A Model for Classification Structures with Evolution Control *

M. C. Norrie, A. Steiner, A. Würgler, M. Wunderli
Institute for Information Systems
ETH Zurich, CH-8092 Zurich, Switzerland.

Abstract

We present a general model for classification structures that supports object role modelling. Objects can be associated with many roles simultaneously in a way that provides multiple classification views over objects. Further, objects can change their roles through migration within a classification graph. Object migration is controlled through the structure of the classification graph and the types of the nodes, thereby restricting possible evolution paths. We show how the model can be generalised to classification structures over relationships. In addition, we consider approaches to constraint maintenance in the case of object and relationship evolution.

1 Introduction

Entities of application domains change their roles during the course of their lifetime. To reflect these role changes in the corresponding database application system, the system must have mechanisms to support and control the evolution of the objects which represent those entities. Support for object evolution requires an operation for object reclassification and this may involve changing the type of an object as well as the addition and removal of the object in various classification sets. Control over object evolution requires a means of restricting the reclassification of objects to ensure that only meaningful forms of evolution are possible.

Object evolution is a basic form of evolution and yet is not supported in most existing object-oriented database management systems. The importance of being able to model multiple and changing object roles has been recognised within the research community and there are now a number of proposed models for supporting some notion of object evolution, for example [GSR96, SLR+92, Pap91, RS91, Ghe90, Sci89]. Some proposals address the specific problem of how objects can evolve when the associated programming language and system do not provide a general mechanism for objects changing their type. For example, [GSR96] presents a solution to object role modelling in Smalltalk. Others, such as [SLR+92] and [Ghe90], propose a general semantic data model with mechanisms for object evolution, but do not address the issue of evolution control.

Proposed mechanisms for evolution control usually involve dynamic modelling of object life-cycles. In some cases, these life-cycles are encoded in object methods and therefore cannot be accessed or updated directly during system operation [Sch90]. Alternatively, they may be encoded either as scripts or rules [BMW84] which, while very powerful in terms of dynamic modelling, of course add an extra layer of complexity to a system.

In this paper, we propose a general model for object role modelling that can be used to provide a level of control over object evolution. The structure of the classification graph in terms of both the

*Proc. 15th Int. Conf. on Conceptual Modelling ER'96, Cottbus, Germany, October 96

types of the nodes and the connection paths between nodes are used to restrict migration paths. While the approach may not be as expressive as other dynamic modelling approaches with respect to modelling object life cycles, it is both flexible and simple and is achieved with little overhead.

Our model for classification structures has three main features that contribute to support for object role modelling. First, the model separates the notions of classification from typing in a way that allows a very flexible means of modelling semantic groupings of objects and different classification views over objects. Second, the model partitions classification graphs into disjoint classification structures thereby distinguishing fundamentally different kinds of objects within a system. Third, we have two varieties of object classifications – kinds and roles – where the former represents fixed forms of classification and the latter represents those that may change over time.

The model presented has been incorporated into the OM object data model [Nor95, Nor93] and has been implemented in the OMS system [Wür95]. In the case of the OM model, two-way relationships between objects may be represented explicitly by an association construct. Classification structures over associations are also supported which means that relationships as well as objects may be classified into various roles. To show the generality of our approach to role modelling, we show how our model for classification structures and evolution control generalises to support the evolution of relationships as well as objects.

This model for classification structures has been used as a conceptual model for the design of systems implemented both on object-oriented and relational database management systems. Further, application systems have been developed on the OMS system directly.

In section 2, we present our basic model in terms of how objects are classified and how these classifications may be related through specialisations to form a set of classification structures. Section 3 extends the model for classification structures to represent kinds and roles and defines a migration operation for object reclassification.

In section 4, we introduce an association construct for modelling relationships between objects and show how the model for classification structures can be generalised to deal with both object and relationship classifications.

Object and relationship evolution, as with other forms of update, can lead to constraint violation. Section 5 discusses issues of constraint maintenance and update propagation specific to the migration operation and the approach currently adopted in our system OMS. Concluding remarks are given in section 6.

2 Classification Structures

Objects are classified according to the roles they adopt within an application system. A particular classification of objects is represented by a set of objects and an associated type definition which specifies the required properties of objects belonging to that set. If C is an object classification, we use $type(C)$ to denote the associated type.

An object belonging to the classification C may have additional properties as specified by one or more subtypes of $type(C)$. The type associated with the classification C may be regarded as a form of constraint that ensures that the classified objects have at least the set of properties specified by the type $type(C)$.

Note that we use the term classification as a general term for a semantic grouping of objects. In other models, the constructs class, collection, type extent or role may be considered equivalent.

However, given the variety of uses of those terms, here we prefer to use a more neutral term since the notions introduced here are not dependent on the details of any particular type/data model.

An individual object may belong to several classifications simultaneously as long as it exhibits the properties required by each of the associated types. This corresponds naturally to the fact that an entity may hold several roles within an application domain simultaneously. For example, a person can be an employee, a manager, a husband and a tennis player. Some roles may be specialisations of other roles. For example, a manager may be a special kind of employee. Other roles, such as husband and tennis player, may be unrelated other than the fact that they are both roles of persons.

The flexibility of object role modelling in our model stems from the clear separation between typing and classification. Typing is concerned with representation, whereas classification is concerned with semantics. While in many traditional data processing applications these may appear very closely related, there are other applications where the distinction is much clearer. For example, in a document management system, documents may be classified according to their semantic content or status. For example, we may have documents classified into `Drafts` and `Publications` as well as being classified according to the projects with which they are associated, e.g. `Project_X_Docs`, or the model they describe e.g. `OM_Model_Docs`. This classification of documents is independent of the form of the document which may be a LaTeX, Postscript or simple ASCII text document. The form of the document would be given by the type with the necessary methods specifying the operations for creating, editing and printing documents of that form. A detailed discussion of the benefits obtained by the separation of typing and classification is given in [Nor95].

Generally, if a classification C_2 is a specialisation of another classification C_1, an object which belongs to C_2 must also belong to C_1. In other words, a specialisation of entity roles implies a subset relationship between the associated sets of objects. It follows that all of the objects in C_2 have the properties defined by $type(C_2)$ in addition to those defined by $type(C_1)$. We can therefore consider $type(C_2)$ to be a subtype of $type(C_1)$. We use $C_2 \preceq C_1$ to denote that C_2 is a specialisation of C_1.

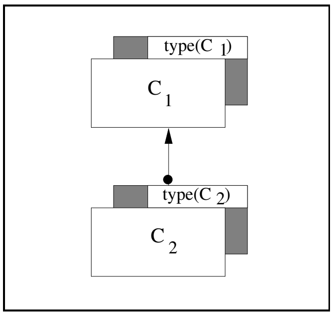

Figure 1: Classification Specialisation

In figure 1, we show our graphical notation for classifications and their specialisations. A classification is represented by a shadowed rectangle. The name of the classification is given in the rectangle and the associated type is specified in the shadow of the rectangle. A declared specialisation $C_2 \preceq C_1$ is represented by an arrow from the specialised classification C_2 to the more general classification C_1.

Object classifications together with specialisations form a classification graph as shown for example in figure 2.

A *classification graph* is a directed acyclic graph $\langle \mathcal{U}, \mathcal{S} \rangle$ where $\mathcal{U}$ is the set of all classifications in the database and $\mathcal{S}$ is the set of edges. If $C_1, C_2 \in \mathcal{U}$, an edge from C_2 to C_1 indicates that C_2 is defined

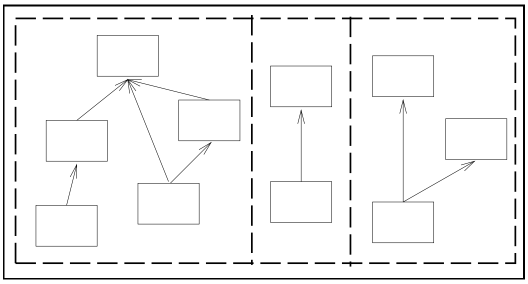

Figure 2: Classification Graph

to be a specialisation of C_1, i.e. $C_2 \preceq C_1$. In general, a classification C can be a specialisation of arbitrarily many other classifications $C_1, C_2, ...C_n$.

A classification graph can be partitioned into a number of disjoint connected subgraphs. For example, the classification graph of figure 2 can be partitioned into three connected subgraphs as indicated by the dotted lines. We refer to a connected classification subgraph as a *classification structure*.

We use this partitioning of the classification graph into classification structures to separate out the fundamentally different kinds of objects within a system. Thus, as shown in figure 3, a system might have objects representing employees and objects representing projects. The classification `Employees` has two specialisations `Programmers` and `Managers`. It is possible to define constraints over specialisations. Here, we declare that `Programmers` and `Managers` are disjoint specialisations which means that no object in `Employees` can be in both `Programmers` and `Managers`. In the OM data model, disjoint, cover, partition and intersection constraints over specialisations are supported [Nor93]. As described, the classifications `Employees`, `Programmers` and `Managers` are part of a single classification structure.

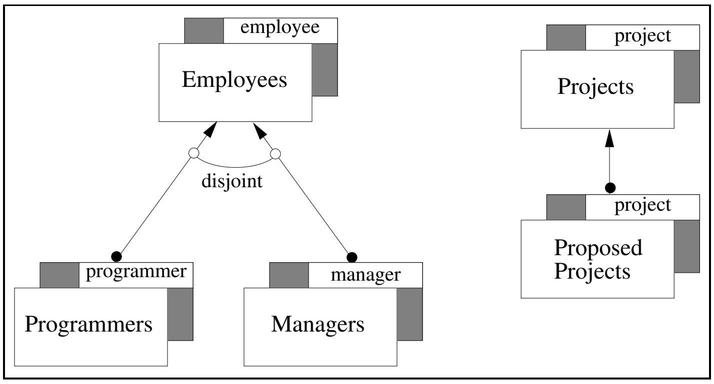

Figure 3: Classification Structures

Figure 3 shows a second classification structure comprising the classification `Projects` and its specialisation `ProposedProjects`. Note that here the types of the two classifications are the same which means that the objects in `ProposedProjects` have no additional properties.

As stated above, each classification structure should be used to model a fundamentally different kind of object and this is then used as a basic means of controlling object evolution. An object can migrate within a classification structure but cannot migrate to a different classification structure.

In this way, an object cannot evolve to a fundamentally different kind of object. For example, in figure 3, an object in `Employees` could not be reclassified in `Projects`.

In the next section, we examine object reclassification in detail and show how a slight refinement of our model for classification structures can be used to provide further controls over object migration within classification structures.

3 Object Reclassification

An object may be reclassified over time according to transitions of the corresponding entities in the application domain. For example, an employee is promoted and becomes a manager capable of leading projects, or, a man marries and is reclassified as a husband rather than as a bachelor. Thus, object reclassification may result in losing as well as gaining membership of classifications.

Some classifications will be more fixed than others. The classification of an entity as a person or document is fundamental and remains fixed, whereas the further classification of persons into trainee programmers is something that reflects a particular role adopted by that entity at a particular point in time and is subject to change. We distinguish those two forms of classification by referring to the former classification as a *kind* and the latter as a *role*. Generally, we then control the reclassification of objects so that an object may change its roles but not its kinds.

Then the set of all classifications $\mathcal{U}$ is partitioned into the set of kinds, $\mathcal{K}$, and the set of roles, $\mathcal{R}$; i.e. $\mathcal{U} = \mathcal{K} \cup \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$.

In [Zdo87], Zdonik introduces a similar distinction by the introduction of *essential types* which are similar in their purpose to our notion of kinds in that they represent the fundamental classification of entities.

Object reclassification may require a metamorphosis of the object from one type to another. For example, the types of `Employees` and `Managers` are not the same. This means that an object migrated to `Managers` will retain all the properties of type `employee` and should gain properties that are specific to type `manager`. In our model, an object can attain the properties of a subtype by means of a *dress* operation. In the OMS interactive system, values for the additional properties can be supplied directly by the user at the time of object reclassification. An alternative approach is to employ some notion of default or null values. In the case of an object migration from `Managers` to `Employees`, an object may remain of type `manager`. However, in OM, we also provide a *strip* operation which can be used to remove type properties from an object when required.

In many object-oriented programming languages, objects have only one type and cannot change types which is a problem for object role modelling. In [GSR96], they discuss the problem in detail for the Smalltalk language and propose a role concept that allows objects to change roles and their associated properties.

We now consider in detail how distinguishing kinds and roles can be used to control object evolution. As stated above, each classification is declared to be either a kind or a role. A classification structure therefore comprises kinds and roles related through specialisation links. A given classification has a context within a classification structure. This context determines the general classification of its objects in terms of the sets of kinds and roles to which they must belong. For example, consider the classification structure for employees given in figure 4. The kinds are shaded black and the roles grey.

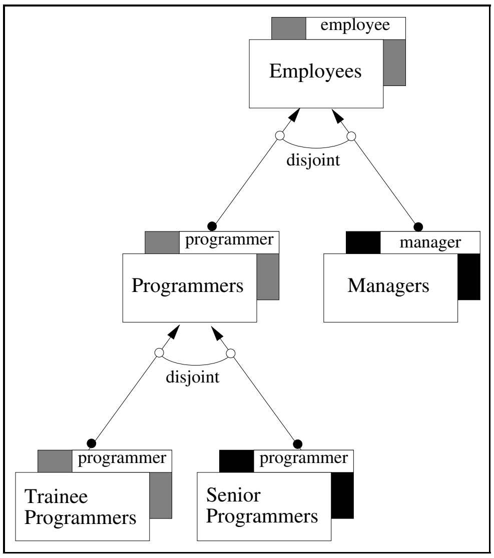

Figure 4: Classification Structure with Kinds and Roles

In figure 4, there are two kinds and three roles such that

$$\mathcal{K} = \{\texttt{Managers}, \texttt{SeniorProgrammers}\}$$

and

$$\mathcal{R} = \{\texttt{Employees}, \texttt{Programmers}, \texttt{TraineeProgrammers}\}$$

If an object is classified in the kind `SeniorProgrammers`, they will also be classified in the roles `Programmers` and `Employees`. We therefore consider the context of `SeniorProgrammers` to be the kind `SeniorProgrammers` (since it is part of its own context) and the roles `Programmers` and `Employees`.

Let $C \in \mathcal{U}$. We introduce the functions *kinds* and *roles* which together give the context of a classification as follows:

$$kinds(C) = \{C_i | C_i \in \mathcal{K} \wedge C \preceq C_i\}$$

$$roles(C) = \{C_i | C_i \in \mathcal{R} \wedge C \preceq C_i\}.$$

Now let us examine the interpretation of the designation of the various classifications as kinds and roles. The classification structure of figure 4 classifies employees within a software company. Since `Employees` $\in \mathcal{R}$, an object can lose this classification and be removed from the associated set. If `Employees` were specified as a kind, it would mean that once an object had been added to this classification, it could never be removed from it.

`Employees` has two specialisations - one designated a kind and the other a role. How should we interpret the situation of a role having a specialisation which is a kind? The interpretation used is that the kind represents a fixed classification in the context of the parent role. For example, if an employee object belongs to kind `Managers`, it must remain in that classification as long as it belongs to `Employees`. Similarly, an object classified in `SeniorProgrammers` can only lose this classification in the event that they are no longer classified in `Programmers`. This models a situation in which once a programmer becomes a senior programmer, they can lose that status only when they are no longer a programmer: they cannot be demoted. However, a programmer in `TraineeProgrammers` can be reclassified in `SeniorProgrammers`. Therefore our classification structure represents a company in which only promotion is allowed: a trainee programmer can be promoted to programmer, a programmer can be promoted to a senior programmer and a senior programmer can be promoted to a manager.

Having outlined the required operation of object evolution by consideration of the above example, we now specify it formally.

Let C_1 and C_2 be classifications and let $x \in C_1$. Then we specify the migration of x from C_1 to C_2 by the operation

$$x :: C_1 \rightarrow C_2$$

This operation is valid in the case that:

1. $\exists C \in \mathcal{U} \,.\, C_1 \preceq C \wedge C_2 \preceq C$
2. $\nexists C \in \mathcal{U} \,.\, x \in C \wedge C \preceq C_1 \wedge C \not\equiv C_1$
3. $\forall C_i \in (\mathit{kinds}(C_1) - \mathit{kinds}(C_2)) \,.\, \exists C_j \in \mathit{roles}(C_i) \,.\, C_j \notin \mathit{roles}(C_2)$

The first condition ensures that an object can only migrate within a classification structure. Thus, it cannot migrate freely within the classification graph but only within connected subgraphs.

The second of these conditions checks that object x is not in any specialisations of C_1. Here, $C \not\equiv C_1$ specifies that C is not identical to C_1. In fact, the dependencies between classifications and their specialisations would ensure that an object could not be removed from C_1 unless it was also removed from all specialisations of C_1.

The third condition specifies that an object migration in which an object loses a kind is valid only in the event that the object also loses a role in terms of which that kind is defined. Thus if an object is deleted from a classification $C_1 \in \mathcal{K}$, it must also be deleted from some classification C_2 where $C_1 \preceq C_2$ and $C_2 \in \mathcal{R}$.

For example, given the classification structure of figure 4, consider the migration

$$x :: \texttt{SeniorProgrammers} \rightarrow \texttt{Managers}$$

We have

$$\begin{array}{lcl} \mathit{kinds}(\texttt{SeniorProgrammers}) & = & \{\texttt{SeniorProgrammers}\} \\ \mathit{roles}(\texttt{SeniorProgrammers}) & = & \{\texttt{Employees}, \texttt{Programmers}\} \\ \mathit{kinds}(\texttt{Managers}) & = & \{\texttt{Managers}\} \\ \mathit{roles}(\texttt{Managers}) & = & \{\texttt{Employees}\} \end{array}$$

Then

$$\mathit{kinds}(\texttt{SeniorProgrammers}) - \mathit{kinds}(\texttt{Managers}) = \{\texttt{SeniorProgrammers}\}$$

but

$$\texttt{Programmers} \in \mathit{roles}(\texttt{SeniorProgrammers}) \text{ and } \texttt{Programmers} \notin \mathit{roles}(\texttt{Managers})$$

therefore the migration operation is valid.

However the migration

$$x :: \texttt{Managers} \rightarrow \texttt{SeniorProgrammers}$$

would not be allowed since

$$\mathit{kinds}(\texttt{Managers}) - \mathit{kinds}(\texttt{SeniorProgrammers}) = \{\texttt{Managers}\}$$

and

$$\mathit{roles}(\texttt{Managers}) \subset \mathit{roles}(\texttt{SeniorProgrammers}).$$

The object x would lose its classification as kind `Managers` but would retain its contextual role `Employees` and the migration would therefore be disallowed.

A migration operation may lead to constraint violations. In section 5, we look at some of the ways in which constraints might be violated and approaches to update propagations to restore consistency.

4 Associations

In this section, we consider an extension to our model to support the direct representation of user-defined relationships between objects. We introduce the relationship classification construct which is used to represent relationships between members of object classifications. A member of a relationship classification is a pair of values representing the related objects. The member type of a relationship classification is a pair type consisting of the source and the target type of the relationship classification.

An association between classifications consists of a relationship classification, associated cardinality constraints and the links to the source and the target classifications. Note that in the OM data model, the source and the target classifications can themselves be relationship classifications [Nor93].

The introduction of relationship classifications is orthogonal to the classification constraints discussed in section 3, therefore allowing the construction of relationship classification structures with migration control as shown in figure 5. As with object classifications, relationship classifications can be divided into kinds and roles.

The notion of classification graph introduced in section 2 also generalises. As before, nodes of the graph are classifications – object or relationship – and edges are specialisations. For any two classifications C_1 and C_2, $C_1 \preceq C_2$ implies that either both C_1 and C_2 are object classifications or both are relationship classifications. Therefore, the classification graph for the example of figure 5 contains three classification structures, two object classifications and one relationship classification as shown in figure 6.

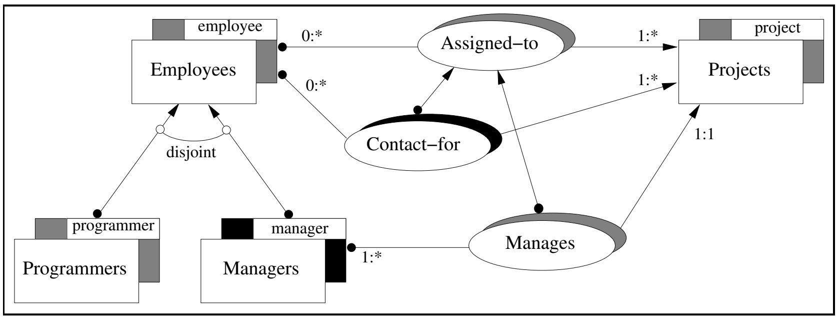

Figure 5: Associations

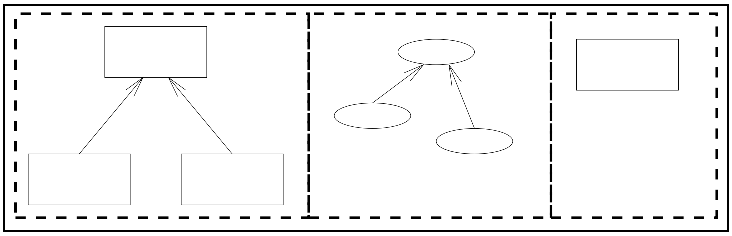

Figure 6: Classification Structures with Associations

Let us concentrate on the relationship classification structure. Employees are assigned to projects. An employee may further be selected as a contact person for a project to which they are assigned. Since contact information is widely spread, employees will remain as contacts as long as they are assigned to the project. To guarantee that constraint, we model `Contact-for` as a kind. On the other hand, being assigned as the manager of a project may change and therefore we model `Manages` as a role.

Migration between relationship classifications is possible according to the same rules given in section 3. For example, we may migrate a relationship object of `Assigned-to` to `Contact-for`. However, migration in the opposite direction from `Contact-for` to `Assigned-to` is not possible since `Contact-for` is a kind.

Migration becomes more interesting in the case where migration in one classification structure effects other classification structures. For example, consider migrating a relationship from `Manages` to `Assigned-to`. This migration would be allowed if we consider the relationship classification structure isolated from the others. However, the migration involves an object in the classification `Managers` and the constraint that a manager must manage at least one project might also require reclassification of a manager object. We will discuss the aspect of migration induced propagation in detail in the next section.

5 Propagations under Evolution

In this section, we consider the issue of constraint maintenance in the case of the migration of objects and relationships within classification structures. A migration operation involves a number of addition and removal operations in classifications and, as with any form of update operation, they may lead to violations of consistency constraints. In the case of constraint violation, the general approach adopted in a particular system may be either to abort the transaction or to propagate update operations and restore consistency. A review of the approaches to constraint maintenance is given in [FP93].

In our system OMS, we adopt the policy of extending the semantics of addition, removal and migration operations to include limited forms of update propagation. For example, as shown in figure 7, when adding an object to a classification, the object is added to more general classifications as well. In the case of removing an object from a classification, it is removed from any specialised classifications automatically. Other forms of propagation may be specified by the user through the use of preference files to explicitly define update propagation operations.

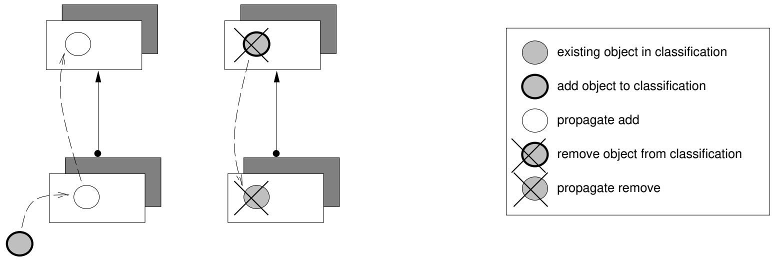

Figure 7: Add and Remove Propagations for Specialisations

We now consider how the introduction of kinds and roles and a migration operation can effect constraint maintenance through update propagation.

The concept of kinds and roles for evolution control forms part of a general constraint mechanism. To show how the specification of a classification as either a kind or a role can influence the way in which consistency is restored, consider the example of figure 8. We illustrate the case of an object being added to the specialisation C_1 when it already resides in the other specialisation C_2, thereby resulting in a violation of the disjoint constraint. To restore consistency, the object must be removed from either C_1 or C_2. If both specialisations are roles, the object could be removed from either C_1 or C_2. In the case of the OMS system, the object will be removed from C_1, i.e. the addition is undone, unless otherwise specified in the preference file. If, however, C_2 is a kind and C_1 a role, these additional constraints would eliminate the choice and force removal from the role C_1.

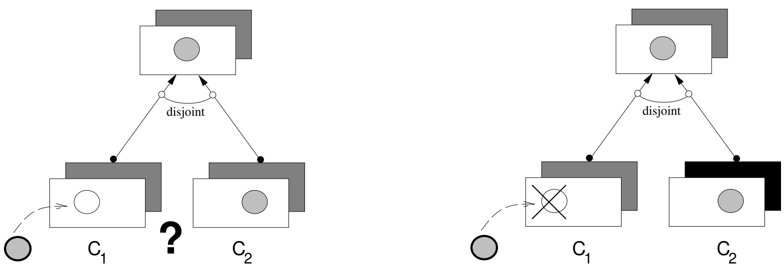

Figure 8: Propagations in Disjoint Specialisations

Associations between classification structures also impose constraints as given by the specification of a source, a target and the associated cardinality constraints. Updates on associations could invoke update propagations in the three directly involved classification structures containing the source, target and relationship classifications.

Recall the example of figure 5 and consider the case of a programmer taking over the management of a project on which they are working. We could migrate the corresponding relationship from `Assigned-to` to `Manages`. However, the source of `Manages` is `Managers` rather than `Employees` and therefore the programmer object would also have to be migrated from `Programmers` to `Managers`. We illustrate the relationship migration together with the resulting propagated migration in figure 9. Generally, the evolution of relationships may invoke the evolution of the objects involved in that relationship.

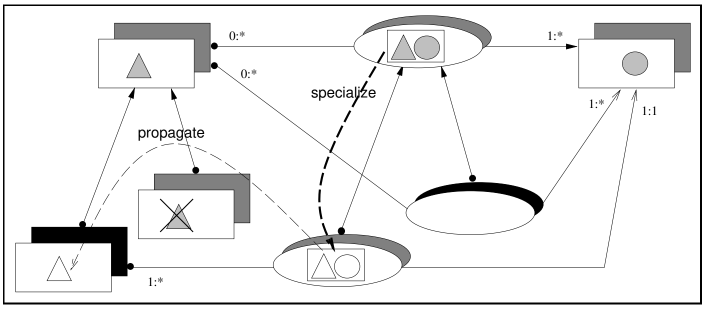

Figure 9: Propagations for Relationship Migration

Alternatively, we could first migrate the programmer object from `Programmers` to `Managers`. In this case, it is not clear which relationship in `Assigned-to` has to be migrated to `Manages` if the programmer is assigned to more than one project (see figure 10). In general, evolving objects in source and target classifications cannot lead directly to propagations in relationship associations.

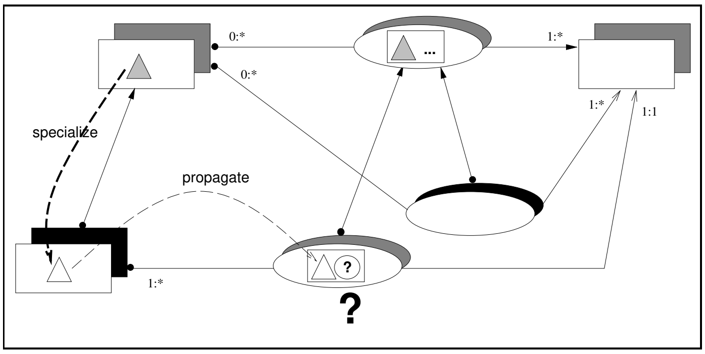

Figure 10: Propagations for Object Migration

At first, one might consider using cardinality constraints to determine the propagation of the migration shown in figure 10 to the relationship classification structure. If a programmer is assigned to only one project and we then promote them to manager, does it imply that they become the manager of that project? It can be dangerous to assume semantics not specified by the user.

As another example, consider the situation where a relationship is removed from a classification `Assigned-to`. In the case where the referenced project object has no further employees assigned to it, the question arises as to whether the update should be propagated and the project object removed from `Projects`. This corresponds to the standard problem of maintaining referential integrity under update. In OMS, the default behaviour would be not to propagate and rather to disallow the removal of the relationship.

Next consider the scenario where someone starts in the company as a manager. If the relationship between the manager and the project they manage is inserted directly into `Manages`, the specialisation constraint causes a propagation which inserts the relationship into `Assigned-to`. Further propagations will cause the referenced manager object to be added to `Managers` and also to `Employees`.

If the relationship in `Manages` later needs to be removed, we cannot simply remove the referenced manager object from `Managers` in the event that the manager has no other projects to manage. The `Managers` classification is a kind and objects can only be removed according to the rules introduced in section 3.

As described in [FP93], update propagations can be triggered either when the updates are executed or when the constraints are checked before committing a transaction. The first possibility effectively extends the semantics of the update operation to include the propagations. As stated previously, this is the approach adopted in our OMS system where we have *enhanced migration operations* which specify the propagations indicated in the above examples.

The OMS system is an interactive system which supports the user by displaying the ways in which objects may evolve. After the selection of an object within a classification, a migration operation will list for selection only those classifications to which migration is allowed. The migration operation handles the necessary type metamorphosis, removal and addition of objects in classifications along the migration path. As described, certain limited forms of propagations will also take place.

6 Conclusions

We have presented a general model for classification structures that can be applied to both object and relationship classification. Classifications can be specialised and it is possible that an object or relationship belongs to many classifications simultaneously. In this way, the model supports a flexible and expressive means of both object and relationship role modelling.

Control over evolution of objects and relationships is achieved in two ways. First, classification graphs are partitioned into disjoint classification structures corresponding to fundamentally different kinds of entities or relationships and evolution is allowed only within a single classification structure. Second, classifications are further classified into kinds and roles according to whether they represent fixed or non-fixed classifications, respectively. In general, an object or relationship cannot lose a kind unless it also loses the contextual role of the kind.

The resulting model for classification structures is expressive in terms of object role modelling and we have used it both for the conceptual modelling of systems developed on other implementation platforms as well as for the development of application systems on OMS. The mechanisms for evolution control are very simple and require little overhead. Also, we have found such structural controls useful for many application scenarios where the life cycles of objects are not well-defined in terms of fixed object evolution paths as would be required in other models. We are currently investigating the different approaches to constraint maintenance under evolution. In particular, we are considering the use of triggered rules as a means of specifying required update propagations to restore consistency.

References

[BMW84] A. Borgida, J. Mylopolous, and H. K. T. Wong. Generalization/ Specialization as a Basis for Software Specification. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling*. Springer-Verlag, 1984.

[FP93] P. Fraternali and S. Paraboschi. A Review of Repairing Techniques for Integrity Maintenance. In *Proc.1st Intl. Workshop on Rules in Database Systems*, Edinburgh, Scotland, 1993.

[Ghe90] G. Ghelli. A Class Abstraction for a Hierarchical Type System. In *Proceedings of 2nd Intl. Conf. on Database Theory, ICDT'90*, LNCS 470, pages 56–70. Springer Verlag, December 1990.

[GSR96] G. Gottlob, M. Schrefl, and B. Röck. Extending Object-Oriented Systems with Roles. *ACM Transactions on Information Systems*, 14(3), 1996.

[Nor93] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *12th Intl. Conf. on Entity-Relationship Approach*, Dallas, Texas, December 1993.

[Nor95] M. C. Norrie. Distinguishing Typing and Classification in Object Data Models. In *Information Modelling and Knowledge Bases*, chapter 25. IOS, 1995.

[Pap91] M. Papazoglou. Roles: A Methodology for Representing Multfaceted Objects. In *Proc. Intl. Conf. on Database and Expert Systems Applications*, 1991.

[RS91] J. Richardson and P. Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proc. Intl. Conf. on Management of Data*, 1991.

[Sch90] M. Schrefl. Behaviour modelling by stepwise refining behaviour diagrams. In *Proc. Entity Relationship Conference*, Lausanne, Switzerland, 1990.

[Sci89] E. Sciore. Extending Object-Oriented Systems with Roles. *ACM Transactions on Information Systems*, 7(2), 1989.

[SLR+92] M. H. Scholl, C. Laasch, C. Rich, H. J. Schek, and M. Tresch. The COCOON object model. Technical report, Department of Computer Science, ETH Zurich, 1992.

[Wür95] A. Würgler. Object model system: An object database management system for the om data model. Master's thesis, Institute for Information Systems, ETH Zurich, 1995.

[Zdo87] S. B. Zdonik. Can Objects Change Type? Can Type Objects Change? In *Proceedings Workshop on Database Programming Languages, Altair, France*, 1987.