Role Modelling: the ASSO Perspective

Donatella Castelli, Elvira Locuratolo
Istituto di Elaborazione dell'Informazione
Consiglio Nazionale delle Ricerche
Via S. Maria, 46
Pisa, Italy
e-mail: castelli@iei.pi.cnr.it, locuratolo@iei.pi.cnr.it

Abstract

This paper shows how a conceptual representation and an efficient handling of roles can be obtained coupling semantic and object models.

1 Introduction

Role modelling is currently an important research topic in the database area since the object-oriented models, which are employed by the more advanced database systems, display serious shortcomings when they are used for this purpose.

This paper discusses the modelling and handling of roles from the perspective provided within the formal database design methodology ASSO.

ASSO supports the development of systems based on two different representation of the same database: a *conceptual schema*, modelled in terms of a semantic data model enriched with behavioral aspects, and an *object-oriented schema* which is a correct and efficient implementation of the previous schema. A schema transformation method, called partitioning method, and a formal theory of refinement guide the definition of the correct object-oriented schema from the conceptual schema. As the two schemas are actually only different representations of the same abstract model, the applications can refer to the high-level and free from implementational details conceptual schema and, at the same time, can be supported by an efficient object system.

This paper discusses how the above approach contributes to the achievement of an high-level role modelling and an efficient role handling. In particular the paper shows that:

- it allows to represent easily and formally roles, role acquisition and role dropping;
- it permits to model in a common framework the static and behavioral properties which characterize roles;

- it offers a context in which constraints on roles can be represented easily and their correctness with respect to the given behavioral specification can be proved;
- it provides an efficient handling of roles on a classical object system.

The rest of the paper is organized as follows: Section 2 introduces the concept of role and discusses how some of the most frequently used database models represent them. Section 3 presents the ASSO perspective. In particular, it shows how role modelling is approached within this context and justifies the correctness of the approach. Conclusions are given in Section 4.

2 Roles and Roles modelling

This section briefly illustrates the concept of role and review the mechanisms which some of the most frequently used database models offer to represent it.

In describing an application we often use the expression "the entity plays the role". In order to explain the meaning of this expression we first need to introduce the concept of *entity*.

An entity is a primitive concept. A set of static and behavioral properties of interest which change through time is usually associated with it. Among these properties there is a property which never changes: the *identity*.

A role can be defined as a set of properties which is interesting to distinguish as a whole. These properties, which regard both the static and the behavioral aspects of an entity, can be shared among different roles.

"The entity plays a role" means that the entity has, among its properties, those which characterize the role.

An entity may play more than one role simultaneously. This means that its properties comprise the properties associated to all the roles it plays. There are, however, roles which are incompatible, in this case an entity cannot play them simultaneously. For example a person cannot be at the same time married and single.

We have said before that an entity can acquire and drop properties through time. This can be rephrased by saying that through time the entity acquires and drops roles. Often role changes are allowed only under certain assumptions. For example, a single gets married under the assumption that he/she is at least sixteen years old. Moreover, not all role changes are always allowed. For example, a single can get married but the viceversa is not possible, or when a person gets married there must be another person, the spouse, that makes simultaneously a similar role changing. The sequences of role changes which an entity perform are usually called *role evolutions*.

Let us now briefly examine how the most important classes of database models represent the concepts of role and role evolution.

Let us begin considering Semantic Data Models(SDM)[7].

The main mechanism of SDM models is the *class*. A class is a sets of elements sharing common attributes. Classes can be related by is-a relationships. If C_1 and C_2 are classes, then C_1 *is-a* C_2 means that C_2 has associated all the attributes of C_1 and, possibly, additional ones. C_2 thus comprises those elements of C_1 which enjoy the additional properties. C_2 is called a subclass of C_1. As a class can have several subclasses, then an entity belongs to all the classes which describe the properties that it enjoys.

Primitive operations are provided to insert and remove elements from a class. These operations are defined appropriately to preserve the inherent constraints of the model. For example, the removal of an element from a class implies an automatic removal of that element from the subclasses.

SDM models usually represent a role through a class which has attributes that describes the static properties associated with the role. Due to the characteristic of the model, the behavioral properties of a role cannot be represented.

An entity plays a role when it belongs to the corresponding class. The acquisition of a role is thus simply represented by inserting the entity into the class which models that role. Viceversa, the drop of a role is modelled by removing the entity from the appropriate class.

The object-oriented models[13, 1] which are currently very frequently used as database models, are based on an object mechanism which defines a set of attributes and a set of operations on these attributes. A class mechanism is also provided which collects all the objects which have exactly the same attributes and operations, i.e. the same properties. As a consequence of this definition an object always belongs to a single class. Classes can be related by a subclass relationships. If C_1 and C_2 are classes, then C_2 *is a subclass of* C_1 means that the objects of C_2 have all the properties of the objects in C_1 plus, possibly, additional properties.

Operations to create a new object as element of a class and to remove it from the class are given.

In an object-oriented model a class usually is used to represents set of roles. The class attributes and operations model the static and behavioral properties associated with the roles they represents. An entity playing a set of roles is represented by an object which belongs to the class which models that set of roles. The acquisition and dropping of a role is modelled by removing the object from the current class and by inserting this object into the class which represent the new set of roles. The inherent constraint which states that an object always belongs to a single class is thus never violated.

The representation of any possible role acquisition with an object model requires the introduction of a class for each possible combination of roles. For example, in order to be able to represent all the role acquisitions which can occur when there are three roles: *person*, *student* and *employee* four classes must be introduced: the class of people which are neither students nor employees, the class of students which are not employees, the class of employees which are not stu-

dents and the class of those people which are both students and employees. The last class must be introduced even if the properties of *employee* and *student* have been completely described by the other classes. A widely recognized drawback of the object-oriented models is that the large number of classes required to model each possible combination of roles renders the schemas very complex. Sometimes this problem is handled by defining a schema in which not all the possible combinations of roles are represented. In several case this solution, however, is not acceptable since the resulting schema is simpler but it is not flexible, i.e. it must be modified every time a change in the applications requires the representation of a set of roles not previously taken into account.

In order to enhance the naturalness in modelling role evolutions, in the last few years several models which introduce an appropriate mechanism to represent roles have been proposed[3, 10, 11, 12, 9, 4]. These models introduce an object mechanism to represent entities. In some models an object can be created with no roles since this object mechanism is independent from the one which represents roles. On the contrary, there are other models in which an object can always created with a specific role since the object and role mechanisms are strictly related.

The role mechanism defines a set of attributes and operations that an object can exhibits during a period of time. Often an *is-a* relationship between roles is also defined. A role R_1 *is-a* a role R_2 means that the latter describes a superset of the attributes and operations described by the previous one. This does not necessarily mean neither that the attributes in common can assume exactly the same values and that the operations in common describes the same behavior. It simply expresses that they have the same name and the same type.

As objects can play several roles simultaneously, the only restriction on the assignment of a role is dictated by the preservation of the integrity constraints, if any. With respect to this point let us note that only few models allow to express constraints on roles.

Primitive operations are given to assign and to drop a role. The dropping of a role sometimes is not allowed since it poses some problems in maintaining the database consistency. When an object drops a role it must also drops all its subroles. In addition, in order not to cause dangling references, the dropping can be done only when the object which loses the role is not referenced with that role.

Summarizing the above review we can observe that:

- SDM models represent naturally role acquisition and role dropping. However, they allow to model only the static properties associated with a role and they do not provide any integrated context to specify integrity constraints.
- The object oriented-models display shortcomings in the representation of role changing since the object-oriented schemas are either complex or not flexible. In addition, again, it is not always clear if and how integrity constraints can be represented.

- The models which introduce a role mechanism offer a more convenient way of modelling roles and role changing. In most of the cases, however, they do not provide an integrated framework to represent all the aspects illustrated at the beginning of this section. For example, constraints on simultaneous role playing and role evolutions, and assumptions on role changing are rarely treated.

In this section we have discussed how the representational mechanisms of the different database models contribute to the achievement of a conceptual representation of roles. Within the database context, however, this is not the only important quality requirement for a database model since much emphasis is also posed on to what extent a model can be supported efficiently by a database system. Regarding this point, it is well known that SDM models have never been implemented efficiently. On the contrary, the object-oriented models have been recognized to be more suitable to support efficient systems.

In rest of the paper we will see how the solution supported by ASSO favors the achievement of both the conceptualness and efficiency quality requirements.

3 The ASSO Perspective

In this section we discuss role modelling from the ASSO perspective.

In defining an appropriate model for database applications, one is always faced with two conflicting quality requirements: the model must be conceptual, since this simplifies the writing of the applications and the model must be simple enough to support an efficient implementation.

As it results from the review given in the previous section, the database models tend to prioritize one of the two requirements. As a consequence, either the applications can be written easily or they are handled efficiently.

ASSO supports the development of systems based on two different representations of the database schema: one which is visible to the applications and the other one which is maintained by the database system. The former is an high level representation whereas the latter an implementational representation. A correct transformation links these two representations, i.e, the behaviour that is exibited by the object model is in accordance with that prescribed by an extended semantic model. Under this perspective, the quality requirements for a database model listed above become: the model seen by the applications should permit a conceptual representation of the database while the model used at implementational level should permit an efficient handling of it.

In order to satisfy these quality requirements ASSO uses a semantic data model, appropriately extended to represent also integrity constraints and behavior, and an object-oriented model.

In the next sections we will see how, with respect to the modellization of roles, these models, when used as suggested by ASSO, permit the achievement of the above database system quality requirements.

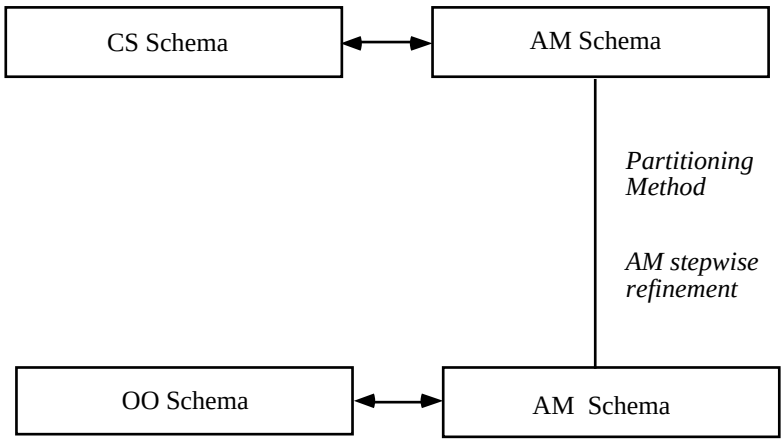

Figure 1: ASSO

Before discussing this point let us briefly introduce ASSO.

3.1 The methodology

ASSO[6] has been defined integrating the Partitioning Method [8] introduced into the database within the formal software development method B-Method[2] B-Method covers all the software development phases from the specification to the implementation. It is based on a single formal model, *Abstract Machine*, which permits to describe both static and behavioral aspects of systems using the same formal framework. A set of tools is associated with this method to help the designer during all the software development phases.

ASSO consists of two stages: *conceptual design*, which focuses on the construction of the *conceptual schema*, i.e. the specification of the database structure and transactions and *refinement*, which is a formal correct transformation from the conceptual schema to the logical schema (see Figure 1).

The conceptual schema is described by two equivalent models: the former, called *Conceptual Schema*, is a semantic data model enriched with modelling mechanisms to represent both integrity constraints and transaction, and the latter is a particular Abstract Machine. The equivalence of these models guarantees that the Abstract Machine theory can be applied to the Conceptual Schema. First order formulas express the consistency the whole specification.

The *refinement* stage consists of two phases: *data refinement* and *transaction refinement*. The *data refinement*[8] transforms step-by-step the conceptual schema into a schema with features of object models. At each step, a new schema equivalent to the previous one is defined. This guarantees both the correctness of this phase and the compatibility of the resulting schema with the Abstract Machine Model.

The *transaction refinement* phase is a particular B-refinement. It defines a sequence of schema transformations from the output of the data refinement to

a formal schema which can be translated easily into a schema supported by an object-oriented system. At each step, the database designer proposes a new schema which reformulates the behavioral definition of the previous schema and proves first order formulas to establish the correctness of the step. By exploiting transitivity, also the correctness of this refinement phase is guaranteed.

As a consequence of the refinement correctness, if the conceptual schema has been proved to be consistent also the object oriented schema is consistent.

3.2 Conceptual representation of roles

Conceptual Schema[5] is formal model which integrates representational mechanisms of semantic data models with appropriate mechanisms to represent integrity constraints and behavior. The main mechanisms of this model are: *abstract sets*, *classes*, *constraints* and *transactions*.

An abstract set, denoted by an identifier written in capitals, is a set made of distinguishable elements which have no internal structure.

A class is a modelling mechanism similar to a semantic data model class. It consists of a both a variable which maintains the class extension, and a set of variables, one for each attribute, that maintain, for each member of the class, the value of the corresponding attribute.

A class defined by is-a relationship is a class whose extension is included in the extension of the parent class and for which additional attributes are given.

Integrity constraints are first order formulas over the class variables. They express properties which must be satisfied in every database state.

A transaction is a transformation over the database state which is defined by the classes of the schema. A transaction specifies, for each state before, i.e. for each possible configuration of values of the class variables, the states after, i.e. new possible values for these variables. Preconditions, which express the assumptions under which the state transformation described is meaningful, are included in the transaction mechanism.

CS transactions are built by means of a basic operator *upd* and a set of transaction constructors. The basic operator specifies a total and deterministic update of the class variables which preserves the class and is-a inherent constraints. The transaction constructors, which for brevity will not be discussed here, allow to combine transactions to express non-deterministic, preconditioned and partial state transformations.

Let us now illustrate by means of an example how roles can be modelled using the representational mechanisms illustrated above.

The example describes an application context in which entities can play the roles *person*, *employee* and *student*. When an entity plays the role *person* a static property *age* and a behavioral property which describes that a person can increment his age by one are associated with it. When an entity plays the role *student* it has the same properties of the role *person*. Finally, when an entity

conceptual schema
example

classes
class *person* **of** *PEOPLE* **with** *(age: N)*
class *employee* **is-a** *people* **with** (*company: N)*
class *student* **is-a** *people*

constraints
$\forall\ e \cdot (e \in employee \Rightarrow age(e) \geq 16)$

initialization
people, age, employee, company, student:= $\emptyset, \emptyset, \emptyset, \emptyset, \emptyset$

transactions

one.more(p) =
upd(*age(p)* = *age(p)+1*)

hire(p,c) =
pre *age(p)* $\geq$ *16*
then upd(*employee* = *employee* $\cup$ $\{p\}$, *company* = *company* $\cup$ $\{(p,c)\}$)
end

fire(e) =
upd(*employee* = *employee* - $\{e\}$)

enrol(p) =
upd(*student* = *student* $\cup$ $\{p\}$)

end

Figure 2: An Example

plays the role *employee*, it has associated the properties of the role *person* and an additional static property which expresses the name of its company. A person can become an employee but only if he is older than sixteen. He can also enroll as student. Finally, an employee can be fired i.e. it can lose this role.

Figure 2 shows a CS conceptual schema which represents the above informal specification[1].

Entities are represented as elements of abstract sets. The elements of this set model quite naturally entities with a proper identity. The abstract set PEOPLE, for example, represents the entities that can acquire the roles *person*, *employee* and *student*.

A role is represented through a class that represent the static properties associated to roles, and a set of transactions that represent the behavioral ones.

The role *person* is modelled by means of the class *person* and the transaction *one.more(p)*. The inherent constraints which define the class *person* express that only the elements of the abstract set PEOPLE can play this role. In addition, they represent that the static property *age* is defined for each element which belongs to this class. The transaction *one.more(p)*, where p is assumed to be a person, specifies that p can have its property *age* incremented by one.

The roles *employee* and *student*, which share properties with the role *person*, have been modelled by exploiting the is-a relationship between classes. The classes which model these roles have been introduced as subclasses of the class *person*. The attribute *company*, specified in the definition of the *employee* class is a property which distinguish the *employee* role from the *person* one.

An entity playing a role is modelled by an element of an abstract set which belongs to the extension of the class that represent that role. Role acquisition and dropping are thus represented by means of transactions which add and remove elements from classes.

In our example, we have modelled the acquisition and dropping of the role *employee* and the acquisition of the role *student*, respectively, with the transactions *hire(p,c)*, *fire(e)* and *enrol(p)*. The preconditions in the transaction *hire(p, c)* express that an entity can acquire the role *employee* only if it is older that sixteen.

The modellization given in the example permits an entity to play all the three roles simultaneously. It would have also been possible, however, to express the incompatibility among roles. This could have been done by adding appropriate integrity constraints. In order to avoid, for example, the simultaneous playing of the roles *employee* and *student* we should have added to the following integrity constraint: $employee \cap student = \emptyset$.

In the example there is no constraint on the possible sequences of role changing. At the current state CS does not comprise yet mechanisms to represent this kind of constraints. We are currently investigating the possibility of expressing

[1]The expression $a << |b$, where a is a is a set and b is a relation, denotes the relation formed from b by keeping only those pairs where the first element is in the complement of a.

them relying first order formulas which express conditions between the state before and after the transaction execution. The following are examples of these kind of formulas.

$\forall\ e \cdot (e \in employee \Rightarrow e \notin student')$

$\forall\ e \cdot (e \in employee \Rightarrow \neg\ (e \in student' \wedge e \in employee'))$

In the above formulas a decorated variable denotes the value of the variable after the execution of the transaction. The first constraint expresses that no entity which has the role *employee* can acquire the role *student*. The second constraint specifies that no entity which has the role *employee* can at maintains this role and, at the same time, acquire the role *student*.

Until now we have discussed how the representational mechanisms of a semantic data model, appropriately extended with mechanisms able to express integrity constraints and behavior, allow to represent roles at conceptual level.

As it has already been explained in the introduction to Section 3, CS model corresponds to a particular subset of the Abstract Machine model and thus it has an axiomatic semantics. This means that both static and dynamic aspects of roles can be represented formally and proofs that expected properties of the specification are met can be done. In particular, it is possible to prove that transactions always preserve the integrity constraints.

This aspect, other than contributing highly to the validation of the specification, offers again a different perspective in which representational problems which arise in other proposals vanish. An example is the dropping of a role. In several models the dropping of a role is considered a problematic state change since it can cause dangling references, i.e. it may happen that an object loses a role when it is still referenced with that role by another object. At running time this can cause errors. In our context this situation can never occur since the proof of the schema consistency ensures that no transaction will never violate the integrity constraints.

4 A correct schema transformation

In the previous section the suitability of the Conceptual Schema model in representing roles has been discussed. This model, however, when evaluated with respect to the features which permit an efficient handling of roles suffers of the same drawbacks of SDM. In order to overcome these drawbacks, ASSO provides a correctness preserving refinement that transforms the conceptual schema into a schema which can be supported by an object model. This transformation is the key which allows the coexistence of the two schemas. The rest of this section describes the refinement stage.

ASSO refinement consists of two phases: *data refinement* and *transaction refinement*.

The data refinement is based on an algorithm, called *Partitioning Algorithm*[8]

conceptual schema
exemple

classes
class *person* **of** *PEOPLE* **with** *(age: N)*
class *employee* **is-a** *person* **with** (*company: N)*

constraints
$\forall\ e \cdot (e \in employee \Rightarrow age(e) \geq 16)$

initialization
$person, age, employee, company = \emptyset, \emptyset, \emptyset, \emptyset$

transactions

fire(e) =
upd ($employee = employee$ - $\{e\}$)

end

Figure 3: A conceptual schema

which transforms the static component of the conceptual schema into a data model in which each object belongs to one and only one class. This phase consists in successive decompositions of the conceptual schema state until a new data model which encloses all the class intersections is obtained. A reformulation of the transactions is also done to render them suitable to work on the new model. The data refinement is performed automatically.

The conceptual schema and the schema resulting from data refinement are two equivalent models, i.e., they specify the same database. However, the handling of the data structures provided by the new representation can be done more efficiently.

The transaction refinement is a particular B refinement. It consists in reducing the transaction non determinism while enlarging the pre-condition until a model which can be easily translated into an object programming language is obtained. The transaction refinement is a stepwise approach. At each step, first order formulas are proved to guarantee the correctnees.

As an example of refinement, let us consider the conceptual schema of Fig. 3.

In this schema, the *fire* transaction is a basic operation and thus the ASSO refinement reduces only to data refinement. From the Partitioning Algorithm the following data refinement relations which link the primed variables of the object schema to those of the conceptual schema can be obtained:

$person' = person$ - $employee$

$employee' = employee$
$age'_p = (person \text{ - } employee) \lhd\!\!+ \ age$
$age'_e = employee \lhd\!\!+ \ age$
$company' = company$

the $person'$ variable is defined as the complement of the set $person$ with respect to the set $employee$ whereas age'_p and age'_e are the restrictions to the subsets $person'$ and $employee'$ respectively of the function age.

By applying the above data refinement relations to the following inherent constraints

$person \subseteq PEOPLE$
$age \in person \longrightarrow \mathrm{N}$
$employee \subseteq person$
$company \in employee \longrightarrow \mathrm{COMPANY}$

of the is-a hierarchy specified by the class constructs in Fig. 3, the following set of properties which define the object schema is obtained:

$person' \subseteq PEOPLE$
$age'_p \in person' \longrightarrow \mathrm{N}$
$employee' \subseteq PEOPLE$
$age'_e \in employee' \longrightarrow \mathrm{N}$
$company' \in employee' \longrightarrow \mathrm{COMPANY}$
$person' \cap employee' = \emptyset$

The object schema in Fig.4 is obtained by reformulating the other components of the conceptual schema.

In order to complete this section, let us evidence that the data refinement is particularly significant when the is-a hierarchy of the conceptual schema state has more then one leaf. In this case, no object model permit the extensions of the two leaves to be not disjoint thus reducing the flexibility in specifying the chances occurring in the real life; moreover to satisfy the above constraint, if an employee becomes a student this employee must be removed from the class student and a new class student-employee must be created.

5 Conclusions

This paper has discussed role modelling within the context provided by the formal database design methodology ASSO. It has illustrated how, changing the approach to the database schema development, a conceptual modelling and an efficient handling of roles can be obtained employing semantic and object models.

conceptual schema
example

classes
class $person'$ **of** *PEOPLE* **with** $(age'_p: N)$
class $employee'$ **of** *PEOPLE* **with** $(age'_e: N, company': N)$

constraints
$\forall\ e \cdot (e \in employee' \Rightarrow age'_e \geq 16)$

initialization
$person', age'_p, employee', age'_e, company' = \emptyset, \emptyset, \emptyset, \emptyset, \emptyset$

transactions

fire(e) =
upd ($employee' = employee'$ - $\{e\}$)

end

Figure 4: An object schema

Within the introduced framework, work can still be done to improve both the conceptual modelling and the handling of roles . As far as the first aspect, we are currently investigating both the problem of associating a behavioral description to the class mechanism and the problem of defining an appropriate extension of the is-a relation to treat also the inheritance of behavior. This would allow to treat the modelling of the static and behavioral properties associated to roles in an homogeneous way.

As far as the efficient handling of roles, we are exploring the possibility of defining a set of equivalence preserving transformations which can be applied both before and after the application of the partitioning method. These transformations, guided by the database designer, should reformulate the schema in a form suitable both to reduce the complexity of the partitioning algorithm and to favor a more efficient search and retrieval of information.

References

[1] Sege Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley Publishing Company, 1995.

[2] Jean-Raimond Abrial. Assigning Meanings to Programs. manuscript, 1993.

[3] Antonio Albano, Roberto Bergamini, Giorgio Ghelli, and Renzo Orsini. An Object Data Model with Roles. In *VLDB Conference*, pages 39–51, 1993.

[4] Constantin Arapis. *Specifying Objcet Life-Cicles*, volume Object Management, pages 197–225. Centre Universitaire d'Informatic, 1994.

[5] Donatella Castelli and Elvira Locuratolo. A formal notation for conceptual schema specifications. In *Information Modelling and Knowledge Bases V*, pages 258–275, 1993.

[6] Donatella Castelli and Elvira Locuratolo. Asso: A Formal Database Design Methodology. In *Information Modelling and Knowledge Bases VI*, Jul 1994.

[7] Richard Hull and Roger King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–259, 1987.

[8] Elvira Locuratolo and Fausto Rabitti. Conceptual Classes and System Classes in Object Databases. Acta Informatica, to appear, 1994.

[9] Barbara Pernici. Objcets with Roles Life-Cicles. In *IEEE-ACM Conference on Office Information Systems*, 1990.

[10] Joel Richardson and Peter Schwarz. Aspects: Extending Objects to Support Multiple, Independet Roles. In *ACM SIGMOD Conference*, pages 298–307, 1991.

[11] Edward Sciore. Object Specialization. *ACM Transactions on Information Systems*, 7(2):103–122, 1989.

[12] Roel Wieringa, Wiebren de Jonge, and Paul Spruit. Roles and dynamic subclasses: a modal logic approach. Technical report, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1994.

[13] Kim Won and et. al. Integrating an Object-Oriented programming system with a database system. In *OOPSALA*, pages 142–152, 1988.