

## **Lifetime dependency: An abstraction relation for object-oriented modelling of aspects in the problem domain**

Jens Kaasbøll

and

Renate Motschnig-Pitrik

Department of Informatics,  
University of Oslo  
P.O.Box 1080 Blindern, N – 0316 Oslo,  
Norway  
E-mail: jens.kaasboll@ifi.uio.no  
WWW: <http://www.ifi.uio.no/~jensj>

Institute of applied computer science and  
information systems, University of Vienna  
Rathausstrasse 19/4, A – 1010 Wien,  
Austria  
renatem@ifs.univie.ac.at  
<http://www.pri.univie.ac.at/~renatem/rm.html>

### **Abstract**

*Techniques supporting early phases of the software life cycle such as requirements analysis or conceptual modelling aim to produce both easily comprehensible and truthful models of some problem domain. Therefore, such techniques depend on a proper choice of highly expressive concepts that can easily be acquired and captured throughout the process of analysis. One deficiency encountered in current analysis techniques is the lack of proper means to express inter-object relationships that result from dependencies of individual objects' life cycles.*

*This paper introduces a relationship called lifetime dependency that captures frequently occurring behavioural constraints and shows its modelling benefits. It is argued that incorporating lifetime dependency into object-oriented analysis models both improves their expressiveness and helps to keep the number of special purpose constructs such as role - and relationship objects small.*

*The transition between analysis and design of lifetime dependency is discussed, while implementation is not considered. Guidelines for analysis are provided.*

### **1 INTRODUCTION**

When developing an information system whose data represent a problem domain, a common way of working is first to make a model of the problem domain (eg, a data model or an object-oriented model), and then to implement the model in a programming language. This paper concerns problems in the first step; modelling the problem domain.

"The objects are there just for picking" (Meyer, 1988). This quote indicates that finding the objects in object-oriented analysis is straightforward because the objects one would like to define in the model correspond to an obvious partitioning of the problem domain that is to be modelled. Many textbooks (eg, Martin and Odell, 1992; Coad and Yourdon, 1991) and articles (eg, Sciore, 1989; Richardson and Schwarz, 1991) explicitly or implicitly assume a one-to-one correspondence between an object and a separable piece of physical matter. This assumption leads to three problems of modelling:

**Person-roles:** When a person plays several roles (eg, an author role, several referee roles, and a participant role in a conference), all these roles have to be modelled as aspects of one object.

**Symbolic substance:** When modelling a symbolic inscription (eg, the recording of a movie) and the substance carrying the inscription (eg, a video-cassette), the movie and the cassette have to be modelled as one object.

**Relation with attributes:** A relation between entities (eg, a marriage) cannot be modelled as an object, because two separable pieces of physical matter are required at each side of the relation, and the relation itself has no distinct physical properties.

These problems have been remedied in three major ways:

- Specific kinds of “items” are defined:
  - link objects (Martin and Odell, 1992) to capture relations with attributes
  - aspects (Richardson and Schwarz, 1991) and roles (Gottlob et al, forthcoming) to model a person with several roles.
- Specific kinds of relations are suggested:
  - role relations (van de Weg and Engmann, 1992) to model person-roles
  - materialisation relations (Goldstein and Storey, 1994) to model symbolic substance.
- Existing definitions of objects and relations are used in specific combinations:
  - patterns of aggregation and generalisation (Coad, 1992) and
  - multiple generalisations (Goldstein and Storey, 1992)to model roles of persons, companies, and other objects that can play roles.

However, these proposals bring forth new difficulties. The specific kinds of items introduce a concept similar to, but in addition to objects, and each of the solutions only solves one of the modelling problems. The specific relations each handle a modelling problem, but none covers all the three problems. The patterns do not capture all the semantics, and the multiple generalisations create problems of migrating classes.

We will argue that the three modelling problems can be solved when regarding physical matter as a necessary condition for the existence of more abstract aspects. We define a type of **abstraction relation** called **lifetime dependency** based on this assumption, and we demonstrate how lifetime dependency is capable of modelling the three cases (Section 2). In Section 3 we show that lifetime dependency differs from the well known abstraction relations classification, generalisation, and aggregation. Thereafter we show that lifetime dependency also covers the semantics of all the solutions proposed by others (Section 4). Guidelines for modelling are given in Section 5.

## 2 DEFINITION

When determining whether something should be modelled as an object or not, we state that an object either has a **life cycle** of its own or **properties** of its own.

Determining the life cycles and properties depends on the purpose of modelling. When making an administrative system for a conference, this view enables to say that

- a person is an object, because the person has a life cycle: comes to existence, changes her or his address, acquires skills, dies
- an author of a paper is an object, because an author has the life cycle start writing, submit first version, receive referee judgements, revise, ...

When modelling for a video rental shop,

- the recording of a movie is an object because it has the properties title, length.
- a video-cassette is an object because it has the properties length (may differ from the length of one movie) and exemplar number, and the life cycle: acquired in the shop, rented, returned, rented, returned ...

When modelling kinship relations,

- a marriage is an object because it has the life cycle: wedding, then either widow(er) or separating (alternatively joining), divorce

## 2.1 Lifetime dependency between objects of analysis of problem domains

During analysis of problem domains, we study the part of the world that the data in the computer is going to represent. The objects resulting from this analysis are therefore called **objects of analysis** or “real world objects” with a more colloquial expression.

When identifying an object according to the criteria above, it may seem that the physical matter of the world to be modelled is of no importance. That would contradict our common experience, which tells that when a person dies, then the author disappears from the surface of the earth, and the marriage where she or he is a partner is dissolved. Accepting the need for expressing that some objects depends of the existence of others, we define

When the existence of a real world object (the **dependent**) throughout its lifetime depends on the existence of other objects (the **support**), we say that the dependent stays in a **lifetime dependency** to the support.

The dependent thus comes into existence after (or concurrently with) the objects upon which it exists, and it disappears before (or concurrently with) the first of the other objects that disappears. Consequently, the dependent can be supported through **one and only one lifetime dependency relation during its existence**. This relation may include several support objects. Consequently, also these support objects cannot be substituted.

As a rule of thumb, the physical matter is the support, while the abstract aspects are dependent. An author object depends on the existence of a person through the whole life cycle of the author. The marriage is dependent on the existence of two persons through the whole lifetime of the marriage. The recording of a movie on the tape is dependent of the existence of the cassette; if the cassette is removed from the shop, the movie will not be available for rent any longer.

Existence dependency throughout the whole lifetime is required. If a wheel is produced by a manufacturer, the manufacturer has to exist at the time of production. However, the manufacturer may go out of business, and the wheel may still exist, implying that there is no lifetime dependency between wheel and manufacturer.

We will use the notation illustrated in Figure 1 – 4

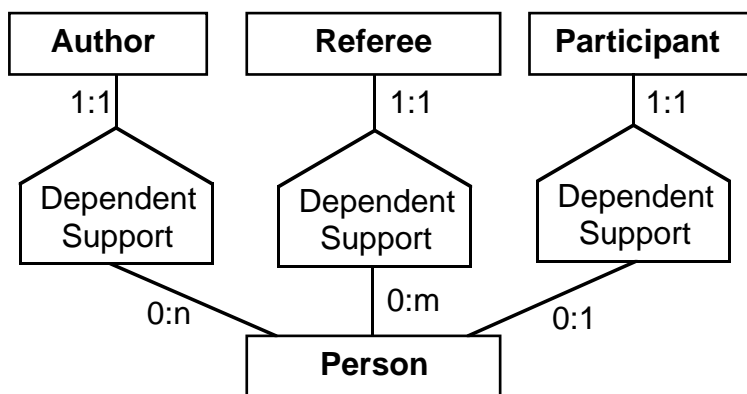


Figure 1: Person with roles. Cardinalities means: Author depends on exactly one Person, while Person can support any number of Authors

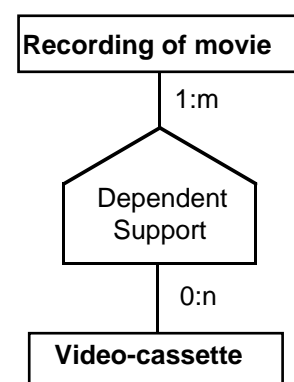


Figure 2: Symbols and substance

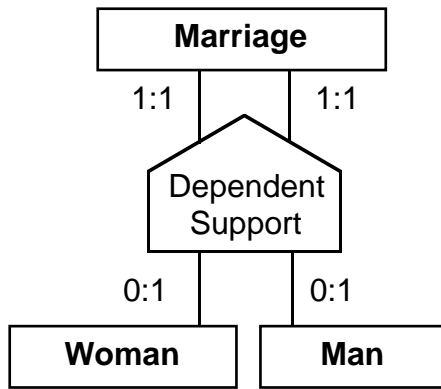


Figure 3: A relation with attributes. Both support objects are related to the dependent through the same relationship.

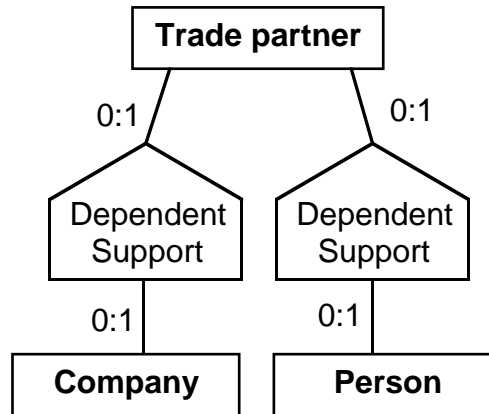


Figure 4: A dependent supported by either one object or the other

The cardinalities shall be read like the following: One author depends on one and only one person, and one person can support between zero and n authors. Because one movie may require several video-cassettes, and because one cassette may contain the recording of several movies, there is a many-to-many relation in Figure 2. If one of the support objects (the cassettes) ceases to exist, the dependent (the recording of the movie) will also disappear.

Because all the supporting objects have to be present during the lifetime of the dependent, the notation in Figure 3 should be read like this: the marriage depends on one and only one woman and one and only one man, each woman can support zero or one marriage at a time, the same for a man.

Because of the lifetime dependency, one may think that the minimum cardinality on the dependent side is 1, and that this cardinality constraint expresses the lifetime dependence relation. However, if a dependent can be supported by an object of one type or an object of another type, the minimum cardinality is 0, illustrated in Figure 4. It is shown in Section 2.3 that minimum cardinalities of 1 has a different interpretation.

When a model shows that a dependent class can be supported through more than one lifetime dependency relation, this means that each object of the dependent class is supported by either of these relations.

## 2.2 Lifetime dependency between design objects

Through an analysis of a problem domain, we may encounter lifetime dependency relations, eg, that marriage is lifetime dependent on both wife and husband. However, when designing the objects that are going to exist in the database, and that should represent the real world objects, the **design objects** (also “database object”) in the computer may have a lifetime that differs from those found through analysis of the real world.

When a marriage is terminated, one may still want to keep the data about the marriage in the computer, eg, for future statistical use. This implies that there is an object in the database (d-marriage) that represents a currently non-existing object of analysis (a-marriage), which during its lifetime depended on the existence of other objects of analysis (a-wife and a-husband). Then two alternatives for the definition of lifetime dependency between computer-objects exist: Either to allow for the existence of d-marriage without the existence of d-wife

and d-husband, or to demand the continuing existence of d-woman and d-husband for the d-marriage to continue its existence. To keep the lifetime dependency relation as simple as possible, we choose to demand that

when there is lifetime dependency between objects of analysis, lifetime dependency is also required between the design objects in the database that represent the problem domain.

However, the objects in the database are allowed to exist also when the real world objects have disappeared. In order to keep the consistency between database objects and real world, the database objects are not allowed to be changed after their real world complements have disappeared.

Corresponding to the after-the-existence period, there may also be a period of planning prior to the existence of the real world objects. Eg, one would like to make reservations for a flight before the actual travel, or one may plan the marriage during a period of engagement. Since there will be lifetime dependency between real world objects when the plans are carried out, we will require dependency between the database objects also before the existence of real world objects.

The design dependent thus comes into existence after (or concurrently with) the design objects upon which it exists, and it disappears before (or concurrently with) the first of the other design objects that disappears.

### **2.3 Existence dependency in data modelling**

The concept of existence dependency in data modelling differs from lifetime dependency. Eg, in data modelling, one would probably have said that for every wheel, there must be a manufacturer, hence the existence of the wheel depends on the existence of a manufacturer (Navathe, 1992; Nijssen and Halpin, 1989). This implies a "total" or "existence dependency" restriction between the wheel and the manufacturer in the data model, and a corresponding minimum cardinality constraint of 1 manufacturer per wheel. However, the existence dependency of data models concern the entities in the database (the design objects), not the wheel and the manufacturer in the real world (the objects of analysis) that the database is supposed to represent. The manufacturer-object in the database may continue its existence after the real world manufacturer is out of business.

While existence dependency in data modelling may appear between real world entities that have no common physical properties, lifetime dependency occurs when the dependent shares physical properties with the supporting objects or when the dependent can be said to constitute aspects of the supporting objects.

### **2.4 Other approaches**

Relations similar to lifetime dependency have been proposed as abstraction relations in previous research. van de Weg and Engmann (1992) include a "role" relation in their conceptual framework, which also consists of generalisation, whole-part, and grouping of set members. However, they indicate that persons are more abstract than the roles they play. Kaasbøll (1995) argues that this runs counter to the intuition that roles (the dependent) are abstract patterns of behaviour while persons (the support) model the less abstract, physical entities. In general, aspects like roles, symbols, and relations with properties are more abstract than the phenomena to which the aspects are connected. Because the support, possibly through several layers, inevitably has to rely on physical matter, the support side will

be the more concrete one in the relation, while the dependent side is the abstract one. Therefore, the lifetime dependency relation is considered an abstraction relation.

Some suggestions for “role” abstraction relations in the literature resemble lifetime dependency. Since lifetime dependency is a special kind of abstraction relation, it builds on these approaches.

van de Weg and Engmann (1992), also referring to Pernici (1990) use an example of a student being a role of a person, similar to the IFIP case in Figure 1. They state that a role “can be created after the creation of the base type” (p.136). This is not the same as lifetime dependency, which requires that the role *must* be created after the person (“base type”). Regarding the termination of roles, they require the termination of the role if the base terminates, similar to lifetime dependence.

They argue that roles and bases are different entities (objects) with their own attributes. However, they indicate that a role object can have only one base object (p.130), which inhibits modelling of the cases in Figure 2–3. Besides, they indicate that the base is more abstract than the role, which runs counter to the view of abstraction presented here.

Velho and Carapuca (1994) also propose a role relation which seems to allow for any cardinalities. However, they do not seem to capture the dynamics of object life cycles, thereby also modelling events as objects and missing the lifetime dependency issue.

Kaasbøll (1995) defines a role-realisation relation between objects, with the roles depending on the existence of the realisation. He refers both to person-role and symbolic-substance relations (Figure 1 and 2). However, he neither discusses the problem of several realisations (Figure 3) nor the difference between lifetime dependence in analysis and design.

We therefore conclude that lifetime dependency is more precisely defined than related, previous approaches.

### 3 COMPARISON WITH COMMONLY KNOWN ABSTRACTION RELATIONS

In the previous section it was argued why lifetime dependency cannot be expressed by restrictions on cardinalities and that it is an abstraction relation. Here we will show that it differs from the common abstraction relations of classification, generalisation, whole-part, set-members.

Generalisation is a relation between class types, and classification is a relation between a class type and the objects belonging to this class. Lifetime dependency is a relation between objects, so it differs from generalisation and classification.

Like lifetime dependency, whole-part is also an asymmetric relation between objects. Thus we have to show the difference between the two relations both when the dependent is selected as the whole and when the support is regarded as the whole.

Assume that the dependent is the whole, eg, woman and man are parts of marriage. If one of the parts disappear, the whole disappears as well, because of the lifetime dependency. This is counter to common definitions of whole-parts (Smith and Smith, 1977), where at least some of the parts of a whole may be removed without the whole to disappear. Eg, in a car repair shop, if removing a wheel from a car, the remaining item is still considered a car.

Assume that the support is the whole, eg, authors, referees, and participant are parts of person. If the whole disappears, so will also the parts, again because of the lifetime dependency. This is also counter to common views of whole-part (Smith and Smith, 1977), where a part can exist without being attached to a whole. Eg, the wheel exists also when removed from the car.

Since both ways of comparing lifetime dependency with whole-part failed, we conclude that lifetime dependency differs from whole-part.

The difference between set-member and lifetime dependency can be established through similar arguments as for whole-part, because both a set and its members are independent of the existence of the other (Motschnig-Pitrik and Storey, 1995).

We therefore conclude that lifetime dependency is an abstraction relation in addition to classification, generalisation, whole-part, and set-members.

## **4 SOLUTIONS FROM THE LITERATURE**

We will show that lifetime dependency capture the semantics of various modelling proposals found in literature. Some of the approaches combine well known types of relations (section 4.1), while others define new types of objects (section 4.2) or relations (section 4.3).

### **4.1 Existing relations**

#### **Multiple generalisation**

Early attempts to capture person-roles were carried out by means of generalisation/specialisation (Sciore, 1989; Goldstein and Storey, 1992). The models that were created consisted of multiple generalisation hierarchies between object types, leaving the options for as many combinations of object types as possible.

This approach results in five problems:

- The model becomes very complex and hard to understand.
- The combination of generalisation hierarchies leads to frequent occurrences of multiple inheritance, requiring additional rules for handling priorities in name conflicts.
- When a combination of object types that was not anticipated has to be included in the model, the complex web of hierarchies has to be changed, with a great risk of failure.
- Even if all combinations of object types were captured, the model cannot handle cardinalities, because the combination of objects is carried out between the object types, and not between instances, as argued for generalisation in section 3.
- When an object changes its roles, the object has to be transformed into another class, creating migrating class problems.

#### **Patterns**

Coad (1992) proposes seven patterns of combinations of generalisation-, aggregation-, and association relations for modelling phenomena that often occur in analysis. One of these is the person-role case, which he proposes to model as in Figure 5.

Compared to the multiple generalisation approach, the combination of generalisation with whole-part avoids the need for multiple inheritance and migrating classes, and it allows for specifying cardinalities. However, compared to lifetime dependency, the cardinalities cannot be given for each of the roles, the existence dependency is not expressed, and the whole pattern indicates erroneously that person is more abstract than the roles.

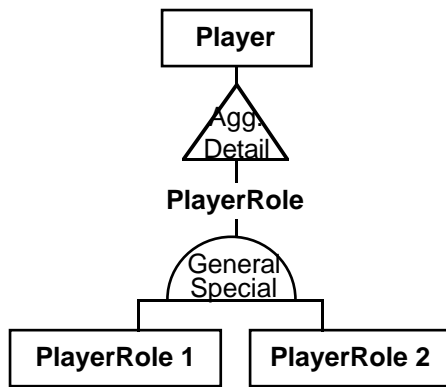


Figure 5: Pattern for modelling roles (Coad, 1992)

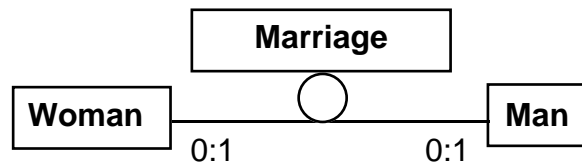


Figure 6: A "link-object" (Martin and Odell, 1992)

### Overlapping sub-types

Binary relationship data modelling has a mechanism called overlapping subtypes (Nijssen and Halpin, 1989, p.134) that resemble multiple inheritance. A model may, eg, specify that person has the subtypes teacher and student, and that these subtypes may overlap such that a person may be both a teacher and a student at the same time. However, since subtypes is a specialisation relation between object types and not between objects, overlapping subtypes have the same limitations as multiple generalisations.

### 4.2 Special kinds of items

“Link-objects” have been introduced in object oriented analysis (Martin and Odell, 1992) to model attributes of associations, see Figure 6.

A link-object is tied to a link between two objects, and therefore it exists as long as the link exists, and the link exists as long as each of the objects in the ends of the link exist. The link-objects therefore depends on the existence of both the objects connected by the link for its whole lifetime. This is the same as saying that the link-object is lifetime dependent on these objects. Therefore, lifetime dependency covers the semantics of link-objects.

Since a link connects two objects, while the lifetime dependency relation can connect a dependent to any number of support objects, lifetime dependency is a more general relation than link-objects. Eg, link-objects cannot be used in the person-role or in the symbolic substance cases.

Binary relations with attributes in entity-relationship modelling (Navathe, 1992) have the same semantics as link-objects. Lifetime dependency can therefore also replace binary relations with attributes.

ER-relations with more than two entities is a more complex case for modelling (Flynn et al, 1995). When all cardinalities in the relation are minimum one, the relation can be modelled as a dependent object with the entities as support. When one of the entities have cardinality zero, this entity cannot act as support for the relation. How to model this case requires a discussion that will not be pursued in this article.

The approach by Reenskaug et al (1992) has an analysis phase in which each role is described as an object on its own. The objects are connected by paths with cardinalities. Both lifetime dependency cases, eg, person - participant, and any other relation between

objects, eg, author - paper, are described through two roles connected by a path. These are distinguished, thus no semantics similar to lifetime dependency is captured. In the synthesis phase, Reenskaug et al (1992) say that roles can be merged into one object. No clear rules for synthesis are provided; it is left to the discretion of the analyst to merge, eg, person and participant into one object. If merging person and participant, and the person continues to exist after finishing as a participant, the model would fail to represent properly.

Techniques for modelling an object with many “roles” (Gottlob et al, forthcoming) or “aspects” (Richardson and Schwarz, 1991) have also been suggested. While the roles approach allows an object to connect several instances of one role-type, the number of aspects of each type connected to an object is restrict to one. Therefore we will consider the roles approach, and its shortcomings will also be valid for aspects.

The roles approach can model the person with roles case in Figure 1. However, each role-instance can only be attached to one object, disabling roles to model the symbolic substance and the relation with attributes cases in Figure 2–3.

The restriction to one object has its benefits when defining mechanisms for the roles to transfer requests they cannot handle themselves. The roles can transfer the requests in the direction of their one and only object. The dependent object in the lifetime dependency relation may have to transfer the request to all its supporting objects.

### **4.3 Special kinds of relations**

#### **Materialisation**

Goldstein and Storey (1994) suggest a specific “materialisation relation” to capture one-to-one relations between a symbolic inscription (eg, a recording of a movie) and the substance carrying the inscription (eg, a video-cassette). However, their relation does capture many-to-many relations that appear when several movies are recorded in one cassette or several cassettes are needed for one movie.

#### **Representation**

Representation is an aboutness relation; a text or another expression is about phenomena that constitutes the extension of the expression. A criterion for deciding whether there is a representation relation is that there exists an expression that can be separated in time and space from the phenomenon that the expression is about.

Some examples found in literature indicate that lifetime dependency or special types of this relation (role, materialisation) is confused with the representation relation. For example, (Coad, 1992) an aircraft description represents an aircraft, the description does not play the role of an aircraft, because the description and the aircraft can exist independently of each other; (Goldstein and Storey, 1994) the description of Golden Gate represents Golden Gate, Golden Gate does not materialise its description, because the description and Golden Gate exist independently of each other.

## **5 GUIDELINES FOR ANALYSIS OF PROBLEM DOMAINS**

When deciding when to use the lifetime dependency relation in modelling, one can both look for the dependency and for concepts that are commonly used for expressing relations that can be subsumed under lifetime dependency.

When searching for dependency, the following questions can be asked:

- Which physical entities are involved?
- Is there an aspect that depends on the existence of the physical entities?
- When do the aspect appear and disappear?

These questions may unveil some of the person-role relations and the symbolic substance relations. However, the relations with attributes may not easily be found, because the relations are seldom regarded as aspects. In order to find these occurrences, one should check whether the relations have their own attributes or life cycles.

When the generalisation relation is used, one should ask:

- Do the specialisation classes have shorter life cycles than the general?
- May an object shift from one class to another during its lifecycle?

In both cases, the generalisation structures should be remodelled as lifetime dependency.

Some of the natural language phrases that indicate lifetime dependency are:

- dependent, support
- aspect
- play, played by, act as, role, actor, ensemble
- realise, materialise, stuff, made of
- symbol, sign, shape, substance, physical, matter, form-function

Some phrases indicate a representation relation rather than lifetime dependency:

- describe, denote, represent, express

To decide whether there is a representation relation, one should ask:

- Can the objects be separated in time and space?
- Can they exist independently of each other?

When users are to evaluate a model, the NIAM method suggests that instead of showing the model to the users, the analysts should translate the model to natural language (Nijssen and Halpin, 1989). A lifetime dependency relation could be expressed like this, giving the proper object names instead of “dependent” and “support”:

“The dependent exists only as long as all the support exist. If one of the support objects has not appeared, the dependent cannot be created, and if one of the support objects ceases to exist, the dependent will also disappear.”

## 6 DISCUSSION AND CONCLUSION

We have defined a new abstraction relation called **lifetime dependency**: when an object (the **dependent**) throughout its lifetime depends on the existence of other objects (the **support**), the dependent stays in a lifetime dependency to the support. Lifetime dependency captures relations between aspects like roles, symbols, and relations with attributes, whose existence depend on other phenomena. The previous approaches solved some of the modelling problems, but they often introduced other problems instead, eg, creating additional types of objects or restricting the possibility to express cardinalities.

However, when introducing a new type of relation, we add complexity to the modelling techniques. On one extreme, one can say that everything can be modelled by means of Turing machines. The practical problem of describing anything relevant is immense, however. On the other extreme, one can create a library consisting of thousands of specific objects and relations. However, there is no indication that it would be an easy task to find the right

relation, even if it had happened to exist in the library. So the maxim for creating modelling techniques is to define the minimal amount of means of expression carrying the maximum of relevant semantics.

Even if agreeing that we have identified a useful modelling relation that is better than the ones we have criticised, a purist may say that there are sufficient modelling mechanisms already, just combine the existing ones. Eg, say that the roles are parts of a person and that woman and man are parts of a marriage, add cardinalities, and do not express existence dependency. However, we will give the purist the three following reasons why she or he should bother with lifetime dependency:

**Real world constitution.** We find three general connections between objects in the problem domains we analyse: the connection between different aspects of the same parts (lifetime dependency), the connection between parts and a whole (separable), and the connection between a description and the extension it denotes (always independent). If we accept one of these relations, we should include the other two also for completeness.

**Reducing the bias of modelling static relations.** Analysis of problem domains have departed from data modelling, which has a rudimentary concept of time. Object-oriented approaches allow for modelling a life cycle of each object. A life cycle may be regarded as a process of states and transitions during the lifetime of the object. In order to include the strength of object-orientation also in modelling, the useful concepts learned through data modelling should be revised according to the expressiveness that object-orientation allows. Since object-orientation adds dynamics compared to data models, concepts that express basic dynamics should be included in the models.

**Frequency of occurrence.** This article has only considered lifetime dependency in a theoretical way. Empirical evaluations will have to be carried out to determine whether or not lifetime dependency occurs so frequently that it should be supported in tools and techniques. Coad (1992) includes the person-role relation in his collection of patterns, and link-objects appear. These are indications that the relation will be found frequently.

## 6.1 Further research

In addition to the empirical studies needed, the lifetime dependency relation also needs to be extended with guidelines for how to model the dynamics between the support and the dependent. Ways to terminate the dependent in a controlled manner when the support disappears have to be indicated. State-transition diagrams like Petri nets (Jensen, 1992), event modelling (Mathiassen et al, 1992), or action models (Jackson, 1983) could be useful.

Guidelines for implementation should also be developed. It would be useful to show how lifetime dependency can be expressed in some programming languages, eg, C++ or Smalltalk. When the specific restrictions that the solutions proposed by others are valid, their guidelines for implementation may be used.

## REFERENCES

- Coad, P. (1992) Object-Oriented Patterns Communications of the ACM 35, 9, 152–159.  
Coad, P. and Yourdon, E. (1991) *Object-Oriented Analysis* Second Edition, Prentice-Hall, NJ  
Flynn, D.J.; Knight, D.R.; Laender, A.H.F. (1995) Multiple Relationships: An analysis of their semantics and their modelling, in E.D. Falkenberg (ed.) *Information System Concepts: Towards a Consolidation of Views* Chapman & Hall, London.

- Goldstein, R.C. and Storey, V.C. (1992) Unravelling Is-a Structures, *Information systems Research*, 3,2,99–126.
- Goldstein, R.C. and Storey, V.C. (1994) Materialization *IEEE Transaction on Knowledge and Data Engineering* 6, 5, 835–842.
- Gottlob, G.; Schrefl, M.; and Röck, B. (1994) Extending Object-Oriented Systems with Roles. To appear in *ACM Transactions on Information Systems*
- Jackson, M. (1983) *System Development* Prentice-Hall, New Jersey.
- Jensen, K. (1992) *Coloured Petri Nets* Springer-Verlag, Berlin.
- Kaasbøll, J.J. (1995) Abstraction and concretizing in information systems and problem domains: Implications for system descriptions, in E.D. Falkenberg (ed.) *Information System Concepts: Towards a Consolidation of Views* Chapman & Hall, London.
- Martin, J. and Odell, J.J. (1992) *Object-Oriented Analysis and Design* Prentice-Hall.
- Mathiassen, L.; Munk-Madsen, A.; Nielsen, P. A.; and Stage, J. (1995) Modelling Events in Object-Oriented Analysis. In D.Patel, Y.Sun, S.Patel (eds.) *1994 International Conference on Object Oriented Information Systems. Proceedings.* Springer, London
- Meyer, B. (1988) *Object-Oriented Software Construction*, Prentice Hall.
- Motschnig-Pitrik, R. and Storey, V.C. (1995) *Modelling of Set Membership: The Notion and the Issues* Accepted for publication.
- Navathe, S. (1992) Evolution of Data Modelling for Databases, *Communications of the ACM* 35, 9, 112–123.
- Nijssen, G.M. and Halpin, T.A. (1989) *Conceptual schema and relational database design : a fact oriented approach*, Prentice-Hall, NY.
- Pernici, B. (1990) Objects with Roles, *SIGOIS Bulletin* 11, 2/3, 205–215.
- Reenskaug, T.; Andersen; Berre; Hurlen; Landmark; Lehne; Nordhagen; Næss-Ulseth; Oftedal; Skaar; Stenslet (1992) OORASS: seamless support for the creation and maintenance of object oriented systems, *Journal of Object-Oriented Programming*, October, 27–41.
- Richardson, J. and Schwarz, P. (1991) Aspects: Extending objects to support multiple, independent roles, *SIGMOD Record*, 20, No.2, 298–307.
- Sciore, E. (1989) Object Specialization, *ACM Transactions on Information Systems* 7,2, 103–122.
- Smith, J.M. and Smith, D.C.P. (1977) Database Abstractions: Aggregation and Generalization *ACM Transactions on Database Systems* 2, 2, 105–133.
- van de Weg, R. L.W. and Engmann, R. (1992) A framework and Method for Object-Oriented Information Systems Analysis and Design. In E.D. Falkenberg, C. Rolland, and E.N. El-Sayed (eds.) *Information System Concepts: Improving the Understanding* ISCO 2, IFIP Transactions A-4, North-Holland, 123–146.
- Velho, A.V. and Carapuça, R. (1994) From Entity-Relationship Models to Role-Attribute Models, in R.A. Elmasri, V. Kouramajian, and B. Thalheim (eds.) *Entity-Relationship Approach—ER '93* Springer-Verlag, Berlin, 257–270

## ACKNOWLEDGEMENT

Thanks to Peter McKenzie, Ragnar Normann, Markku Nurminen, Jan-Erik Ressem, Phillip Steele, Jim Sykes, and the anonymous referees for constructive comments.