different kinds of redundancies that may arise. We evaluate the impact of redundancy in specifications from the point of view of their desirable properties. We also propose how to obtain a canonical analysis model, which does not include any of the identified redundancies, and we sketch the possibility of having redundant views of some aspects of the canonical model.

1 Introduction

In this paper we study the problem of having redundant specifications at the analysis level. This is done for the particular case of object-oriented models, written in the UML. To the best of our knowledge, redundancy in object-oriented analysis models has not been considered before.

We say that an analysis model has redundancy when an aspect of the specified system is defined more than once. We think that identifying the different kinds of redundancy that may appear at the analysis level helps to fully understand the artifacts that are used in that stage. Furthermore, having (or not having) redundancy in a given specification has a great impact on their desirable properties, such as, understandability, modifiability, consistency and ease of design and implementation. Being aware of this impact helps the designer to define better and more comprehensible analysis models. Understanding redundancy also contributes to obtain better design models since it makes easier to decide the responsibilities of each software component.

In the context of the UML notation [RJB99], the most well-known proposal of software development process is the Unified Process (UP) [JBR99]. The Unified Process must be understood as a framework that encompasses the best development

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 659-674, 2002.

practices and not just as a strict universal process. The main strengths of the UP are to be use-case driven, architecture-centric, iterative and incremental.

One of the main difficulties that arise during software development is that of establishing a clear frontier between the analysis and design phases [Kai99]. We think, following the most accepted practices for software development [Rum96, Pres97] that this deserves special attention because the analysis model constitutes a permanent model of the reality in itself. In the UP (or in object-orientation in general) this is even more difficult due to the iterative process, the use of the same models in both phases and to the different criteria used by different authors.

As far as analysis is concerned, the UP admits a certain degree of freedom with respect to the way to view and employ the models or artifacts generated during this stage. In this sense, an interesting proposal is that of Larman [Lar98] because it provides good criteria to define the boundary between analysis and design and it proposes how UML artifacts have to be used acording to that criteria. Our work takes this proposal as starting point.

In this context, we study the already mentioned problem of having redundant specifications at the analysis level. One of the main points of this paper is to identify which kind of redundancies may arise. Moreover, we evaluate the impact of redundancy in specifications from the point of view of their desirable properties.

We advocate also for avoiding redundancy in the analysis models. In this sense, we propose how to obtain a canonical model which does not include any of the identified redundancies. Moreover, we sketch the possibility of having redundant views of some aspects of the analysis model to help the designer in his task. In this way, he may take advantage of having redundant and non-redundant models altogether.

This paper is structured as follows. Next section reviews the main UML models used during the Analysis phase according to Larman's proposal. Section 3 is devoted to identify different kinds of redundancy that may appear in that phase. Section 4 discusses the issue of redundancy: advantages and inconveniences and provides a set of guidelines to avoid the identified redundancies. Finally, we present our conclusions in section 5.

2 UML Models for Analysis

In this section, we will briefly describe the UML models used during the analysis phase following the main lines of Larman's proposal [Lar98]: the Analysis Use Case Model, the Conceptual Model, the System Behaviour Model and the Analysis State Model.

In Larman's work, analysis is characterized by emphasizing static and dynamic information about a system. Therefore, a static model describes structural properties of the system while a dynamic one describes the behavioural ones. Larman's main contribution relies on proposing to define the system behaviour as a 'black box' before proceeding to a logical design of how a software application will work. The main idea behind this solution has also been sketched in [Boo96, FS97, Mul97, Dou98, RV00].

This has a clear impact, during analysis, on the sequence diagrams definition and on the assignment of operations to classes. Sequence diagrams are considered as *system sequence diagrams* that show the events that external actors generate; their order and the system response to those events. On the other hand, operations responding to those external events are not assigned to particular classes since they are recorded in an artificial type named *system*. In this way, responsibilities are not assigned to objects during analysis.

We believe this is a good criteria to define the boundary between analysis and design. We should mention that some difficulties may arise when trying to develop these models along this direction. The overcoming of those situations is out of the scope of this paper.

In the following, we will use a simple example that is aimed to model the assignments of employees to departments in a certain company to illustrate the Analysis Models.

2.1 Use Case Model

In the plan and elaborate phase, high-level use cases are identified. They are defined very briefly, usually several sentences that describe a process. The Analysis Use Case Model consists of the definition of expanded use cases which are longer and more detailed than high-level use cases, in order to describe the course of events that occur as a result of actor actions and the system responses to them. In our example, we need a use case to fire employees. This use case is described in Figure 2.1.

Use Case: Fire employee
Actors: Director (initiator), Adm-Staff.
Overview: Fires an employee.
Type: Primary and essential.
Typical Course of Events:

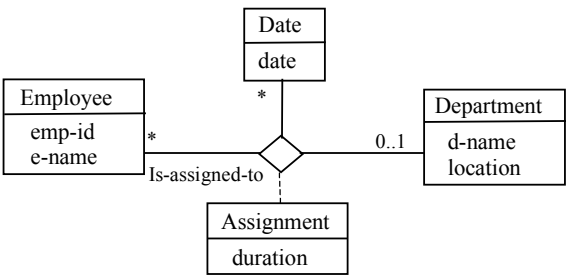
Actor Action	System Response
1. The use case begins when the director decides to fire an employee.	
2. An administrative staff introduces the employee identifier.	
	3. The system removes the employee and all her/his assignments, if this can be done.

Figure 2.1

2.2 The Conceptual Model

The Conceptual Model conforms the static part of the analysis and, consequently describes the structural properties of the objects that model concepts of the problem domain. It is illustrated with a set of static structure diagrams (a class diagram) in which no operations are defined.

For example, Figure 2.2 shows a static structure diagram. Object classes Employee, Department and Date are linked with the Is-assigned-to association. An occurrence of the association indicates that a particular employee has been assigned to a particular department at a given date. The corresponding association class named Assignment has an attribute to indicate the duration of this assignment. The diagram is complemented with textual constraints, which cannot be expressed graphically in the UML.



Textual Constraints:
- Class identifiers: (Employee, emp-id); (Date, date); (Department, d-name)
- An employee may not have two overlapping assignements.

Figure 2.2

2.3 The System Behaviour Model

The System Behaviour Model describes dynamic aspects of the system. As mentioned in the introduction of this section, the system behaviour is defined as a 'black box' in the Analysis phase and it is illustrated through system sequence diagrams and the contracts of the corresponding operations.

A system sequence diagram shows, for a particular course of events within a use case, the external actors that interact directly with the system, the system as a 'black box', and the system events that the actors generate. The following sequence diagram corresponds to the use case of Figure 2.1:

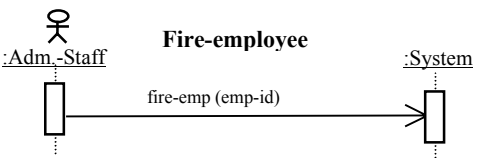


Figure 2.3

A system operation is an operation that the system executes in response to a system event. Therefore, there is a one to one correspondence between events and operations and we will refer to them as events or operations, indistinctly. It is defined by means of a contract that includes the signature of the operation, its semantics, a set of conditions that are guaranteed to be true when the operation is executed (its

precondition) and the set of conditions that hold after the operation execution (its postcondition). The contract of the previous operation *fire-emp* looks as follows:

Operation: fire-emp (emp-id)
Precondition:
Semantics: Fire an employee.
Postcondition:
1. If the employee emp-id does not exist then the operation is invalid.
2. Otherwise, the operation is valid and
   2.1. The employee identified by emp-id is deleted.
   2.2. All the assignments of this employee are deleted.

Figure 2.4

2.4 The Analysis State Model

Finally, the Analysis State Model consists of a set of state diagrams that illustrate the interesting events and states of objects, and the behaviour of those objects in reaction to an event. An example of analysis state model will be shown in Section 3.3.

3 Identifying Redundancy in the UML Analysis Model

Each one of the models explained so far shows different perspectives of a software system: the static one, the behavioural one, etc. It usually happens that a certain aspect of the system can be viewed from different models. We say that an analysis model has redundancy when an aspect is specified in more than one of its models.

In this section we identify several redundancies that may arise among the UML analysis models.

3.1 Conceptual Model and System Behaviour Model

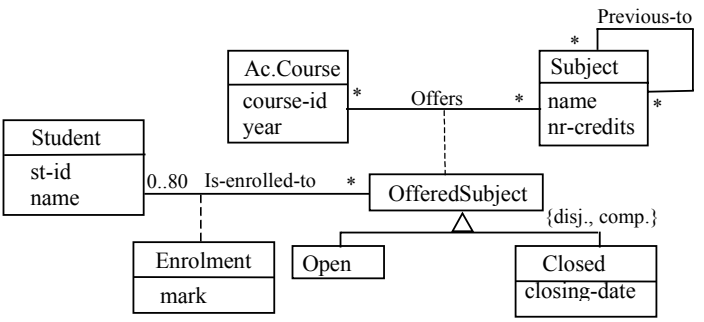
The Analysis Conceptual Model allows to express several constraints about the information that it defines. In general, we may consider three different kinds of constraints: *graphical*, *textual* and *structural*.

Graphical constraints (like multiplicity, generalization constraints, subset, xor, etc.) can be directly represented by means of a graphical symbol in the UML. Textual constraints define conditions that the conceptual model must satisfy but that can not be graphically represented. They will usually be specified in the OCL language [WK99]. Structural constraints are implicit in the Conceptual Model and, thus, they are not represented neither graphically nor textually. For instance, each association (associative class) implies a structural constraint stating that there may not exist two instances of an association linking the same objects.

On the other hand, the System Behaviour Model includes Operation Contracts that describe the effect of the operations upon the system and, in particular, how the information of the Conceptual Model will be updated.

It is not difficult to see that an update may lead to a violation of a certain constraint. In this sense, it may happen that the post-condition of a certain operation contract checks a constraint that is already entailed by the conceptual model.

Example 3.1: consider the following Conceptual Model, which is aimed to model the Enrolments of Students into Subjects, during Academic Courses. Subjects may be offered during academic courses. An offered subject can be open or closed. When an offered subject is closed, no new students can be enrolled in it.



Textual Constraints:

- Class identifiers: (Student, st-id); (Ac.Course, course-id); (Subject, name)
- A student may not be enrolled in a subject if s/he had a mark greater than 5 in a previous enrolment of that subject.
- A student may not be enrolled in a subject if s/he does not have a mark greater than 5 in all its previous subjects.
- A student must be enroled at most in five subjects during an academic course.
- The association Previous-to is transitive.
- A subject may not be previous to itself.

Figure 3.1

The previous Conceptual Model specifies several constraints. In particular, regarding the enrolments of students into subjects, it entails that:

An offered subject is defined by a subject and an academic course (structural constraint). Two different assertions are deduced from this constraint. First, it guarantees that two different instances of offered subject may not be linking the same instances of subject and academic course. Second, if an offered subject exists, it is guaranteed that the subject and the academic course also exist.

An instance of the association *Is-enrolled-to* is defined by a student and an offered subject (structural constraint). Two different assertions are deduced from this constraint. First, it guarantees that two different instances of enrolment may not be linking the same instances of student and offered subject. Second, if an instance of the association exists, it is guaranteed that the student and the offered subject also exist.

a) An offered subject must have at most 80 enrolled students (graphical).
b) An offered subject may not be open and closed at the same time (graphical).
c) A student may not be enrolled in a subject if s/he had a mark greater than 5 in a previous enrolment of this subject (textual).

d) A student may not be enrolled in a subject if s/he does not have a mark greater than 5 in all its previous subjects (textual).
e) A student must be enrolled at most into 5 subjects during an academic course (textual).

There are at least four operations that may violate some of the previous integrity constraints: to offer a subject in an academic course, to enrol a student, to change the enrolment of a student and to close an offered subject. We will illustrate this issue in the context of the operation that enrols a student in a subject offered in an academic course which could be specified by means of the following contract:

Operation: enrol-1 (course-id, subject-name, st-id)
Precondition:
Semantics: To enrol a student into a subject offered in an academic course.
Postcondition:

1. If the student st-id or the course course-id or the subject subject-name do not exist, then the operation is invalid.
2. If the subject subject-name is not offered in course course-id, then the operation is invalid.
3. If the student st-id is already enrolled into subject subject-name for the course course-id, then the operation is invalid.
4. If the student st-id has already five enrolments corresponding to academic course course-id, then the operation is invalid.
5. If the student st-id has already an enrolment into subject subject-id with a mark greater than 5, then the operation is invalid.
6. If the offered subject corresponding to course-id and subject-name is closed, then the operation is invalid
7. Otherwise, the operation is valid, and
   7.1 An instance of the association Is-enrolled-to, defined by the corresponding Student, Subject and Academic Course is created.

Figure 3.2

Clearly, the postconditions 4 and 5 and the integrity constraints g) and e) are redundant. Also, postconditions 1, 2 and 3 are redundant with the structural constraints a) and b). In this latter case, it is not difficult to see that structural constraints guarantee that an enrolment of a student in an offered subject can only be performed if the student is not already enrolled on that offered subject and if the student and the offered subject exist. Similarly, if an offered subject exists, the academic course and the subject also exist.

On the other hand, if we had specified the contract of Figure 3.3 for enrolments, the previous redundancy would not appear since the postconditions of this operation do not include the checking of the previous constraints.

Note that the existence of redundancy between the operation and the conceptual model does not imply that all constraints of the conceptual model that affect the operation are included in its postconditions. As an example, enrol-1 does not check

constraints c) and f) although they can be violated by enrolling a student into an offered course.

Operation: enrol-2 (course-id, subject-name, st-id)
Precondition:
Semantics: To enrol a student into a subject offered in an academic course.
Postcondition:

1. If the offered subject corresponding to course-id and subject-name is closed, then the operation is invalid
2. An instance of the association Is-enrolled-to, defined by the corresponding Student, Subject and Academic Course is created.

Figure 3.3

Therefore, in addition to the conditions already implied by its postconditions, an operation will also be invalid if any of the constraints entailed by the conceptual model is violated. For instance, operation enrol-2 will be invalid if any of the constraints entailed by the conceptual model is violated due to its postcondition.

In general, it is very difficult to ensure that an operation contract guarantees all constraints specified in a conceptual model. For this reason, whenever we talk about redundancy in this paper we will be referring to relative redundancy since absolute redundancy (in the sense that operation contracts specify all the aspects of the software system that can be affected by operation execution) is as difficult to achieve as non-redundancy.

3.2 Redundancy inside the System Behaviour Model

In addition to the Operation Contracts, the System Behaviour Model includes System Sequence Diagrams that specify the external events generated by actors in the context of a use case, its order, and the system operations that respond to those events.

Since, at the analysis level, there is a one-to-one correspondence between external events and system operations, the system sequence diagram already guarantees that the operations are handled in a specific order. Consequently, a certain operation is only executed when all previous operations have been handled satisfactorily. Therefore, it is guaranteed that the postconditions of its precedent operations in the sequence diagram are already satisfied.

A System Sequence Diagram and an Operation Contract would be redundant if the contract checks aspects already guaranteed by the sequencing entailed by the sequence diagram.

Example 3.2: assume that there is a use case that, at the beginning of an academic course, offers a new subject and enrols some students to that subject. The system sequence diagram of this use case could look as follows:

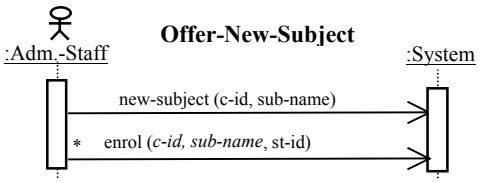


Figure 3.4

The operations *new-subject* and *enrol* should be specified by means of the corresponding contracts. The first operation, *new-subject*, is aimed to add the subject as an offered subject of the given academic course. This offered subject is specialised to open. The second operation, *enrol*, will perform individual enrolments of students for that subject into that course.

Note that, in a non-redundant analysis model, the contract of the operation *enrol* does not need to check none of the conditions that are entailed by *new-subject*. In this sense, part of the postconditions of *new-subject* become preconditions of *enrol* and, thus, they must not necessarily be specified again in its contract.

In particular, postcondition 1 stating that the offered subject must be open is not necessary. Then, the corresponding contract enrol-3 would be:

Operation: enrol-3 (course-id, subject-name, st-id)
Precondition:
Semantics: To enrol a student into a subject offered in an academic course.
Postcondition:

1. An instance of the association Is-enrolled-to, defined by the corresponding Student, Subject and Academic Course is created.

Figure 3.5

In some cases, it is very difficult to avoid redundancy completely among sequence diagrams and operations contracts. This will happen when a certain event (operation) may appear in several sequence diagrams since the events previously invoked will differ from diagram to diagram. In such cases, it is not possible to ensure that the conditions that are already satisfied are always the same and we have to choose between considering a different operation contract in each case or considering a single operation contract that may be redundant for a certain sequence diagram.

As an example, assume an additional use case that enrols a student in a subject. The system sequence diagram of this use case could look as follows:

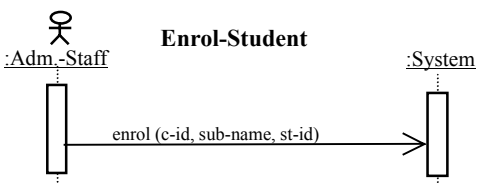


Figure 3.6

In this case, the contract for the operation *enrol* must check that the offered subject is open since it is not guaranteed by a previous operation. As a consequence, the operation contract enrol-2 would be adequate. When this happens, we may choose between specifying two slightly different operations (*enrol-3* for *Offer-New-Subject* and *enrol-2* for *Enrol-Student*) or defining a single enrol operation (*enrol-2*) that would be partially redundant when invoked from sequence diagram *Offer-New-Subject*.

3.3 Analysis State Model and System Behaviour Model

An Analysis State Diagram shows the life-cycle of an object, the events that affect it, its transitions and the states it is in between these events. In this sense, it shows the behaviour of an object in reaction to events.

Each state transition specified in the diagram defines changes on an object that are caused by the invocation of a certain event (operation) on that object. An object may only suffer a certain transition if its state is the one required for the transition when the event is invoked.

An Operation Contract and a State Diagram are redundant if the contract checks conditions about the state of the objects that are already entailed by the state diagram transitions.

As an example, assume the following Analysis State Diagram for offered subjects:

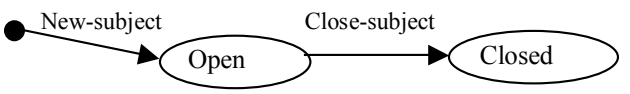


Figure 3.7

This state diagram specifies that the event *new-subject* is the creation event for offered subjects and that the event *close-subject* can only be applied to open offered subjects. Therefore, it would be redundant to specify this information in the contracts of the corresponding operations. The designer could easily specify the following contract to close an offered subject:

Operation: Close-subject (sub-name, course-id)
Precondition:
Semantics: To close a subject of an academic course.
Postcondition:

1. If the offered subject o defined by the academic course course-id and the subject sub-name is not Open then the operation is invalid.
2. Otherwise, the operation is valid and the specialization of the offered subject o is changed from open to closed.

Figure 3.8

The postcondition 1 of the previous contract is redundant since it specifies an aspect already defined in the state diagram for offered subjects defined in Figure 3.7.

4 Dealing with Redundancy in the UML Analysis Model

In this section we discuss about the advantages and inconveniences of considering non-redundant UML Analysis Models. We propose some guidelines to avoid redundancy and we define the semantics of the execution of a sequence diagram in this case. We finally show the advantages that could provide a tool able to generate (redundant) views of a non-redundant model.

4.1 Discussion about the Advantages of Non-redundant Analysis Models

We think that a non-redundant analysis model presents the following advantages as far as software development is concerned:

1. It contributes to desirable properties of a software system specification

Modifiability and consistency are some of the properties that must be achieved to obtain a well-written specification [Dav93].

Usually, a software system is under continuous evolution. A specification is *modifiable* if changes to the requirements can be made easily, completely and consistently. It is not difficult to see that, if a certain redundant aspect evolves, we need to modify it in all the models where it is specified. On the contrary, this does not happen in a non-redundant specification.

For instance, if it happens that the requirement about the maximum number of subjects a student may be enrolled in during an academic course changes from five to seven, it will be more difficult to modify the analysis model if we consider the operation contract enrol-1 (see Figure 3.2) than with the operation contract enrol-3 (see Figure 3.5)

On the other hand, a specification is *consistent* if no subset of requirements stated therein conflict. Again, in a redundant specification, it is easier to define inconsistent requirements because, since a single aspect is specified several times and in several models, it is difficult to guarantee that it is always specified in the same way.

For instance, it will be more difficult to keep the specification consistent with the operation contract enrol-1 since we could easily have specified in it that the maximum number of subjects for a student into an academic course is three, which would clearly enter in contradiction with the information provided by the conceptual model.

2. It facilitates software design and implementation

Redundancies at the analysis level will be easily propagated to the following stages of software development, i.e. to design and implementation, causing in general a significant reduction in the efficiency of the final system. For instance, a certain aspect could be designed and implemented twice, also in two different ways, if the designer or the programmer do not realise that it is redundantly specified.

Consider again the conceptual model of Figure 3.1. If we are implementing it in a relational database, we will probably obtain a (partial) logical schema with the following tables (primary key attributes are underlined):

Ac-course (course-id, year)
Subject (s-name, nr-credits)
Offered-course (course-id, s-name)
    {course-id} is a foreign key that references Ac-course
    {s-name} is a foreign key that references Subject
Student (st-id, name)
Enrolment (course-id, s-name, st-id)
    {course-id, s-name} is a foreign key that references Offered-course
    {st-id} is a foreign key that references Student

If we directly design the transaction corresponding to the operation contract enrol-1 (see Figure 3.2), the transaction will include a check for each of the validation postconditions of the operation contract. In particular, it will verify that (postconditions 1, 2 and 3):

- Student st-id, course course-id and subject subject-name exist.
- Course-id is offered in course course-id.
- Student st-id is not enrolled into subject subject-name for the course course-id.

However, it is not difficult to see that due to primary and foreign keys the database alone will already guarantee that these postconditions are satisfied, (and thus, that they must not be implemented inside the transaction).

In a similar way, we could implement the checking of the constraints e) and g) directly into the database management system by means of triggers or stored procedures. Again, the transaction corresponding to enrol-1 would probably also unnecessarily include a check to guarantee that these constraints are not violated by the transaction.

Although it can be argued that this drawback can be prevented if software design includes a first initial step to remove redundancy between the different models, we believe that this would enter into a contradiction with the way in which we do analysis since we would first redundantly specify an aspect into different models to remove the same redundancy afterwards.

4.2 Defining a Canonical (Non-redundant) Analysis Model

As we have seen in the previous section, non-redundant analysis models present several advantages over the redundant ones. Therefore, we need some criteria to define non-redundant models. We call *canonical analysis model* the non-redundant analysis model resulting from applying the criteria proposed in this section.

Removing redundancy between the graphical and structural constraints of the conceptual model and the operation contracts can only be done by avoiding the definition of the corresponding postconditions in the operation contracts since they cannot be removed from the conceptual model.

In a similar way, redundancy between sequence diagrams or the state model and the operation contracts can only be done by avoiding the definition of the corresponding postconditions in the operation contracts. One could argue that it could also be removed from the sequence diagram or the state model but this would imply not to define them at all. Nevertheless, since the utility of these UML diagrams is doubtless, we believe that this alternative is not realistic.

To remove redundancy between textual constraints in the conceptual model and the postconditions of the operation contracts, we could choose to specify either the constraints or the postconditions. In the first case, if we just specify textual constraints, we would increase the localization of the information definition since we would specify a constraint a single time in the conceptual model rather than distributed among all the operations that may affect it. Since localization is an important issue regarding the construction, understanding and changeability of analysis models [Oli86, CSO+97], we think that this is the best alternative to obtain the canonical model.

For instance, we increase the localization of the information definition if the constraint that at most 80 students can be enrolled into an offered course is specified only once in the conceptual model instead of several times in the contracts of the operations to enrol a student, to change the enrolment of a student, to offer a new subject, etc., i.e. once for each operation that may violate this constraint.

The canonical model of our example of section 3 would contain the conceptual model of Figure 3.1, the sequence diagram of Figure 3.4, the state diagram of Figure 3.7 and the operation contracts of enrol-3 (Figure 3.5) and close-subject (Figure 3.8, without the postcondition 1).

As a conclusion, we can see that the canonical analysis model keeps away from redundancy by avoiding it in the definition of the postconditions of the operation contracts. This is not surprising since we can observe that operation contracts are involved in all different types of redundancy that we have identified. Moreover, an aspect specified into a certain model (other than the contracts), may be affected in general by several operations. Therefore, to increase localization, it is more reasonable to specify it in only one place.

Furthermore, this proposal leads to postconditions with only dynamic constraints (all the static ones are specified in the conceptual model). As a side effect, this results in an emphasis of dynamic constraints that improves their understandability.

Finally, in the canonical model, since only the operation contracts are textual descriptions, a certain aspect is specified graphically whenever possible. This results also in a more comprehensive analysis model.

4.3 Semantics of Sequence Diagrams Execution

When we consider non-redundant operation contracts, its execution semantics depend on the rest of the analysis models. Therefore, an operation postcondition must be understood as something that is performed as a necessary, but not always sufficient, condition to execute the corresponding operation. In this sense, it is not guaranteed that an operation will be executed satisfactorily if its postcondition succeeds, since its execution may lead to a violation of a constraint specified somewhere else.

As a consequence, the semantics of the execution of a system sequence diagram has to be defined as follows:

- The conditions entailed by a State Diagram must be satisfied before the execution of each operation appearing in the system sequence diagram.
- The conditions entailed by the Conceptual Model must be satisfied after the execution of all the operations that appear in a system sequence diagram.
- For each operation of the system sequence diagram, its postconditions do not lead to an invalid execution of the operation.
- If any of the previous three conditions does not hold, then all the operations are rejected and the information base is not updated at all.

Note that this semantics is required not only for non-redundant models but also for any relative redundant model. Therefore, in practice, as absolute redundancy is almost never achieved, the semantics stated here should be the usual one.

4.4 Redundant Views of a Canonical Analysis Model

The whole idea of analysis models is to foster understanding and this may sometimes require to isolate certain aspects from other aspects. Since many aspects participate in multiple views, one could argue that this redundancy is unavoidable and, in this way, disagree with the guideline that suggests not to have redundancy in the analysis model. However, the use of a canonical model does not prevent the designer having redundant views of the analysis model.

In fact, the use of a graphical language like the UML is based on the assumption that we will be using computer-based tools and not 'paper and pencil' or mere semantics-free drawing tools. Therefore, we think that such a CASE tool should store explicitly only the canonical model and should be able to reason about it to identify redundancy that could appear in the operation contracts and to keep consistency among them. Moreover, a tool of this kind could also be able to generate a skeleton of an operation contract that includes all aspects already specified in other models. This would clearly help the work of the designer since he would have a more complete view of the impact of each decision taken. The definition of such a CASE tool is far beyond the scope of this paper and it is left for further research.

For instance, if we had the conceptual model of Figure 3.1, we could think of a CASE tool that would automatically generate the following absolute redundant skeleton of a contract for the operation to enrol a student into an academic course. Note that this skeleton includes the checking of all constraints specified in the conceptual model.

Operation: enrol (course-id, subject-name, st-id)
Precondition:
Semantics: To enrol a student into a subject offered in an academic course.
Postcondition:

1. If the student st-id or the course course-id or the subject subject-name do not exist, then the operation is invalid.
2. If the subject subject-name is not offered in course course-id, then the operation is invalid.
3. If the student st-id is already enrolled into subject subject-name for the course course-id, then the operation is invalid.
4. If the student st-id has already five enrolments corresponding to academic course course-id, then the operation is invalid.
5. If the student st-id has already an enrolment into subject subject-id with a mark greater than 5, then the operation is invalid.
6. If the offered course defined by subject subject-name and academic course course-id has 80 enrolments already, then the operation is invalid.
7. If, for any of the subjects previous to subject-name, the student st-id does not have a mark greater than five into some enrolment in that subject, then the operation is invalid.

Figure 4.1

Even if such a CASE tool does not exist yet, we believe that the advantages we have discussed in section 3 justify putting the effort to generate a canonical model rather than falling into redundancy. Moreover, the inexistence of such a CASE tool alone does not justify the preference for redundant analysis models since, as we have seen, it is as difficult to define an absolute redundant analysis model (the only way we can get rid of all interactions between multiple views) by hand than a canonical one.

5 Conclusions

In this paper, we have studied the problem of redundancy in the context of the UML Analysis Models. First, we have identified redundancies that may arise and we have classified them in three different types: redundancy between the Conceptual Model and Operation Contracts, redundancy between System Sequence Diagrams and Operation Contracts and redundancy between Analysis State Diagrams and Operation Contracts. We think that being aware of these different kinds of redundancy helps the designer to fully understand the relationships that exist among different parts of an specification.

We have also justified the advantages of non-redundant analysis models in terms of desirable properties of software specifications and ease of design and implementation. We have defined a canonical model that avoids redundancy by not including already entailed checkings in the postcondition of operation contracts. We have described several advantages that our canonical model presents over alternative ways of avoiding redundancy. We have provided an adequate semantics of Sequence Diagrams

execution in order to define the correct behaviour of non-redundant operations. This semantics should also be applied in case of having not fully redundant operations. Finally, since redundancy may be useful in some cases to permit the isolate use of certain parts of an specification, we have sketched the possibility of having redundant views of some aspects of the analysis model.

Acknowledgements

We would like to thank Antoni Olivé, Cristina Gómez and the anonymous referees for their useful comments. This work has been partially supported by the CICYT program project TIC99-1048-C02-01.

References

[Boo96] G.Booch. “Object Solutions: Managing the Object-Oriented Project”, Addison-Wesley, 1996.

[CSO+97] D.Costal; M.R.Sancho; A.Olivé; M.Barceló; P.Costa; C.Quer and A.Roselló. "The Cause-Effect Rules of ROSES", Proc. of the First East-European Symposium on Advances in Databases and Information Systems (ADBIS'97), St. Petersburg, September 1997, pp. 399-405.

[Dav93] A.M. Davis “Software Requirements. Objects, Functions and States”, Prentice-Hall, 1993.

[Dou98] B.Douglass. “Real-time UML: Developing Efficient Objects for Embedded Systems”, Addison-Wesley, 1998.

[FS97] M.Fowler and K.Scott. “UML Distilled”, Addison-Wesley, 1997.

[JBR99] I.Jacobson; G.Booch and J.Rumbaugh. “The Unified Software Development Process”, Addison-Wesley, 1999.

[Kai99] H.Kaindl. “Difficulties in the Transition From OO Analysis to Design”. IEEE Software, Sept./Oct. 99, pp. 94-102

[Lar98] C.Larman. “Applying UML and Patterns”, Prentice Hall, 1998.

[Mul97] P.A.Muller. “Modélisation Object avec UML” (in french), Éditions Eyrolles, 1997.

[Oli86] A.Olivé. “A comparison of the operational and deductive approaches to conceptual information systems modelling”, Proc. IFIP-86, North-Holland, Dublin, 1986, pp. 91-96.

[Pre97] R.Pressman. “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 1997.

[RJB99] J.Rumbaugh; I.Jacobson and G.Booch. “The Unified Modeling Language Reference Manual”, Addison-Wesley, 1999.

[Rum96] J.Rumbaugh; et al. “Object Oriented Modeling and Design”, Prentice-Hall, 1996.

[RV00] P.Roques and F.Vallée. “UML en action” (in french), Éditions Eyrolles, 2000.

[WK99] J.Warmer and A.Kleppe. “The Object Constraint Language”, Addison-Wesley, 1999.