# Approaches to the Specification of Object Associations

*D. Ramazani and G. v. Bochmann*
*Département d'informatique et de recherche opérationnelle*
*Université de Montréal*
*C.P. 6128, Succursalle Centre-Ville*
*Montréal, Canada H3C 3J7*
*Phone: (514) 343-7484, fax: (514) 343-5834,*
*E-mails: {bochmann, ramazani}@iro.umontreal.ca*

**Abstract**:
Many practitioners agree on the key role of object associations during the requirements specification and analysis phases of application development, since they contribute to the definition of the semantics of applications. However, the literature shows that there are multiple semantics for associations, and confusion about how they should be represented. As a matter of fact, various interpretations of the concept of association exist, leading to a multiplicity of representations.

The contribution of this paper is an exposition of four practical approaches to the formal specification of associations. It also introduces a conceptual model for associations which is used as a baseline for comparing the four approaches to formal specification of associations. These four approaches are based on different constructs of the specification language Object-Z which can be used for formally describing associations. The way these approaches capture the requirements represented by associations is central to selecting the approach to be used for the application development.

**Keywords**
*Associations, Formal specifications, Object-oriented modeling, Object interactions, Relationships*

*This work was funded by the Ministry of Industry, Commerce, Science and Technology, Quebec, and the Natural Sciences and Engineering Research Council of Canada under the IGLOO project organized by the Centre de Recherche Informatique de Montreal.*

# 1 INTRODUCTION

The cornerstone of convergence of object-oriented analysis and design methods is the explicit formulation of the semantics of concepts used in these methods. In this formulation, formal specifications play a key role. An important concept of these methods are associations. Based on the following arguments, we claim that special attention should be given to associations between objects. Mili et al. (1990) observed that a number of run-time interactions between objects correspond to enforcing relations. Among the weaknesses of object-oriented methods, Monarchi and Puhr, (1992) report the identification and representation of associations, and the maintenance of a consistent and correct semantics for associations. Associations contribute to the specification of the dynamic behavior of applications. Generally, the complexity of an application is due to complex interactions between its components. As some of these interactions are abstracted by associations, the explicit description of associations allows to manage complexity of applications.

In fact, it can be demonstrated that associations are key to the specification of applications, especially of complex applications (D'Souza, 1993). A quick tour of existing object-oriented methods reveals that associations are neglected during the development of applications (Monarchi and Puhr, 1992). This results in applications which are difficult to enhance, to modify and to reuse (Tanzer, 1995). The problem is caused by the lack of traceability from requirements to the implementation code via the associations expressed in analysis specifications of applications. At least, many practitioners agree on the key role of associations during the requirements specification and the analysis phases of application development, since they contribute to the definition of the semantics of applications. As a result of this role, associations need to be represented and manipulated. Further, there are multiple semantics for associations, and confusion in how they should be represented. As a matter of fact, various interpretations of the concept of association exist (ANSI, 1995), leading to a multiplicity of representations. For instance, associations may be realized as attributes, as separate classes, as operations along with their results, as separate modeling construct, or not at all.

Facing this problem in the context of the definition of standards, the ISO/IEC JTC1 SC21 Working Group 4 proposes a General Relationship Model (GRM) (ISO-2, 1993). GRM provides a framework for specifying semantic properties of relationships independent of how they are represented. It encourages the definition of generic, reusable relationship classes applicable to multiple management applications. To extend the usability of conceptual models for relationships to general applications, Kilov developed a generic concept of relationships (Kilov, 1993).

The work described in this paper was done as part of an ongoing research dealing with the modeling of composite objects. In this research, it appears that object composition can not be successfully handled if we are unable to capture the semantic properties of object associations. This has led to the study of the formal approaches which can be used for specifying object associations. The contribution of this paper is an exposition of practical approaches to the formal specification of associations. Before this exposition, we need to agree on a common base for discussing the concept of association. For that purpose, in Section 2, we present a conceptual model for association. The remaining part of this paper reviews four approaches to the formal specification of associations. An example is used to demonstrate some of the semantic properties of associations. It illustrates the practical usability of the approaches. We close the paper with a review of the kind of requirements captured by each approach.

2 WHAT IS AN ASSOCIATION?

## 2.1 Review of associations in object-oriented analysis and design methods

Object-oriented methods propose two distinct semantics for associations. Some methods describe an association as being a pair of attributes of the associated classes. Each attribute allows to find which objects are associated to the object to which it belongs. The other methods view an association as a relation which is equivalent to a relation in relational databases, i.e. a set of tuples with operations for its management. In addition, we may define association classes which allow associations to have attributes, operations, associations and subtypes. The difference between the two semantics lies in the fact that the first approach considers associations as properties of classes (i.e. attributes) while the second approach identifies associations as autonomous entities. This difference can be summarized by the question whether an association is allowed to exist on its own? In many cases, an association is merely a connection between objects, each participant object using the association for referencing a certain number of other objects. In other cases, we want to act on the association, e.g. by making some queries, adding tuples, etc. One association may imply these two kinds of situations. Therefore, the two approaches complement each other.

## 2.2 Evolution of associations from analysis to implementation

The object-oriented development consists of three phases : analysis, design and implementation. Analysis serves to define the semantic properties of associations. These semantic properties determine the nature (semantics) of the connection between the objects. The semantics is expressed using abstract concepts, i.e. independent of any particular representation. An object-oriented method provides a notation which allows to capture the semantics of associations. In OMT-2, for instance, the semantic properties of associations are its degree, its roles including multiplicity and ordering, its attributes, operations and associations with other objects (Rumbaugh, 1996). The notation which is proposed consists of representing binary associations by lines between associated classes and ternary associations by diamonds with one line path to each participating class. Multiplicity and roles are indicated by text while attributes and operations are represented using the class construct.

During the design phase, the semantic properties of associations are interpreted in terms of object-oriented design artifacts. The interpretation depends on the design decisions taken by the designer. The design notation determines the repertoire of design artifacts. The designer may take five kinds of design decisions. He may choose to interpret associations as follows:

1 Not at all: associations are ignored by the designer.
2 Correlation of behaviors: associations are interpreted as object interactions. This is the case when object interactions may be linked to object associations.
3 Correlation of states: associations are interpreted as structural constraints.
4 Collection of correlated behaviors: In this case, not only the designer wants to interpret object associations as correlated behaviors like in 2, but he wants to act upon all the instances of the association at the application level.

5 Collection of correlated states: Here also the designer wants to interpret object associations as correlated object states like in 3, but he wants to act upon all the correlated states at the application level.

During the implementation phase, design artifacts are translated into constructs of some object-oriented programming language. This representation depends on the constructs provided by the language. For example, considering C++, design artifacts can be translated into object inclusion, pointer structures, operations along with their results, objects, macros, and templates.

In the evolution of associations from analysis to implementation, a formal basis for associations can play a significant role. Semantic properties of associations can be precisely defined allowing a rigorous interpretation of these semantic properties in terms of design artifacts. The formal representation contributes to the ease of understanding, to modifiability and reuse of association specifications (Kilov, 1993). The translation of design artifacts into programming language constructs can be mechanized since formal reasoning is possible and adequate translation rules can be devised. Further, using the formal basis, it becomes straightforward to produce tools, such as implementation generators.

## 2.3 A conceptual framework for associations

### *The model*

An association is the abstraction of a set of constraints between classes of objects. It has a name and a set of instances. An instance of an association is a set of constraints on known object instances. These objects are the participants in the association. The participating objects in an association can be classified according to the type obligations (attributes, operations, associations and behavior) which define the responsibilities assumed by these objects. The set of type obligations which must be met by a participant is a role. Each role has a name. The number of roles of an association is the degree of this association. For each role, the number of objects which assume this specific role in an instance of the association is the role cardinality. Association cardinality is the number of instances of the association in which a given object may assume the same role. The participation of an object in an association is mandatory when the object can not exist without participating in the association with other objects. Otherwise, the participation of the object is optional. It is static when a participant can not be changed without destroying the instance of the association. Otherwise, it is dynamic.

An association may have mathematical properties including reflexivity, symmetricity and transitivity. In addition to its mathematical properties, an association may have application-specific properties. Among these properties we find constraints defined on states and behaviors of the participating objects. These properties are application-specific since they define the semantics of the application. Kilov uses the term business rules of an application to denote such properties (Kilov, 1993). Instances of an association may have state and behavior. The state is captured by attributes which characterize each instance of the association. The behavior consists of operations for querying and changing the state of an instance of the association. These operations should not be confused with the management operations of the associations which consists of operations for creating, deleting and testing the existence of an instance of the association. Similar to objects, associations can be mutually constrained. For example, we may define exclusive associations, derived associations, composed associations, etc.

*Exemplifying the conceptual model*

It is difficult to grasp a conceptual model without an illustration of its usability. We exemplify the model using the employment association. The informal description of this association follows. A person may work for many companies; she is an employee of these companies. A company may employ many persons; it is the employer of these persons. A person is characterized by her name, social security number (ssn), address and age. A company has a name, an address and a budget. When a person is employed by a company, she has a job in that company and she must be aged of at least 18. This job is characterized by a position (title) and a salary. These attributes of the job can be modified. The budget of the company includes all the salaries of its employees. Hiring and firing an employee correspond to creating and deleting an instance of the employment association. We may also check if someone works for a given company (respectively if the company employs a given person). A person who is employed by a given company is not allowed to be enrolled to the unemployment insurance provided by the ministry of labor. Such an insurance has specific terms which can be modified by the ministry of labor. Using the conceptual model, we specify this association as described in Figure 1 which is self explanatory.

## Association definition

| | | | |
|---|---|---|---|
| *name:* employment | | *degree:* 2 | |
| *roles* | | | |
| *role-name:* | *employee* | | *employer* |
| *type obligations:* | with attribute age such that age ≥ 18 | | with attribute budget |
| *role cardinality:* | one | | one |
| *participation:* | optional | | optional |
| *association cardinality:* | many | | many |
| *mathematical properties:* | irreflexive, anti-symmetric, non-transitive, ordered (salary, position) | | |
| *application-specific properties:* | company budget ≥ ∑ salary | | |
| *attributes:* | position, salary | | |
| *behavior:* | change-position, change-salary | | |
| *management behavior:* | hiring, firing, checkworking status | | |
| *constraints with other associations:* | exclusive with the association "enrolled to unemployment" | | |

**Figure 1** Employment association description according to the conceptual model

## 2.4 Comparison with other models for associations

We restrict this comparison to a few well-known models for associations, namely the ODMG object model (Loomis et al. 1993), the Unified method (Booch and Rumbaugh, 1995), GRM (ISO-2, 1995) and Kilov's model (Guttapale et al. 1992).

*ODMG*

In the ODMG object model an association is modeled by a pair of association signatures, each defining the type of the other object(s) involved in the association and the name of a traversal function used to refer to the related objects. Traversal functions are similar to roles. They are declared within the interface definitions of the object types that participate in the association. It means that traversal functions are defined as attributes (or operations) of the participating object types. Generic operations for adding (create, add_member), deleting (delete, remove_member) and testing (exists?) the existence of association instances as well as miscellaneous operations

(traverse, create_iterator_for) are also provided. The ODMG model is grounded on the notion of a mathematical relation. Each tuple of the relation is represented by a pair of traversal functions. Considering the employment association, it can be described using ODMG as shown in Figure 2:

**Employment**
cardinality: many-to-many
traversal functions

| | | | |
|---|---|---|---|
| *signature:* | <employees: set of Person> | *defined in the interface of:* | Company |
| *signature:* | <employers: set of Company> | *defined in the interface of:* | Person |

generic operations: depend on the database system which is used

**Figure 2** Employment association description using the ODMG model

## *GRM*

In GRM, associations are defined using relationship classes. A relationship class consists of roles and behavior. Each role defines the participant type obligations, the role cardinality and the relationship cardinality. Cardinalities have minimum and maximum value sets. GRM also specifies static and dynamic participation of objects. The behavior defines the interactions between roles.

Relationship classes are defined independently of object classes. They define one or more roles, but do not specify the participants that can fulfill these roles. A relationship binding specifies the class of the objects that can participate in that relationship for each role. Relationship bindings are specified independently of the relationship classes. GRM is formalized using GDMO (ISO-1, 1989). To make our presentation concise, we shall assume that the reader has no background in GDMO. Instead, we use an informal notation which is intuitive (see Figure 3).

**Relationship class employment**
roles

| role-name: | *employee* | *employer* |
|---|---|---|
| type obligations: | age ≥ 18 | none of interest |
| role cardinality: | [1, 1] | [1, 1] |
| relationship cardinality: | [1, *] | [1, *] |
| participation: | static | static |

behavior
hiring an employee: creating a new instance of the relationship
firing an employee: deleting an instance of the relationship
mutual exclusion with "enrolled to unemployment" relationship class

**Relationship binding for employment**

| | | | |
|---|---|---|---|
| role | employee | binding | Person |
| role | employer | binding | Company |

**Figure 3** Employment association description using GRM

In GRM, attributes and operations for instances of the relationships can not be defined. Therefore, the constraint imposing that the company budget should includes its employee salaries can not be expressed.

## *Unified method*

In the Unified method, an association represents a structural relationship between objects. The instances of an association are called links. A link consists of a tuple of object references. Each end of the association is a role which may have a name. The role shows how its class is viewed by the other class. Multiplicity consists of minimum and maximum cardinalities of roles. Links may have attributes, operations and associations. These properties are regrouped under an association class which is shown in a class diagram by drawing a dashed line from the association line to a class box that holds the attributes, operations and associations for the link.

Other semantic properties can be defined for a role. Navigability of a role indicates that the implementation should make it straightforward to navigate from a class to another class across the association. Mutability of a role indicates that the link can be modified.

In Figure 4, we have two object classes Company and Person, and a binary association employment with roles employer and employee. Each role has a multiplicity of many indicated by the "*" symbol. In addition, there are attributes and operations for links. These properties are indicated by the association class "Job" within the class diagram. Constraints between participating objects are indicated by text within braces free standing in the class diagram. It is not clear from (Booch and Rumbaugh, 1995) that we may express constraints involving the attributes of the association instances and the attributes of the associated objects. As a consequence, we drop the constraint imposing the budget of the company to includes all its employee salaries. Unemployment and employment associations are mutually exclusive. This is indicated by linking the two associations by means of a dashed line labelled "or".

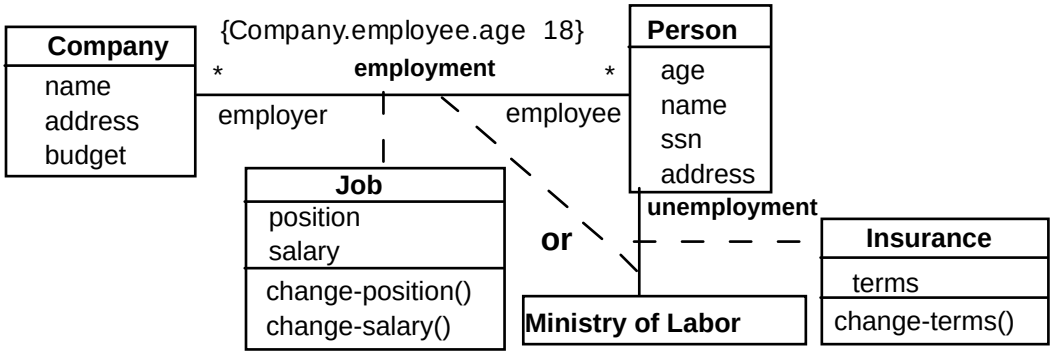



**Figure 4** Employment association description using the Unified method

## *Kilov's model*

Kilov's approach tries to harmonize object-oriented modeling with entity-relationship modeling. Associations are described by its participants, cardinalities, and invariant, as well as the create, read, update and delete (CRUD) operations. These CRUD operations represent the management behavior of the association. The approach postulates the existence of primitive generic associations which are used (refined or combined) to define an association according to the application at hands. The employment association can be defined as a specialization of a relationship association, one of the generic associations provided by Kilov. A relationship association consists of relationship objects. Each relationship object associates several entities. It corresponds to exactly one instance of each of its participating entities. It has properties that provide information about itself, and not information about any of its participating entities. The employment association is pictured in Figure 5.

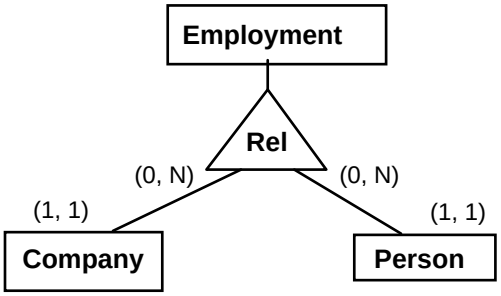



**Figure 5** Employment association description using Kilov diagrams

The employment association is an entity since it has properties. In Kilov diagrams, rectangles indicate entities and triangles represent associations. Cardinality is specified using upper and lower bounds. In this diagram, for one part, an instance of employment is associated with only 1 instance of Company; an instance of Company is associated to a minimum of 0, and a maximum of N, instances of employment. On the other hand, an instance of employment is associated with only 1 instance of Person; an instance of Person is associated to a minimum of 0, and a maximum of N, instances of employment. Kilov diagrams do not specifically indicate relationship roles. Position, salary, change-position and change-salary are specified as properties of the employment entity. The invariant of the association constrains each person to be aged of at least 18. It also states that the company budget includes the salaries of the employees. Hiring and firing employees correspond to CRUD operations.

### *Summary of comparison*

The main features of these models are summarized in the Table 1. The ODMG model is a simple one, but one with a strong mathematical basis since it is based on the concept of a mathematical relation. The model of the Unified method captures more semantic properties than the ODMG model. It considers association instances as objects which may have attributes, operations and associations. On the other hand, GRM subsumes the ODMG model. In GRM, specific modeling features, such as the management behavior of an association, are introduced to reflect the semantics of associations in the context of system management. In addition, GRM decouples associations from object classes to make the representation of associations independent of object classes.

Kilov's approach tries to reconcile GRM with the Unified method. This is done by proposing generic associations which can be specialized to represent associations between types, as well as object associations. The conceptual model proposed in Section 2.3 subsumes all these models. Its main objectives are to be more abstract and to clearly distinguish between the various semantic properties of associations. Our goal in devising this model was to present an intuitive, clear and precise model for associations which can be used as a baseline by practitioners.

| Semantic properties | ODMG | GRM | Unified Method | Kilov's models |
|---|---|---|---|---|
| *name* | no | yes | yes | yes |
| *degree* | only binary (2) | yes | yes | yes |
| *(role) name* | yes | yes | yes | yes |
| *(role) type obligations* | yes | yes | yes | yes |
| *(role) cardinality* | only one and many | yes | yes | yes |
| *(role) participation* | no | yes | yes | yes |
| *association cardinality* | no | yes | no | no |

| | | | | |
|---|---|---|---|---|
| *mathematical properties* | no | yes | yes | yes |
| *application-specific properties* | no | no | yes | yes |
| *attributes* | no | no | yes | yes |
| *behavior* | no | no | yes | yes |
| *management behavior* | generic operations | yes | no | CRUD operations |
| *constraints with other associations* | no | yes | yes | no |

**Table 1** Semantic properties supported by some well-known models for associations

# 3 FORMAL SPECIFICATION OF ASSOCIATIONS

In this section, we describe four approaches to the formal specification of associations using the Object-Z language (Duke et al. 1994). The choice of Object-Z is motivated by the following reasons:

- It is an extension of Z, a well-known formal specification language;
- It has a strong mathematical base for defining relations;
- It is used for the formal specification of OMG, ODP and OSI standards;
- It is a general purpose specification language with built-in object-orientation;
- It is model-oriented, therefore we may express correlations of states and behaviors;
- It has been proven to be expressive enough to specify industrial applications.

In this presentation, there is no emphasis on the formal specification of cardinalities. The reader is referred to (Liddle et al. 1993) which is an excellent tutorial explaining how various notions of cardinality may be formalized. In addition, the formal representation of Kilov's models can be found in (Guttapalle et al. 1992).

Object-Z, as many other object-oriented formal specification languages, does not have a specific construct for representing object associations. In order to explicitly represent associations, the specifier needs to express the associations in terms of the constructs provided by the language. Among the constructs offered by Object-Z, the constructs of attributes, operations, classes and mathematical relations may help for representing associations. This gives rise to four basic approaches to the formal specification of associations. These approaches are described hereafter.

## 3.1 Approach using attributes (or operations)

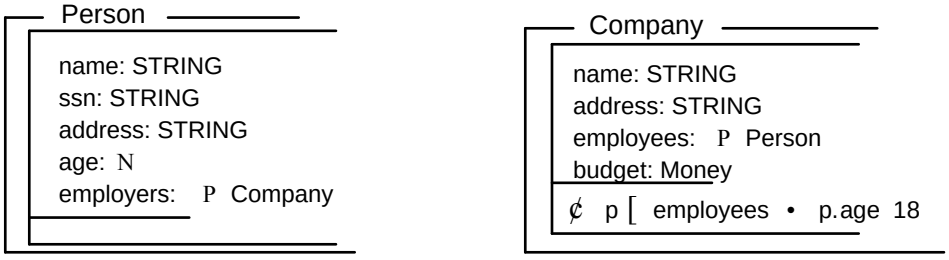
This approach consists of representing associations by attributes (or operations) of the associated objects, as suggested by the ODMG model. The complexity added to class definitions may compromise modifiability and reuse of class definitions, as reported in (Tanzer, 1995).

In this approach, the definition of associations is distributed among its participating classes. Instances of the association are represented by pairs of attribute (or operation results) values. The degree of the association is the number of attributes required for representing a single instance of the association. Roles are captured by attributes. Type obligations, defined for each role of the association, are captured by the type of the attribute representing the role. Role and relationship cardinality are conjointly represented by using attributes which may be references or set of references to objects. Mandatory and optional participations are represented using assertions on the attribute value. To some extent, we may define application-specific properties of the

association using assertions in the class definitions of the participating objects. While the other semantic properties of the association are only expressible in terms of properties of the participating classes.

We illustrate this approach using the employment association. In Object-Z specifications the definition of attributes, within a class definition, consists of two parts, one part is devoted to the types of the attributes and the other part to the constraints on the attributes. In the class Person (see below), the attribute employers represents a set of references to Company objects since the role cardinality is many. Theses references represent the companies the person is working for. In the class Company, the attribute employees represents a set of references to Person objects. These references represent the employees of the Company. The usage of sets implies that the duplication of instances of the association is not allowed. The constraints on these attributes are specified by assertions included in the class definitions of Person and Company. For example, in the Company class, the assertion states that each employee must be at least aged of 18.

*[STRING] [Money] [Position]*



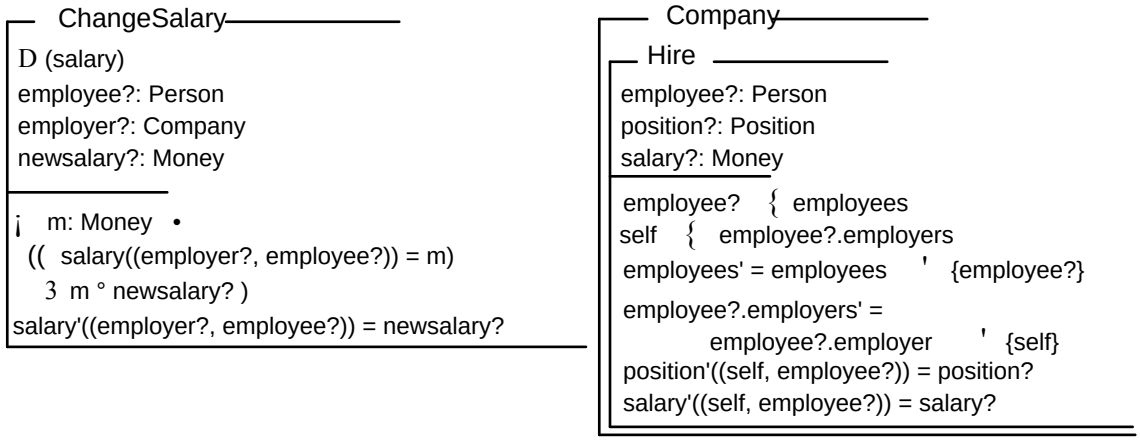

Salary and position are determined by a person and the company she is working for. They can be defined as partial functions which may be declared as global variables or localized within the classes Person or Company.

salary: (Company x Person) Þ Money

¢ p:Person, c: Company• ( (¡ m: Money•salary((c,p)) = m) € c [ p.employers 3 p [ c.employees)

Salary and position are defined by axiomatic descriptions (see above the definition of salary). An axiomatic description introduces one or more global variables in a specification, and optionally specifies a constraint on their values. We use surjections to denote the fact that two employees may hold the same position or they may have the same salary. Position (salary) takes as input a tuple consisting of a Company and a Person. It returns the position (salary) of the person within the company. An assertion links position (salary) to the employment association. It states that for each tuple of Company and Person such that position (salary) is defined, this tuple is an instance of the employment association. Keep in mind that position and salary are attributes of the employment association. Forcing the domain of the partial surjections salary and position to be the employment association means that to each instance of the association corresponds a salary and a position.

Operations consisting of changing the salary and the position are defined as operations affecting the partial functions representing these attributes of the association.




In Object-Z, the definition of an operation consists of two parts, the parameter declarations (input parameters are decorated with ? and output parameters with !), and the pre- and post-conditions for the operation. As illustrated above, the operation changesalary modifies the current value of the salary of an employee of a given company. As input, it takes the company, the person, and her newsalary within this company. Its pre-condition imposes that the salary of a person working for the given company is different from her new salary. Its post-condition guarantees that when the partial surjection salary is applied to the tuple formed by the company and the person, it returns the newsalary.

Operations for managing the association are represented by operations decoupled from the association representation. For instance, hiring an employee is an operation which adds an instance to the employment association. It is declared as an operation of the class Company as illustrated above. The operation hire is included in the definition of the class company, since intuitively a company hires one of its employees. We define the operation fire analogously.

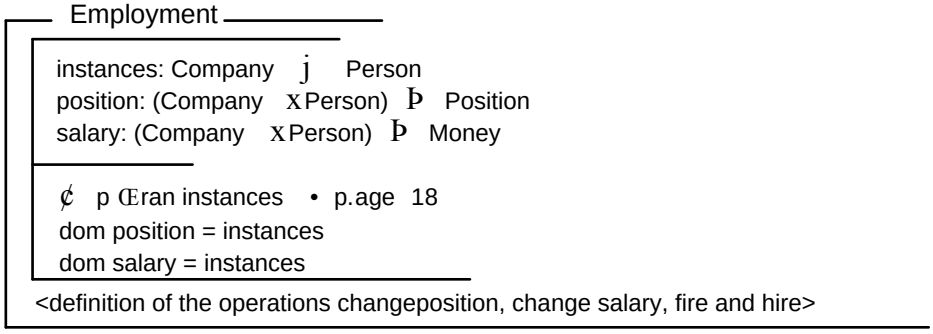
## 3.2 Approach defining the entire association as an object

In this approach, one considers the association as an object having state and behavior. This allows to apply operations to the entire association. Many expressions over the entire association can be written concisely since all the information about the association is gathered within a single object which can be manipulated as a unit. The state of the object representing an association consists of the set of tuples (association instances) and the semantic properties of the association. These elements are expressed in terms of class attributes and assertions on these class attributes. The behavior of the object includes update and query operations. Among these operations, there may be operations for adding or deleting association instances, testing the membership of given association instances, selecting a subset of the association instances according to some condition, and iterating over the set of association instances. In addition, the invariant of the class captures the association invariants and other constraints which need to be maintained between the participating objects.

Here, the definition of associations is localized in a single class definition. Instances of the association are stored using an attribute as a container (i.e. a set of tuples). The degree of the association is represented in the type definition of this attribute. This is also the case for roles, type obligations defined for each role, and role cardinalities. Association cardinality, mandatory and optional participations, and mathematical and application specific properties are represented

by assertions in the class definition of the association. In this approach, like in the previous one, it is difficult to represent attributes and behavior of associations instances. Therefore, we also use the equivalence between association attributes and partial functions. These partial functions are attributes of the object representing the association. Association behavior is represented by operations affecting the partial functions. Management operations for the association are represented by operations of the object representing the association. Constraints with other associations are represented by constraints between objects representing these associations.

As an example, we define below a class Employment representing the employment association. The attributes employees and employers in the class definition allow to write concise assertions on the employment association. For instance, using these attributes we may act upon one kind of the particpants at a time. This is useful when constraining each employee to have an age of at least 18. The tuples of the association are stored in the attribute instances. The partial functions position and salary are introduced in the definition of the class Employment.



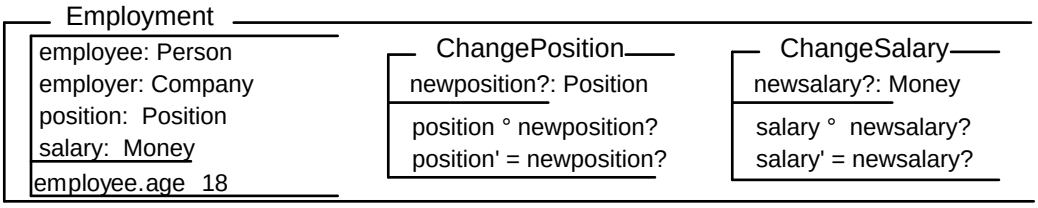

In this case, the operations hire, fire, changesalary and changeposition are declared as operations of the class Employment.

## 3.3 Approach defining each instance of the association as an object

In this approach, one considers each instance of the association as an object having state and behavior. The state of the object representing an instance of the association consists of the objects playing the roles along with other semantic properties. Application-specific properties and attributes of the association are expressed in terms of attributes and invariant of the object. The behavior of the object consists of application-specific operations which can be applied to each instance of the association.

In this approach, the instances of the association are objects. The degree of the association is represented by the number of the attributes required for representing the participant objects in an instance class. Roles, type obligations defined for each role, and role cardinality are captured by the type definition of the attributes representing the participant. Application-specific properties are represented by assertions in the class definition. Association attributes and behavior are made attributes and operations of the object representing the instance of the association. Management operations for the association are represented by operations for creating and deleting objects. However, using Object-Z, these operations can not be represented since Object-Z does not provide operations for creating and deleting objects. On the other hand, when the constraints with other associations are expressed in terms of association instances, they can be represented.

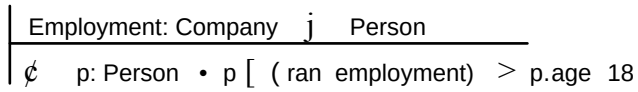
As an example, we define below the class Employment which represents a single instance of the employment association. Each role is represented by a specific attribute. With this approach, the representations of change-salary and change-position operations are intuitive.




## 3.4 Approach viewing associations as mathematical relations

This approach consists of reducing an association to a mathematical relation. Instances of the association are tuples of the relation. The degree of the association is the degree of the relation. The roles, and the type obligations are defined by definitional constraints on the domain and the range of the relation. The role cardinality defined for each role are specified as mathematical properties of the relation. Association cardinality, mandatory and optional participations, and mathematical and application specific properties are also defined in the same way. Association attributes and application specific operations can not be represented. Management operations for the association are represented as operations acting on the relation. Constraints with other associations are expressed in terms of constraints between mathematical relations.

The formal representation of associations with this approach is done using axiomatic descriptions. For instance, the employment association is defined as follows.




The semantic properties of the employment association can be formulated using the artifacts utilized for the previous approaches. However, all the semantic properties of an association can at some level be represented using mathematical relations. Such a description of an association becomes overloaded and for each semantic property there may be many representations. In order to avoid such a level of detail, we limit ourselves to the direct mapping between an association as a set of related objects and a mathematical relation understood as a cross product over domains.

# 4 OBSERVATIONS AND FINDINGS

We do not claim that our review is complete. However, we hope that it provides to practitioners a good starting point for writing formal specifications of object associations. In that sense, this paper is a companion to many existing papers and textbooks which focus on the formal specification of object-oriented applications. As described in Section 3, we may use four approaches to the formal specification of associations. These approaches, due to the constructs of Object-Z on which they are based, can capture certain semantic properties directly, certain other properties only by using some specification artifacts, and may not be able to represent some other semantic properties. This is summarized in the table below. In addition, the last line of the

table indicates which of the well-known models for associations can be formally represented using each approach.

| Semantic properties | Attribute | Association as an object | Instance as an object | Mathematical relation |
|---|---|---|---|---|
| *name* | no | yes | yes | yes |
| *degree* | yes | yes | yes | yes |
| *(role) name* | yes | yes | yes | yes |
| *(role) type obligations* | yes | yes | yes | yes |
| *(role) cardinality* | yes | yes | yes | yes |
| *(role) participation* | yes | yes | yes | yes |
| *association cardinality* | yes | yes | no | yes |
| *mathematical properties* | ordering | yes | no | yes |
| *application specific properties* | no | yes | yes | yes |
| *attributes* | no | no | yes | no |
| *behavior* | no | no | yes | no |
| *management behavior* | no | yes | no | no |
| *constraints with other associations* | no | yes | no | yes |
| *Models which can be captured* | ODMG | GRM and Kilov | Unified method | GRM |

**Table 2** Semantic properties which can be formally captured by the approaches

These basic approaches for the specification of associations can be combined to overcome their respective disadvantages (see Table 3). The following table presents four possible combinations and discusses their advantages and disadvantages. We use the following abbreviations: Attr stands for the approach using attributes, AO stands for defining an association as an object, IO stands for defining an instance of an association as an object, and MR stands for viewing an association as a mathematical relation.

Note that our conceptual model can be fully represented by the combination of AO and IO. A question which any requirements specifier may ask, is which specification approach is more adequate? In other words, what are the criteria for selecting one approach over another? We may opt for the genuine attitude of rating the approaches according to the semantic properties which can be represented. As a consequence, it is difficult to distinguish between approaches which have the same ratings. This raises the question of the relative importance of the different semantic properties. This, in turn, depends on the type of the application. It is up to the requirements specifier to evaluate which requirements are more important, and to select the appropriate approach accordingly.

In general, we would suggest to use the IO approach if needed combined with the AO approach. This recommendation is based on our experience with these approaches as well as the experience of other practitioners. This approach avoids redundancy and it removes any direct couplings between the associated classes. In addition, it leads to complete (with respect to the abstract conceptual model for associations) and reusable requirement specifications .

| Attr and AO | Attr and IO | AO and IO | IO and MR |
|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| *Principle* | Having attributes in the associated classes which reference the association object | Having attributes in the associated classes which reference the instance of the association | Representing each instance of the association by an object and then using another object as a container for the association instances | Defining each tuple as an object itself. The result is quite similar to the combination of AO and IO. |
| *Advantages* | offer many alternatives for writing expressions involving the associated objects | represent some of the semantic properties of associations which can not be captured by only one approach | represent all the semantic properties of associations | • represent all the semantic properties of associations except the management behavior<br>• concise specifications |
| *Disadvantages* | consistency is difficult to maintain | consistency is difficult to maintain | the specification becomes complex | it transforms the mathematical relation into a set of objects |

***Table 3*** *Possible combinations of the four basic approaches*

# 5 CONCLUSION

The importance of associations in the development of applications, especially complex applications, and the key role played by formal specifications in the formulation of semantic properties of associations led us to review how we may formally specify associations. We identified four approaches. In order to ease the comparison of these approaches, we have introduced an abstract conceptual model for associations. As described in Section 2, depending on the way the specifier interprets object associations, some approaches may be appropriate to the task, while the others may simply render specifying an application a cumbersome task. Our experience with the well-known models presented in Section 2 and the different approaches for providing formal specifications reveals that there is still a confusion on the semantics of cardinalities (multiplicities) of associations. We note that (Liddle et al, 1993) give an excellent formal treatment of cardinalities for various models. Attributes of association instances can be represented by means of partial functions. In addition, there is sometimes a confusion between associations and attributes. Embley et al. (1992) have shown that attributes of objects may be represented by associations, and the converse is not necessarily true. In addition, management operations for associations are most of the time ignored, except in database and system management applications.

The main contribution of this paper is the demonstration that although there exists multiple well-known approaches for modelling object associations corresponding to different interpretations, as exemplified by the use of Object-Z, only four basic approaches suffice to formally specify object associations. These four basic formal approaches can be used individually or combined in order to capture various semantic properties of object associations. In addition, the four basic formal approaches can be applied in the context of any model-oriented formal specification language.

# 6 REFERENCES


ANSI-95 (1995) Object Data Management Reference Model. ANSI Accredited Standards Committee X3, Information Processing Systems.

Booch, G. and Rumbaugh, J. (1995) *The Unified method*, Rational Corporation.

D'Souza, D. (1993) Working with OMT in the construction of large systems, *JOOP*.

Duke et al. (1994) Object-Z: a Specification Language Advocated for the Description of Standards. Technical Report No. 94-45, SVRC, The University of Queensland, Australia.

Embley et al. (1992) *Object-Oriented Systems Analysis, A Model Driven Approach*, Yourdon Press/Prentice Hall Englewood Cliffs, NJ.

Guttapalle et al. (1992) *The Materials: A Generic Object Class Library for Analysis*. Information Modeling Concepts and Guidelines, ST-OPT-002010, BellCore.

ISO-1 (1989) Guidelines for the Definition of Managed Objects. ISO/IEC JTC1/SC6 N5402.

ISO-2 (1993) Information Technology - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 7: General Relationship Model, CDC ISO/IEC 10165-7.

Kilov, H. (1993) Information Modeling and Object-Z: Specifying generic reusable associations, Proceeedings of NGIT'93.

Liddle, S. et al. (1993) Cardinality Constraints in semantic data models, *Data & Knowledge Engineering*, **11**, 235-70.

Loomis, M. et al. (1993) The ODMG Object Model, *JOOP*, **6** (3):64-9.

Mili, H. et al. (1990) An Object-Oriented Model Based on Relations, *Journal of Systems Software,* **12**, 139-55.

Monarchi, D. and Gretchen, I., A Research Typology for Object-Oriented Analysis and Design, *CACM*, **35** (9), 35-47.

Rumbaugh, J. (1996) Models for design: Generating code for associations, *JOOP*, **8** (9), 13-7.

Tanzer, C. (1995) Remarks on object-oriented modeling of associations, *JOOP*, **7** (9), 43-6.