Scattered Associations in Object-Oriented Modelling

Kasper Østerbye
Norwegian Computing Center, Postboks 114 Blindern, 0314 OSLO, Norway.
Kasper.Osterbye@nr.no

Johnny Olsson
WM-data, Hermodsvej 22, 8230 Åbyhøj, Århus, Denmark,
jooln@wmdata.com

Abstract. The popularity of UML has bought a new abstraction mechanism, associations, into the realm of object-oriented programming. Associations are used in the analysis and design phases of software development. We have used UML in a paper and pencil redesign of a large existing system where the goal was to move from a relational model to an object-oriented model. Our experience has shown us that the concept of associations in UML is very useful, but also that there are several shortcomings regarding their expressive power. *Scattered associations* are global association structures whose definition is scattered throughout a UML diagram. We have identified two kinds of scattered associations. A *structural association* is a scattered association that defines the structure of a set of objects, and a *co-variant association* is a scattered association that describes pair-wise parallel inheritance hierarchies.

1 Introduction

The graphical modelling language UML [Rational, 1998] has brought associations into object-oriented modelling, and seem to complement the existing modelling capabilities well. A hallmark of object-oriented modelling has been its closeness to human conceptualisation, with objects representing nouns, and methods/functions/procedures representing verbs. Associations can be said to model the word class, transitive verbs, with one object being the *subject* and the other the *object*, e.g. WM-data *employs* Johnny. When we talk about the association in general, we will often use a substantiated verb, e.g. *employment.*

In a model of a large system concerning administration of students, student programs, teaching, exam planning, and public scholarships, we have encountered the situation that the same association ties together elements of different types. We call the situation "scattered associations", as the typical characteristic is that the same association type is used many places in the same class diagram.

Our experiences are drawn from an experiment where we used an extension of Smalltalk that provides linguistic support for associations [Østerbye *et al*, 1998]. The extension builds on the DSM model developed by Rumbaugh and others [Rumbaugh, 1987][Shan *et al*, 1989].

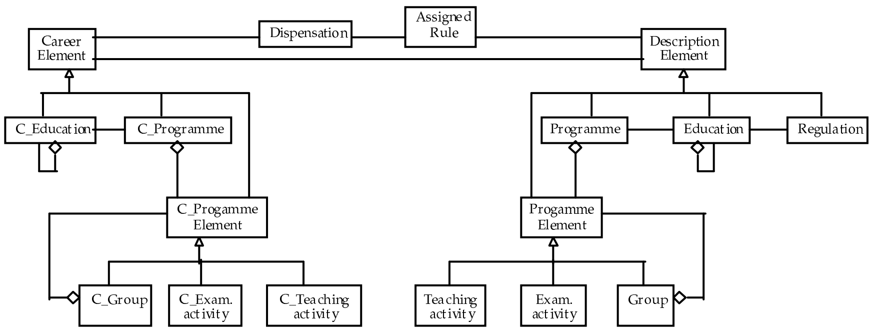
The system from which the experiences are drawn will be presented first, followed by a discussion of the notion of scattered associations. Two different types of scattered associations are identified, structural and co-variant scattering. The general principles behind these two are presented, and several possible solutions are discussed.

2 The STADS rule checker

The case deals with one aspect of a large generic administration system called STADS[1]. The selected part deals with describing educations and the courses and exams that constitute these educations. It deals with describing the concrete career of individual students, and, it deals with describing the rules that make up the passing criteria and structural constraints of the educations themselves. The analysis and design of the STADS system is further described in [Olsson *et al.* 1998].

To provide a framework for *describing* educations, concepts ranging from government regulations and degrees, to courses and individual exams have been defined. Each of these concepts can be described using rules. For an individual exam, the rule might state the passing grade, for a course it might specify the minimum grade average needed for its constituent grades. The rules associated with a degree might state that a certain number of courses must be passed, or that all exams must be passed with a certain minimum grade. The *career* of a specific student is the courses actually followed by the student. Most educations are structured with required as well as voluntary courses. The career of each student therefore varies in structure as well as in grade results, even if they might end up with the same degree.

Associations have played an important role in describing the STADS rule checker. They have been used to describe the association between the different descriptive elements (degrees etc.), the association between the corresponding career elements, and to associate career elements with their descriptions. Associations are also used to associate descriptive elements to their rules, and to associate individual dispensations from the rules with career elements for the student who has the dispensation. The above diagram gives a fuller picture of the model.



Simplified class diagram for the STADS System. "C_" is short for Career

The class diagram contains two main hierarchies, the right on rooted in *Description Element*, and the left in *Career Element.* A descriptive element is an element that de-

[1] STADS is a short for 'STudieADminstrationsSystem' (Education Administration System in Danish). The system concerns administration of students, student programs, teaching, exam planning, and public scholarships. STADS has been developed since 1993. The first universities started to use the system in 1996. As the system gets more developed, more universities are taking in into use. The system is implemented on a Oracle 7 platform and works for UNIX as well as VMS

scribes an element of an education. A career element is a part of the career for a specific student. It is worth to notice that the career elements are *not* instances of the descriptive elements. The reason is that the descriptive elements are tangible objects in the domain of the system. The users of the system are used to having such things as course descriptions in their problem domain. This is an example of the object-oriented analysis pattern “item-description” described in [Coad, 1992].

In the diagram, there is an association between Career element and Description element. The two hierarchies are nearly *parallel*, for each descriptive element, there is a corresponding career element. The association between Career element and Description element is interpreted in a pair wise fashion, e.g. associating Exam_activity to C_Exam_activity, Programme to C_Programme *etc*. Just as the career-description association is intended to work in a pair wise fashion, the assigned rule and dispensations are supposed to work in a pair wise manner. All rules are associated to descriptions, and dispensations are associated to rules as well as the career element for a specific student. A dispensation must associate to a career element that corresponds to the descriptive element of its rule.

2.1 Experiences

As a first step in implementation, each association on the UML diagram must be given a name. To come up with meaningful names for each association was very hard, and in the end we just settled for a simple systematic naming involving the first part of each participating class. We believe, however, that the problem has a more profound explanation. The element hierarchies have both a specializations structure, and a containment structure. The containment structure is described using aggregation and simple association in the diagram. Many of the associations in both career and description are *containment associations*, and a compelling idea is to interpret these unnameable associations as making up a single association (one for career and one for description), which exactly defines the containment structure. This is also reflected by the observation that we were lacking a proper object, which captured the containment structure of a programme and a career, throughout the work with the STADS system.

A programme element can be part of both a programme and a group (which is a modelling concept used to provide some reuse of rules and descriptions). If we move upwards in the containment hierarchy, this leaves us with a typing problem because the type of the object is not known when we traverse the containment association, we know that it is either a program or a group.

An implementation of associations should allow us to move from one object to the other using role names specified as part of the association definition. The association binding career elements to program elements raised the problem that the return type of the roles is different for each matching pair of specialised element type.

3 Scattered associations

As mentioned in the previous section, our overall experience of using UML for modelling was a mixed blessing. There are two points where we have encountered problems. In both the description part and the career part of the model, the different elements make up a hierarchical containment structure. We need a way to view this structure as a single property of the model, but as it is represented in the UML diagram, its definition is scattered all over the diagram.

Another problem is the association that binds career elements to their associated description elements. This association needs to be refined (not done in the diagram) to specify that career programmes are associated to programmes, that career groups are associated to groups, that is, the two hierarchies are parallel. Two problems arise from this. First it would be incredible tiresome to create all the actual associations in the diagram. Second, and more serious, the semantics of this is not quite simple to work out. Again, we believe that this pattern of associations across parallel inheritance hierarchies is a reoccurring pattern, and it is addressed in section 3.2.

3.1 Containment associations

The career elements are organised into a career structure, as are the description elements. In both cases this containment structure is made up from a number of individually defined associations. However, it was our experience that we needed to be able to talk about the containment structure as such, as it in many situations provide the right level of abstraction. At an intuitive level, we feel that the containment association can be decomposed into the individual associations. That is, one can talk about the same association at different levels.

We feel that the containment structure is in nature a scattered association. Our solution is to assume that an association can be defined more than one place in a class diagram. This mean that the containment association associates exams to programs, programs to educations etc. This works well in most cases, but a problem arise when the same class of objects can participate two different places in the containment structure, e.g. an exam can be part of a program or of a group.

To solve the problem, we extend the simple Relation definition to allow several clauses as in:

```
Relation EducationSctructure is
  (element: Education ↔ regulation: Regulation  OR super_education: Education) AND
       (program: Program ↔ group: Education 0:M) AND
       (element: ProgElement ↔ program: Program OR group: Group 0:M)
```

This states that an education is associated to either a regulation, or to another education (itself being a sub-education). In the actual STADS system the OR’s are actually not exclusive, as the education structure is hierarchical, though not tree structured (it is a directed a-cyclic graph). The above seem like a solution at the moment, but the introduction of Boolean connectives indicates to us that the final solution has not yet been reached.

3.2 Covariant associations

The exact semantics of the association between career elements and descriptive elements is so that it does not associate arbitrary career elements to arbitrary description elements. This association is indeed the association that establishes that the two hierarchies are parallel. The problems relate to uncertain semantics of associations in connection with specialisation. If we assume that association A associate class C to class X. Then one will expect that A also associate a subclass D of C to the X class. If Y and Z are specializations of X, then A might associate a D object to both an X, Y, and Z object. This seems a reasonable way to interpret the interplay between an association and class inheritance.

But in the case of parallel hierarchies, the situation is such that A should be constrained in some way so that the general element to element association will not enable all possible associations, but can be restricted in the proper way.

As before, the fundamental question is if this the restricted association is a *new* association, or if it is a restriction of the existing one. However, it seems that in defining a new association for each parallel pair the general picture is easily lost, e.g. the new relation need to specify that it is a elaboration of the general one. Another problem is that the general one still exists but is useless, as all concrete associations will be done using new associations. We therefore believe that it is most promising to find a way to restrict the existing association. That way we can maintain the view that there is really only one association, which "does the right thing".

The title of the section indicates a hybrid solution, which has its origin in virtual classes as known from BETA [Madsen *et al.* 1993]. If the general relation is described as associating a virtual career element to a virtual description element, then new concrete associations can bind these appropriately. In a concrete syntax, this might be:

```
Relation Description
    (career: virtual CareerElement ↔ description: virtual DescriptionElement).
```

Without full understanding of what specialisation of associations is, Description could then be specialized as:

```
Relation ExamDescription is-a Description
    (career: bind CareerExam ↔ description: bind Exam)
```

The term covariant in the title refers to the idea that both association ends are specialised simultaneously.

In [Shan *et al*, 1989] access methods are automatically compiled into the classes that are associated. This means that a method for accessing the description will be compiled into the CareerElement class. If the type of association endpoints is declared virtual, the return type of the description access method is virtual, and the ExamDescription association overrides this virtual to return an Exam element rather than a DescriptionElement. The notion of virtual classes seems to provide an elegant solution for covariant associations. However, no appropriate solution has been found for notations that does not support virtual classes.

4 Conclusion

During our work in modelling the STADS rule checker using the UML diagram notation, we found associations to be quite natural. However, we came across two situations in which the notation seemed in adequate in its support for associations. The common theme in these two situations was that a number of associations seemed really to be only constituents of a larger association, which could not be expressed in UML. The two situations were named structural associations and covariant association, and their general term is scattered associations. For both kinds of associations linguistic solutions were proposed.

In the case of structural associations, we propose to expand the notion of a binary association, so that the same association can be used to associate different kinds of objects. The concrete proposal introduced Boolean connectives. While this seem to solve the problem, it is not quite satisfactory, as we believe that a linguistic solution which captures the solution in a combination of aggregation and inheritance will ultimately prove more elegant.

Such a solution present itself for covariant association. Here the problem was how to relate parallel hierarchies. Here a proper linguistic solution did arise. By declaring the endpoints of an association virtual, these endpoints can then be further bound in specializations of a general association.

However, it is important to point out that specialisation of associations has no known well-defined semantics. We believe that work like this may aid in laying down the experience needed to define such semantics.

Acknowledgements. This work has been supported by the Danish Centre for Information Technology in the subproject “COT project 74.1”.

5 References

[Coad, 1992] P. Coad, *Object-Oriented Patterns*, 1992, Communications of the ACM, September 1992, pp. 152-159.

[Madsen *et al.* 1993] O. L. Madsen, B. Møller-Pedersen and K. Nygaard. *Object-Oriented Programming in the BETA Programming language*. Addison-Wesley, 1993.

[Olsson *et al.* 1998] J. Olsson, K. H. Nielsen, K. Østerbye, A. R. Lassen. *Objektorienteret Analyse og Design af udvalgte dele af STADS.* Technical report COT/4-03. 1998. Centre for IT-research, Aarhus University, Ny Munkegade, Bygn. 540, 8000 Århus C, Denmark. (In Danish).

[Rumbaugh, 1987] J. Rumbaugh. Relations as Semantic Constructs in an Object Oriented Language, *Proceedings of OOPSLA’87.* Pages 466-481.

[Shan *et al*, 1989] A. V. Shan, J. Rumbaugh, J. H. Hamel, and R. A. Borsari. DSM: An Object-Relationship Modelling Language. *Proceedings of OOPSLA’89.* Pages 191-202.

[Rational, 1998] *UML resource center*, http://www.rational.com/uml/.

[Østerbye *et al*, 1998] K. Østerbye, A. R. Lassen, J. Olsson. Object Relational Modelling. Technical report, COT/4-04, 1998. Centre for IT-research, Aarhus University, Ny Munkegade, Bygn. 540, 8000 Århus C, Denmark.