CHAPTER 2

WHAT'S IN A RELATIONSHIP? ON DISTINGUISHING PROPERTY HOLDING AND OBJECT BINDING

This paper was published in *Proceedings of the 3rd International Conference on Information Systems, ISCO3: Towards a Consolidation of Views*, Chapman & Hall, 1995, pp. 144–159.

WHAT'S IN A RELATIONSHIP? ON DISTINGUISHING PROPERTY HOLDING AND OBJECT BINDING

Steven Kelly
Department of Computer Science and Information Systems
University of Jyväskylä
P.O. Box 35
FIN–40351 JYVÄSKYLÄ
Finland
email: kelly@cs.jyu.fi
fax: +358 14 603011

This paper proposes improvements to the underlying data- or meta-metamodels used to represent information systems and IS development methods. It concentrates on the modelling of relationships, with particular reference to CASE and methodology engineering. Existing models are examined and some problems are identified arising from the confusion of the connection aspect of relationships with their property-holding aspect. The separation of attributes of relationships from information about what objects they link is proposed as a solution, and a concept of binding is proposed to model the linking information. The improved model is tested theoretically, semantically, and in practice on three metamodels.

Keywords: relationship, conceptual model, metamodel

1 Introduction

This paper addresses issues concerning the concept of relationship used in information systems research, particularly in the models and metamodels used by computer-aided software engineering (CASE) tools. After providing some background, it shows problems arising from the current understanding and use of the concept of relationship, proposes changes to the concept, and examines the benefits of these changes.

The field of information systems has seen a continuing growth in complexity in the modelling of relationships. It started out with the network (COD62) and relational (Cod70) database types of links between objects, based on pointers or shared values. This was extended to the Entity-Relationship model (Che76) by adding a 'relationship' concept between objects. This was extended to the OPRR model (Wel92) by adding a 'role' concept between each relationship and its object (whilst Chen did mention roles in his paper, the concept was not fully developed). Yet still there is no link between objects themselves: we still have nothing that exists between an object and its role, except for what we had already in a relational database model with shared values, or a network database with pointers. To state the matter another way, our attempts to model the concept of relationship seem to have been mostly adding another object-like concept between our objects, without properly addressing the question of how this new object-like concept is connected to the objects. Perhaps because these additional concepts have been only partially successful, there has been considerable debate over the nature of relationships, in particular about relationships and properties: whether relationhip and property are different concepts (ER vs. binary models (Lou92, p.9)); and whether a relationship can have properties or not (ERA vs. EAR (Wel92)). In this paper we present a conceptual level approach which aims at going beyond these differences by analysing the rather vague concept of relationship into constituent parts, each clearly defined and distinct.

A relationship (in the most general sense) consists of two kinds of information:

binding information, saying what it is a relationship between, (e.g. a Data Flow relationship from Process '3.1' to Process '3.2', or an Inherits relationship with Class 'Window' in a Superclass role, Class 'TextWindow' in a Subclass role, and Class 'GraphicsWindow' in another Subclass role)

property information, describing features of the relationship itself (e.g. a Data Flow relationship may have a Name property 'handle order').

In this paper we propose the adoption of the concept of *binding* as maintaining that actual information of 'what is connected to what', and continue the use of object-like relationships and roles to provide extra information about the various points of the connection. The concept of binding is applicable on the semantic, conceptual and implementation levels: here we shall concentrate mostly on the conceptual level, and take a broad look at the implementation level to add clarity by showing the concepts at work in the field of CASE and metaCASE.

First we shall look at the background of the topic — various theories and points of view on relationships, and at metaCASE in general and the MetaEdit+ metaCASE environment in particular. In Section 3 we will examine some of the requirements for relationships in metaCASE, and see some weaknesses in current approaches. Section 4 suggests bindings as an approach to resolving many of the problems with relationships in metaCASE. Finally, we present some conclusions and directions for further research.

2 Background

2.1 Theoretical background of relationships

In this section we will look at some of the theories and models of relationships: the number of models is huge, and only a small fraction can be even briefly mentioned here: a good source of references can be found in (Lou92) and (Rol92).

Research in the human sciences has shown that semantic relations between concepts are basic components of language and thought (Cha84). This research is now being brought to bear in the field of semantic modelling, in an attempt to allow information systems to capture more understanding of the real world (Sto92). Whilst the human sciences are more concerned with identifying different types of relationships, ontology attempts to analyse the nature of the relationship concept itself, as a construct for modelling the real world. Wand, Storey and Weber (Wan93) use Bunge's ontology (Bun77, Bun79) to analyse a relationship as being a mutual property of a set of object types. (In their ontology, the term property is used differently from what is common in information systems: a property is considered to be a basic underlying reality dependent on possibly several things (objects), and not an observed reality. The IS meaning of 'property' — a characteristic assigned to a thing by humans — is called by them an attribute.) Their analysis thus bears a close resemblance to the binary relationship model (Abr74, Smi77) which has given rise to such models as the Object-Role model (Fal76) and NIAM (Nij89).

This ontological analysis of relationship goes some way to separating out what objects the relationship connects from any attributes of the relationship itself. However, the separation is by no means complete, and their property concept becomes overloaded, modelling what in the IS world are called attributes, relationships and complex objects. While keeping the number of basic concepts to the absolute minimum is a good thing in ontology, it may not be so useful when applied to information systems modelling.

Indeed, the history of data models in information systems shows a steady growth in the number of concepts considered basic, especially in the area of relationships. Early data modelling was closely tied to database theory and implementations. The CODASYL database specification (COD62) allowed the connection of objects in the database by a network of pointers: each record object contained its own set of pointers. It also allowed connection using the same principles as the more formally defined relational databases (Cod70), where objects were linked by shared attributes. The linking attributes were handled in the same way as normal non-linking attributes, the main difference being that for a linking attribute to be useful it had to uniquely identify the object it linked to.

In the late 1970s, implementation-based data modelling began to give way to semantic data modelling. One of the early steps in this direction was Chen's Entity Relationship model (Che76), which introduced a separate concept of relationship, and also initial ideas on roles and attributes. In Chen's model, relationships could have attributes, but roles were only a means of typing the way an object participated in a relationship, and thus had no attributes. Many extensions to Chen's ideas have been presented, including various models where relationships may not have attributes, and, in the other direction, models where roles may also have attributes. Hainaut (Hai89) presents a detailed analysis of the features of the variants. These variants have formed the basis of many metaCASE tools (see e.g. (Tei80) for a description of one ER-based system).

2.2 MetaCASE

One main approach to solving the software crisis (Bro75) has been the maturing of software development into an engineering discipline by the adoption of well-motivated methods to be followed when building a system. Such methods involve the collection and maintenance of large amounts of highly interconnected data, a job at which computers excel, hence the development of CASE (Computer Aided System / Software Engineering) tools to support this process. Whilst these tools have had some success, they have typically been inflexible, supporting a single method and allowing no modifications to it. This inflexibility has led to problems as they are taken into use (LeQ90, Smo90), as they are unable to support the contingency-specific modifications which organisations need to make (Wij90).

Whilst commercial CASE tools are allowing ever more customisation, this is often on a fairly superficial level. However, research has been progressing into fully configurable and customisable CASE tools, called variously CASE shells (Bub88), metaCASE tools or metamodelling tools. The ancestors of such tools can be seen as far back as the 1970s with text-based environments such as PSL/PSA (Tei77). Plexsys (Kot84), QuickSpec (Wel88), and Metaplex (Che89). More recently, graphical CASE shells have been developed, such as Eclipse (Bee87), Metaview (Sor88), Ramatic (Ber89), IPSYS Tool Builder (Ald91), and VSF (Poc91).

As the field began to mature, there was a steady move to the definition and formalisation of the concepts and activities involved in metaCASE (e.g. Bri90, Wij92). The discipline of method engineering — a term coined by Kumar and Welke (Kum92) — has appeared, concerned with the development and application of methods for specific contingencies (Slo93). This in its turn has given rise to the concept of CAME (Computer Aided Methodology Engineering) (Hey93a), a topic attracting research interest in both academia and industry (Hid93).

The concepts of method engineering are now being applied to improve the support for method engineering in metaCASE tools. CASE shells where its effects can be seen would include MetaEdit (Smo91a), the use of a separate front-end to a CASE tool by Harmsen and Brinkkemper (Har93), Heym's prototype Method Engineering Tool MEET (Hey93b), Saeki and Wenyin's adoption of Object-Z as a metamodelling language to address problems identified by method engineering (Sae94), and the prototype MetaEdit+ (Kel94a).

In the following section we examine MetaEdit+, and we will use it, and the GOPRR metametamodel it is based on, as a concrete basis to provide examples for our discussion in the remainder of the paper.

2.3 MetaEdit+

MetaEdit+ is a multi-user, multi-tool, metaCASE environment under development in the MetaPHOR project (Lyy94). It is based on the earlier single-user, single-tool MetaEdit, which used the four concepts, or *metatypes*, of the OPRR model (Wel92, Smo91b) (described below) to model design methods, and also as the underlying data model for the design diagrams themselves. The implementation, however, did not exactly follow the formalisation of OPRR, especially in the area of relationships. For MetaEdit+, the OPRR model has been extended to GOPRR (Smo93), for instance by the metatype of Graph, which allows modelling of multiple diagrams and complex objects, and facilitates the use of several methods simultaneously.

It is worth noting that GOPRR is intended to be used to model observed, interpreted and recorded reality (including the world of thought and abstract ideas), in particular ISD methods and models made with them. In this respect it differs from ontological models, which attempted to model what actually *is*, rather than just what is perceived and recorded.

Here then are the GOPRR metatypes:

- **P**roperties, which typically appear as textual labels in diagrams, contain single data entries such as a name, text field or number, and model concepts such as the number of a Process in a Data Flow Diagram.
- **O**bjects, which typically appear as shapes in diagrams, have properties and model concepts such as an Entity in an Entity Relationship Diagram or a Process in a Data Flow Diagram.
- **R**elationships, which typically appear as lines between shapes in diagrams, have properties, and model concepts such as a Data Flow in a Data Flow Diagram.
- **R**oles, which typically appear as the end points of Relationships (e.g. an arrowhead), have properties, and model the part an Object plays in a Relationship, such as which end of a relationship is 'to' and which 'from'.
- **G**raphs, which typically appear as windows on whole diagrams, contain objects, roles, relationships and bindings of these, have properties, and model concepts such as a whole Data Flow Diagram.

This fixed conceptual meta-metamodel forms the basis on which varied representations of data, including not only the usual graphical diagrams but also hypertext, text and matrices, can be built. This allows the application to support a wider range of existing methodologies, and gives users the ability to see and manipulate their information in a variety of representations.

The primary tools available for manipulating models within MetaEdit+, in addition to the graphical DrawWindow of MetaEdit, are the Matrix Editor (Kel94b) and the Table or Text Editor (used for methods whose main representational format is textual or tabular, e.g. many business systems re-engineering methods, which are increasing rapidly in number (Dav90)).

3 Requirements and starting point

We can judge the effectiveness of a set of ideas about relationships on three levels, not dissimilar from the ANSI database model (ANS75):

- Semantic: does the model fit in with our ideas of what a relationship is?
- Conceptual: do the individual concepts cover the semantic field with no gaps, without overlap, and with a number of distinct concepts that represents an appropriate balance between a minimal set and a coherent single purpose for each concept?
- Implementation: is there an implementation that accurately reflects the concepts? Can we map between the concepts and implementation structures without loss? Can the concepts be implemented efficiently compared to other concept sets in terms of storage and speed of operation?

In this section we will see how well these requirements are addressed within the field of metaCASE for the current standard data models, and in the following sections examine how the concept of binding can help resolve some of the problems. First, we must present metaCASE and its needs and assumptions in some more detail.

3.1 MetaCASE: modelling methods, not the real world

In this section we intend to demonstrate the essential arbitrariness of models of methods. To be more exact, we propose that a method modeller should be free to model a method in a way which best fits his contingency, and that there is no one best way to model a method with a given data model or meta-metamodel.

When modelling a method, various means may be used to decide how to analyse the description of the method (often presented in a book including the steps to be followed and examples of what to draw at each step) to produce a set of components, and then to map these components onto the metatypes of the meta-metamodel: the facilities available for modelling methods. Common strategies for analysis and mapping are to use the graphical representation of the symbols and their interrelationships, to use the names of the types from the method, or to model in such a way that the meta-metamodel and metaCASE tool functionality are used to best effect with respect to the method.

Examples

For instance, consider using GOPRR (Smo93) to metamodel the NIAM method (Nij89) compared and contrasted with Entity-Relationship Diagrams (Che76), based on the diagram in Figure 1.

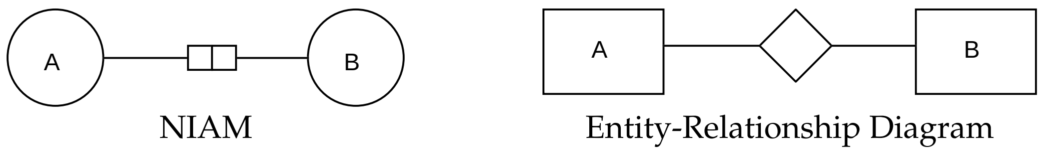


FIGURE 1 NIAM and E-R diagram graphical representations

Deciding on a purely graphical basis, we would most likely conclude that in both methods A and B were objects of one type, that the symbol in the middle of the diagrams was an object of another type, and the lines were relationships with roles (the roles and relationships thus having no symbols).

Deciding based on the names used, we would again find A and B as objects (entity being a common synonym), but the diamond in ER would become the relationship symbol on the line from A to B. In NIAM, we may decide that there is a relationship with no symbol at the centre of the double square, and the symbols for the roles appear (unusually) at the centre point of the relationship. Alternatively, we could posit an object with no symbol being at the centre of the double square, each square being a role, and each line being a relationship, with the roles near the circles having no symbol.

If our metamodelling environment does not allow relationships to have symbols, but rather only lines (as is the case in for instance MetaEdit (Smo91a)), we are forced to eliminate the second solutions for each method. Similarly, if our final modelling environment provides useful method-independent functions for transitive closure etc., based on connections of objects via relationships, we would be more likely to choose a solution where each method had only one object type, so the connections between objects of this type are made using relationships and roles, and thus are found by the transitive closure algorithm.

In all but the simplest of methods, there is often no clear answer to the question of which metamodelling strategy is best. To take an example at hand, consider what happens in NIAM when we objectify the association between A and B, as in Figure 2.

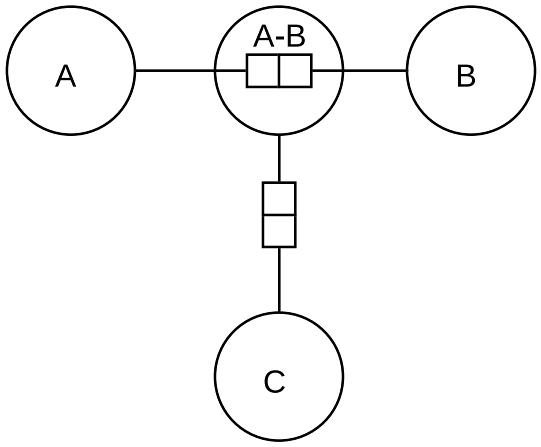


FIGURE 2 NIAM's objectified association

What now is the best way to model the objectified association A-B? Is it an object, a relationship, or even a complex object? Which of the previous ways of metamodelling NIAM must now be abandoned in the light of this new phenomenon? For a given meta-metamodel and environment, we are currently forced to decide that the objectified association is of exactly one metatype, and cannot be considered as being of another metatype at any stage. This goes against the semantics associated with the method, which view it at some times as a relationship, and at others as an object (Hal94).

From this we can see that in deciding how to model a method, we must look at the method, not the real world it tries to represent. Further, there may be several different ways to model a method, depending on our approach, goals, and even personal style.

3.2 Problems with combining binding and property-possession

The underlying data model in the majority of metaCASE tools is ER-based, of the branch of derivations that allow relationships to have attributes. One particular example is OPRR, variants of which are used in QuickSpec and MetaBase (Wel88), Metaview (Sor88), and MetaEdit (Smo91a), which we shall take as our example. In the formal definition of OPRR (Smo91b) relationship binding information was separate from the relationships and roles. However, in the implementation of OPRR in MetaEdit (Smo91a), the binding information was stored within each of the property-holders along the path of the binding: the objects, role and relationship. In this section we propose to show problems arising from this lack of separation.

All the types apart from Property stored property information and relationship binding information. The handling of property information was uniform for these types, but the binding information was necessarily different. Thus an OPRR Object type stored not only its property types, but also a list of all the role types that could be connected to it. Similarly, a Role type stored a list of the object types that could be connected to it, and the single relationship type to which it could be attached. Finally, each Relationship type stored an ordered pair of Role types that could be attached to it.

There are two main problems with this arrangement: the information about relationship bindings is duplicated, with ensuing inefficiency and consistency dangers; and the expressive power is significantly less than that of OPRR as formalised. We will concentrate on the latter problem, which we may restate thus:

The implementation of type-level bindings in MetaEdit is unsatisfactory, as it is not powerful enough to express all the kinds of bindings found in common methods. There are three main sides to this weakness:

1. it is unable to model relationship types that have more than one binding, e.g. Flow in Data Flow Diagrams;
2. it can only represent binary relationships;
3. it divides up the binding information among object, role and relationship types, thus splitting up what should be a conceptual unit, overloading those types with information that does not belong there, and preventing reusability of those types in a different graph type with different bindings.

Weakness 1 follows from the fact that because it includes the binding information with each of the types MetaEdit can only provide for one binding per relationship type, consisting of a set of ‘from’ objects, a ‘from’ role, the relationship, a ‘to’ role, and a set of ‘to’ objects.

The Flow relationship in Data Flow Diagrams requires two bindings because of its asymmetric nature, as shown in Table 1: any of the object types can participate in the 'To' and 'From' roles, but relations involving only Stores and/or Externals are not allowed.

TABLE 1 Bindings for Flow in Data Flow Diagrams

Relationship	Role	Object	Role	Object
Flow	From	Process External Store	To	Process
Flow	From	Process	To	External Store

On the instance level, the second and third of these remain problems: indeed, concurrent access considerations make the third even more problematic, as we must lock far more than we would otherwise need to, e.g. all connected objects when manipulating a single object. This occurs because if we delete an object, then its associated roles and relationships are deleted, and these deleted roles must be removed from the role list in all connected objects. The information stored in each of the connected objects therefore changes, and hence they should be locked before the operation.

Further, the comparatively slow speed of network access means that we do not want to have to transfer complete objects unless we need them. The main operations that access large numbers of objects are queries, which often only follow bindings, not needing the information inside each member of a binding until the objects which form the query result. For example, to prove there is no transitive connection anywhere in a model between two given objects, we would need to transfer all the data of the model if we were using an implementation that combined property-holding and binding in all its concepts, even though we would only ever use the bindings.

As is clear, some of these issues constitute problems in a single-user, single-tool, single-method, binary relationship environment such as MetaEdit, whereas others only become apparent when moving to the multiple graph type, n-ary relationship scenario of MetaEdit+. Many of them follow from the belief that roles, relationships and bindings are entirely dependent on each other, and that no instance of one can exist without full instances of the others. I propose that this is nothing more than an assumption, and determine to show that the sky does not fall if we proceed without it.

4 Bindings as a solution to relationship problems

We have seen how the combination of the property-possessing and binding aspects of relationships into a single concept can lead to problems. We suggest the separation of the aspects so that there is a new basic concept, binding, which can point to the objects, roles and relationship concepts. Thus, all the connection information, and no property information, is in the binding. Similarly, all the property information about the intended relationship is stored in the relationship and the roles which accompany it.

If we then propose a new concept, we must examine possible criticisms of this addition to the model. Clearly, in the general case adding a concept does not decrease the power of the model, but we must ask the following questions:

1. does the increased complexity make implementation and use harder?
2. do we gain any extra modelling power?
3. does the new concept allow undesirable possibilities as a side-effect?

We can note now in passing that 2 and 3 are largely the same question viewed from different perspectives: we may introduce new possibilities, and we must examine these to see if they are useful and give us extra power, or if they are undesirable.

We must also examine how applicable these ideas are to the instance level, i.e. to the models created with the methods. One of the benefits of the OPRR model is its ability to effectively model both the type (method) and the instance (model) level. For the existing concepts in OPRR the application is simple: 'O' on the type level is an object type, e.g. Process in a Data Flow Diagram; 'O' on the instance level is an instance of that type, e.g. Process '2.3' which models 'Order handling'.

For bindings, we will give as general a definition as possible at this point. A type-level binding is a concept which specifies which object types may be linked by which relationship types and which role types. An instance-level binding is a connection of object instances via relationship and role instances. Note that according to these definitions, bindings are not necessarily part of, nor do they contain, the object, relationship or role types or instances. Further, there is no specification of how bindings should be implemented. Working with this definition, we can state our new question thus:

4. Can bindings be used satisfactorily on both type and instance levels?

In the course of this section we will address these questions and build up a clearer picture of relationships in the GOPRR data model, developing it from the basis of the OPRR model.

4.1 Beyond the binary relationship: more roles and objects

One problem with some data models is that they only allow binary relationships, and thus are unsatisfactory when modelling methods such as NIAM (Nij89), which use relationships of higher degrees. We therefore want the ability to say that several role types can be attached to a relationship type. Many methods allow roles of a certain type to appear from one to many times attached to an instance relationship. For instance, some Data Flow Diagrams allow a flow to fork to many receivers (Figure 3). Thus it is useful to be able to specify a numeric constraint for each role type attached to a relationship type.

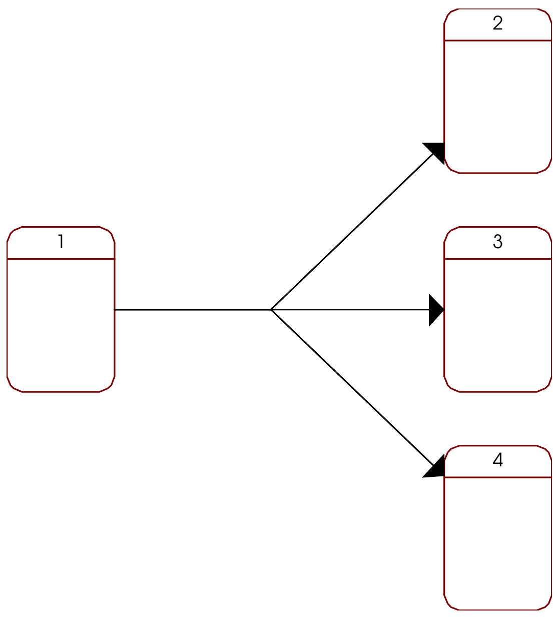


FIGURE 3 Reusing roles of the same type for a given relationship

It should be noted that there are three general ways that the metamodeller can allow a user to include more objects in a relationship:

1a. Add a role type, different from any already present
1b. Add a role type, the same as one already present
2. Allow an existing role type to occur more than once for each relationship
3. Allow more than one object to be attached to a single role

All of these possibilities are theoretically sound, all are useful, and all are present in currently used methods (that is, there are valid metamodels of existing methods containing each of the possibilities). 1a and 1b are of course basically the same, but 1b is explicitly included to show that it is different from 2, in that in 1b the role type may have its own values indicating how many times it may appear in the relationship, whereas in 2 we have simply increased that value for an existing role type.

The distinction between 2 and 3 probably requires some explanation. From a theoretical point of view, we should examine the description of role that we use: 'the part an object plays in a relationship' (Wel92, Smo91b). Hence, if two objects play the same part in a relationship, they should have the same role. From our implementational point of view, the difference is that 2 allows a set of role properties for each object, whereas 3 has several objects sharing role properties. A possible example would be that of a 'family' relationship, with obligatory, single roles 'father' and 'mother' (I prefer the more traditional kind of family!), and optional role 'child', which may occur from 0 to many times. Child has properties 'date of birth', 'place of birth', 'midwife present at birth'. In the general case, each child will get its own role. However, in the case of twins the role should in general be shared, as the information is the same (clearly we can make a still more complicated model that allows for twins born in different places at different times, as can rarely happen. That doesn't mean that this method shouldn't be allowed — remember, we are modelling methods, not the real world).

Thus it is theoretically sound, useful, and necessary (if we want a metamodel sufficiently powerful to model all plausible methods) to extend our concept of role:

A role is the part an object or set of objects play in a relationship.

We have yet to see this in practice on the instance level, but on the type level we have long had the possibility that a given role type (represented by an object of type Role in an OPRR diagram) may be connected to several object types: for instance, in Table 1 the Role type "Flows to" may connect to a Process, Store or External in a Data Flow Diagram (Gan79). It would also be useful in cases where a relationship is binary, and one object is connected to several others. One other possibility would be a slight extension of NIAM (Nij89), where there could be multiple lines coming from a single role in the relationship (Figure 4. Note that it is not desirable to model NIAM relationships as objects, as then all our built-in facilities for navigating among and manipulating objects will not work as the user would expect).

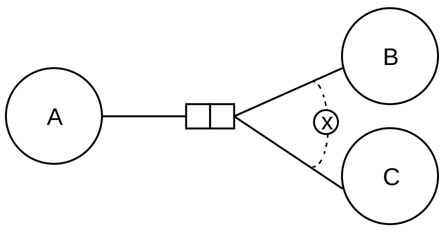


FIGURE 4 NIAM extended with multiple objects per role

4.2 Relationship properties and bindings can exist separately

In Smolander's paper on the definition of OPRR (Smo91b), the bindings are introduced as being separate from the relationship, role, and object types, and role types too are totally distinct from the relationship types. However, in implementing MetaEdit the importance of these distinctions was lost, their necessity not being immediately visible in the majority of methods (Data Flow Diagrams being a notable counterexample), and bindings became thought of as inseparable from the relationships that take part in them. The purely graphical nature of the representations in MetaEdit and other CASE tools also contributes to this misapprehension: what is a relationship but a representation of a binding, and what is a binding if not that which is shown by a relationship? When we look at matrices, however, relationships and roles start to assume an altogether more independent life (Kel94b), and become fully-fledged concepts in their own right, able to exist without necessarily being part of a binding. As this idea is so radical, I will take some time to explain it, lest the reader think that I have lost the entire concept of relationship.

Firstly, let us take an analogy from the concepts of object and graph. It is important to notice that although initially we may feel it makes no sense to have a graph without objects, we have allowed this possibility already, as it is a stage that can be passed through on the path to creating a complete, coherent model. In other words, although it may not be legal within the method in the final analysis, we can allow the situation to exist for a while without terrible consequences.

In graphical CASE tools, to create a relationship the user typically first selects the objects to be related and the type of the relationship. A new relationship of this type is then created between the objects, and the user is asked to fill in the properties of the newly-created relationship and roles. Thus the user is forced to create the binding information first, and allowed to leave the property information until later. However, it is also possible that the user may know first about the properties of the relationship, and only later about exactly which objects it will connect between. This could occur for instance in a matrix representing a Data Flow Diagram where Flows are included with the objects on the axis (Lan76). In this case, the matrix editor could allow the creation of a relationship with its associated properties, but without a binding. The binding should, of course, be added later, but in the mean time the model, although probably semantically inconsistent, will still function as expected.

As a concrete example of the semantic possibility, we can look at the parallel between the relationship metatype and the Entity-Relationship Diagram's 'Relationship' object type in CASE tool implementations of that method: clearly it is possible to create instances of ER 'Relationships' without linking them to objects; indeed, it is impossible to create the ER 'binding' (formed by two objects of type 'Entity', one object of type 'Relationship', and two relationships connecting these three) until the 'Relationship' object has been created (see Figure 5).

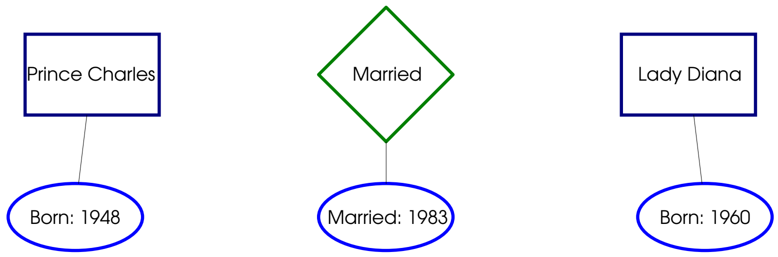


FIGURE 5 Relationship and attributes created in ERA, no binding yet

Whilst the assumption that relationships always finally connect at least two objects is certainly generally true, it is not necessarily true of all relationships in all methods (e.g. triggers in real-time methods (e.g. (War85)) are relationships that have no object on the starting end).

4.3 Roles and relationships can exist separately

We have seen that relationships can exist without being in a binding. Another new possibility that the binding scheme allows is for relationships and roles to exist independently of each other. We thus check in this section to see if this situation could be useful, or merely provides a potential source of errors and consistencies.

A prime example of the usefulness of roles separate from relationships is seen if we examine the case of a decomposition involving n-ary relationships on the interface boundary. In a graphical representation we would see 'orphan' lines leaving from the edge of a diagram: there would be only one line leaving from the decomposition, even in the case where this is only one of two inputs to a relationship in the parent graph (see Figure 6). As an aside, we may note another false assumption inherited from focusing too narrowly on only binary relationships: the assumption tends to be that a line in a graphical method is a relationship, and only the possible symbol part at the end is a role. In fact, the line in general belongs to the role: imagine in Figure 6 that the output line to C was shown thicker: this would not then be a feature of the relationship (else the input lines from A and B would also be thick), but rather of the 'output' role type attached to C.

It *is* possible to make the representations appear as desired, and still maintain the previous assumptions (by e.g. always creating fake temporary objects, roles or relationships). However, it is far simpler to allow the situation in the data model, and then provide rules to check it — so certain methods can allow 'orphan' relationships or roles in cases they can specify, and others can disallow them entirely.

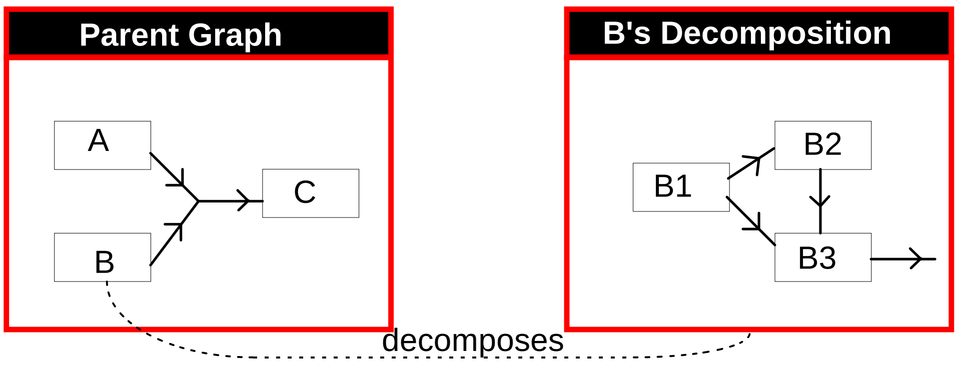


FIGURE 6 Arrow leaving B3 is a role, not a relationship

4.4 Implementation efficiency

If we decide to adopt bindings as separate concepts from roles and relationships, we must also consider the effect on the implementation. Three possible schemes for specifying binding information for a single graph type were considered: binding information stored in the property-holding metatypes along the path of the binding, as in the original MetaEdit OPRR implementation; a separate binding concept consisting of a relationship type and a collection of (object type, role type) pairs; and a separate binding concept consisting of sets of relationship, object and role types instead of just single types. This third scheme was found to be largely similar to the second, but less efficient on all but the very simplest models, and is thus omitted from the discussion for brevity.

First we present the formal grammars of the schemes, in Backus-Naur format: note that property and representation information is omitted for clarity and fairness of comparison.

Binding and property-holding combined
(MetaEdit OPRR extended for n-ary relationships and rules)

```
ConnRuleSet       ::= ObjectType*, RelationshipType*, RoleType*
ObjectType        ::= RoleType*
RoleType          ::= RelationshipType, ObjectType*
RelationshipType  ::= Rule*, RoleType*
```

Separate binding concept containing sets of types

```
ConnRuleSet       ::= ConnRule*
ConnRule          ::= RelationshipInfo, Connection, Connection+
RelationshipInfo  ::= RelationshipType+, Rule*
Connection        ::= ObjectType+, RoleType+
```

Note that the schemes include the concept of Rule for each binding. A Rule is a condition dependent on the objects, roles and relationship (and / or their types) to be connected. It further defines when a particular binding is legal, and all the rules for a given type binding (ConnRule) must be fulfilled before the binding is legal in a given situation. In MetaEdit, some of the more usual rules were implemented by allowing cardinality and connectivity constraints to be stored with Roles: including these constraints in the calculations would have unfairly increased the values for the first scheme, and thus they have been replaced by rules there too.

These schemes are now applied to three real metamodels, representing various fields of ISD methods, and various levels of complexity. As we include MetaEdit OPRR in the comparison, we limit ourselves to methods which can be modelled satisfactorily with that scheme. The methods we take are Data Flow Diagrams (Gan79), Activity Modelling (Gol92), and Express-G (ISO91).

For each scheme and method, we count the number of objects required to describe the method's connections with that scheme (the metamodels are included in Appendix 1). The number of objects provides a good indication of the level of complexity of the implementation, and its efficiency in terms of storage (the efficiency in terms of speed will be considered later).

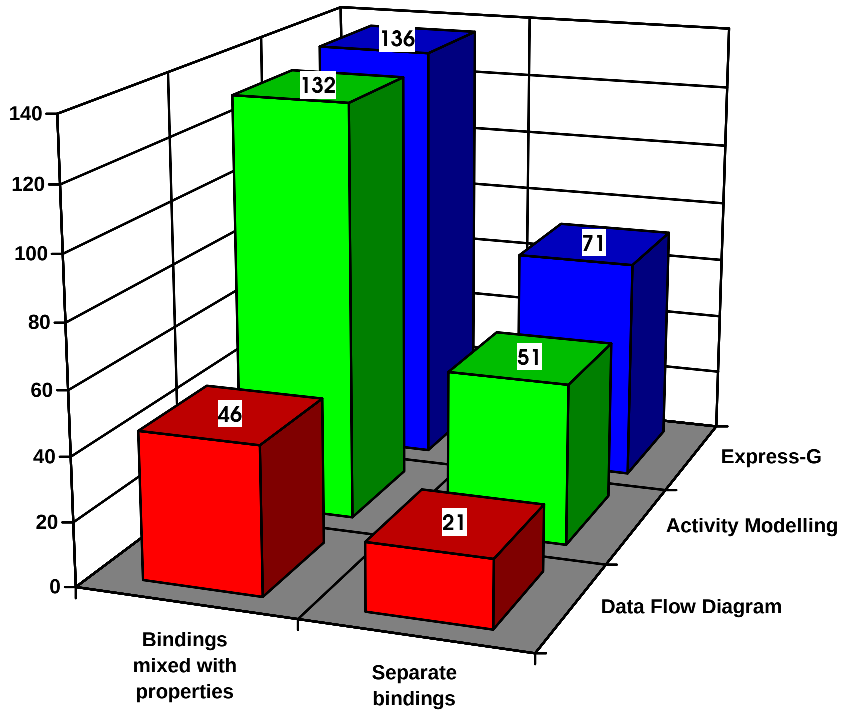


FIGURE 7 Storage efficiency of separating bindings from properties

As the graph in Figure 7 shows, the results are clear: in all cases, bindings are the more efficient scheme, by a factor of around 2 or 3.

The use of sets for all parts of a binding, whilst efficient on the type level, is not always as desirable or efficient on the instance level. Instances of bindings contain only one relationship, and each Connection (a role and its objects: see above) has only one role. Fortunately, the part of the binding where sets were most helpful on the type level was for objects, and it has been shown that multiple objects can usefully participate in the same role on the instance level too. Thus, to maintain the similarity of the model between type and instance levels, it has been decided to adopt bindings where relationship and role parts are filled by single instances or types, and there can be a set of objects or object types for each role. This modified scheme is still more efficient than mixed bindings and property information on the type level.

4.5 Graphs and bindings

Recall that we are considering a multi-method environment: in other words, there can be more than one graph type present in a metamodel for a method. Further, this is a CAME environment, and must thus support the method modeller, for example by allowing him to reuse previously created method components (Kum92) or 'fragments' (Har93). This reuse requirement motivated already against the combination of the binding and property-holding aspects of a relationship type — the same property-holding type may be used with different bindings in a different graph type, for example because there are different object types there.

Similarly, on the type level, bindings must clearly be considered as contained within a graph type: the same relationship, role and object types may have different bindings in different graph types, e.g. a modified version of Data Flow Diagrams could allow a Flow directly between Stores, and this graph type could be included in a metamodel with a normal Data Flow Diagram graph type, for instance as an overview of how information flows through stores.

On the instance level, aside from trying to maintain a common approach with the type level, a user may want to query whether A and B are connected anywhere in a specific set of graphs: a global set of bindings would be unable to answer this, as it could not tell whether the connection was in the specific set of graphs, or somewhere else in the model. Further, almost all navigation along bindings will take place within a graph (inter-graph relationships, as opposed to explosions, are uncommon), and it is then more efficient to work on that smaller set. Similarly, having to transfer all the bindings for a project over a network when a graph is opened will be slow. When a user wants to make a query at the model level, the query can simply access the bindings of all the constituent graphs.

We are now in a position to define a reasonable subset of GOPRR with bindings. First let us considerably shorten the definitions by defining in advance that a Property-Holder is a Graph, Object, Role or Relationship. Most of these definitions apply to both the type and instance levels: those that only appear on the type level are underlined.

Property-Holder	::= Rule* Property*
Property	::= Rule* Value
Graph	::= Object* Role* Relationship* Binding*
Binding	::= Rule* Relationship Connection+
Connection	::= Role Object*

4.6 Bindings allow polymorphism of metatypes

We have seen that the actual ‘relating’ is not just of objects, but of objects, roles and relationships (two objects connected by different roles and relationships are related in different ways; following a given role, relationship and role from an object can lead to possibly several objects). The objects, roles, and relationships are not themselves aware of which bindings they are in; this is information that only the parent graph knows. This does *not* make roles and relationships into ‘objects’: they are indeed all similar in that they have properties, can explode etc., but very different in the matter of bindings: in a typical binary binding (see e.g. Table 1), the first member is a relationship, the second and fourth roles, and the third and fifth objects. Making the distinction between object, role and relationship only at this point gives us the possibility of later allowing a set of properties for something which is an object in one graph to be viewed as a relationship in another graph (of a different type), e.g. a ‘Relationship’ object in an Entity-Relationship Diagram could also be a 'Data Flow' relationship in a Data Flow Diagram (Kel94a).

Similarly, the perceived association of position in a binding with a metatype gives us the ability to have types or instances that can be viewed as belonging to more than one metatype: we can put a concept whose metatype is not Object into the position in a binding normally filled by an object. This gives us one possible solution to the problem of modelling NIAM's objectified associations: A-B is a relationship (from its metatype), but can be placed in the third position of the binding with C, allowing it to also be considered as an object. The metamodel for this can be consistently defined by creating a new Object type, 'objectified association', which can play the same part in the normal binding as object, and by adding a second binding, similar to that in the original metamodel, but whose relationship position is filled by the objectified association type.

The binding set for the type level would thus be as below in Table 2 (note that for simplicity we stay with binary NIAM, and do not specify any rules).

TABLE 2 Type level bindings for NIAM's objectified association

Relationship	Role	Object	Role	Object
Association	Role	Object Objectified association	Role	Object Objectified association
Objectified association	Role	Object	Role	Object

Thus an objectified association can take the same part as a normal object in an association (the first binding in the table, e.g. A-B behaving like an object in its binding with C in Figure 2), and can also be an association in its own right, but only involving prima facie objects (the second binding, e.g. A-B behaving like an association in the binding between A and B in Figure Figure 2).

This extension requires some slight changes to the above formalism:

Binding ::= Rule* RelationshipPart Connection+
Connection ::= RolePart ObjectPart*

where RelationshipPart, RolePart and ObjectPart are all the same as a general BindingPart:

BindingPart ::= Object | Role | Relationship

The definition of BindingPart can be extended to enable us to deal with bindings that exist on the interface between e.g. an Object's decomposition Graph and the parent Graph in which the Object itself is. These are dealt with by having two partial bindings, one in each graph. These interface bindings give us the ability to reuse model components in a similar way to that of circuit or chip design, with the components maintaining interface data, but no knowledge of the particular situations they are used in. More background and details for these inter-graph links are given in (Kel94a).

5 Conclusions

This paper presents improvements to the concept and modelling of relationships in (meta-)CASE. The main conceptual innovation is the separation of the binding or connecting aspect of a relationship from the property-holding aspect, whilst still maintaining the necessary ability to describe a relationship and its parts with properties. Other improvements include the extension of the possibilities to include objects in relationships by allowing several objects to participate in the same role; the definition of a concept of graph to model complex objects, fully supporting methods having a hierarchy where there is an interface of relationships crossing the graph boundaries between levels of the hierarchy; and the association of perceived semantics with positions in bindings to allow polymorphism of meta-types.

The addition of the concept of binding has been shown to increase the modelling power of the data model, without introducing undesirable possibilities. As the model is based on OPRR, it too is able to model the many methods for which OPRR has been demonstrated sufficient, e.g. the 50+ methods currently supported by MetaEdit. The improved model has been shown capable of representing three methods of varying natures, and of significantly reducing the complexity of the metamodels of these methods compared with OPRR. It has also been shown to increase the efficiency of the stored metamodel data, and to be likely to provide benefits in terms of speed increases when working within a shared multi-user environment.

The model has been shown to be effective for modelling both type level information (methods) and instance level information (models of information systems). Further, it is particularly suitable for method engineering, as it supports reuse and connections between methods and instances of different methods, impossible with OPRR.

The model is currently implemented in a prototype metaCASE environment, MetaEdit+, with promising early results. The intention is to further demonstrate the practical application of the model in conjunction with its implementation, testing and development in MetaEdit+.

One area in particular requiring further work is the concept of rule, mentioned briefly at various points. Currently there is no formalisation of it: in MetaEdit+ it can be any piece of Smalltalk code that produces a Boolean value. Further work needs to be done on the rule constructs provided for the metamodeller to model constraints in metamodels, particularly in the area of cardinality constraints (Lid93).

References

ISO91 ISO, *"EXPRESS Language Reference Manual,"* ISO TC184/SC4/WG5, Document N14, Owner: Philip Spiby, CADDETC, 171 Woodhouse Lane, Leeds LS2 3AR, UK (1991).

Abr74 Abrial, J. R., "Data Semantics Database Management," in *Proceedings of the IFIP Working Conference on Data Base Management*, J. W. Klimbie and K. L. Koffeman (Ed.), North-Holland, Amsterdam (1974).

Ald91 Alderson, Albert, "Meta-CASE Technology," in *Software Development Environments and CASE Technology*, A. Endres and H. Weber (Ed.), Springer-Verlag, Berlin (1991).

ANS75 ANSI, *"Study Group on Data Base Management Systems: Interim Report 75-02-08,"* ACM SIGMOD Newsletter 7(**2**) (1975).

Bee87 Beer, S., R. Welland and I. Sommerville, "Software Design Automation in an IPSE," in *ESEC '87: Proceedings of the 1st European Software Engineering Conference, Strasbourg, France, Sep 9–11, 1987*, H. Nichols and D. Simpson (Ed.), Springer-Verlag, Berlin (1987).

Ber89 Bergsten, Per, Janis Bubenko jr., Roland Dahl, Mats Gustafsson and Lars-Åke Johansson, *"RAMATIC — A CASE Shell for Implementation of Specific CASE Tools,"* SISU, Gothenburg (1989).

Bri90 Brinkkemper, Sjaak, *"Formalisation of Information Systems Modelling,"* Thesis Publishers, Amsterdam (1990).

Bro75 Brooks, F., *"The Mythical Man Month: Essays on Software Engineering,"* Addison-Wesley, Reading, Mass, USA (1975).

Bub88 Bubenko, J. A., *"Selecting a Strategy for Computer-Aided Software Engineering (CASE),"* Technical Report, SYSLAB, University of Stockholm, Sweden (1988).

Bun77 Bunge, M., *"Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World,"* Reidel, Boston (1977).

Bun79 Bunge, M., *"Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems,"* Reidel, Boston (1979).

Cha84 Chaffin, Roger, Douglas J. Herrmann, *"The Similarity and Diversity of Semantic Relations,"* Memory and Cognition 12(**2**) (1984) pp.134–141.

Che76 Chen, P. P., *"The Entity-Relationship Model: Toward a Unified View of Data,"* ACM Transactions on Database Systems 1(**1**) (1976) pp.9–36.

Che89 Chen, Minder, Jr. Jay F. Nunamaker, "METAPLEX: An integrated environment for organization and information systems development," in *Proceedings of the Tenth International Conference on Information Systems, December 4–6, 1989, Boston, Massachusetts*, J. I. DeGross, J. C. Henderson, and B. R. Konsynski (Ed.), ACM Press (1989).

COD62 CODASYL, *"CODASYL Development Committee: An Information Algebra,"* Communications of the ACM 5(**4**) (1962).

Cod70 Codd, E. F., *"A Relational Model of Data for Large Shared Data Banks,"* Communications of the ACM 13(**6**) (1970) pp.377–387.

Dav90	Davenport, T. H., J. E. Short, *"The New Industrial Engineering: Information Technology and Business Process Redesign,"* Sloan Management Review (1990) pp.11–26.
Fal76	Falkenberg, E., "Concepts for modelling information," in *Modelling in Database Management Systems*, G. M. Nijssen (Ed.), North-Holland, Amsterdam (1976).
Gan79	Gane, C., T. Sarson, *"Structured Systems Analysis: Tools and Techniques,"* Prentice Hall, Englewood Cliffs, NJ (1979).
Gol92	Goldkuhl, G., "Contextual activity modeling of information systems," in *Proceeding of the Third International Working Conference on Dynamic Modelling of Information Systems*, Delft Technical university, Noordwijkerhout (1992).
Hai89	Hainaut, J.-L., "A Generic Entity-Relationship Model," in *Information Systems Concepts: An In-depth Analysis*, E. D. Falkenberg and P. Lindgreen (Ed.), Elsevier Science Publishers (North-Holland), Amsterdam (1989).
Har93	Harmsen, F., S. Brinkkemper, "Computer Aided Method Engineering based on existing Meta-CASE technology," in *Proceedings of the Fourth Workshop on The Next Generation of CASE Tools*, Sjaak Brinkkemper, Frank Harmsen (Ed.), Univ. of Twente, Enschede, the Netherlands (1993).
Hey93a	Heym, M., H. Österle, *"Computer-aided methodology engineering,"* Information and Software Technology 35(**6/7**) (1993) pp.345–354.
Hey93b	Heym, M., *"Methoden-EngineeringSpezifikation und Integration von Entwicklungsmethoden für Informationssysteme,"* PhD Thesis, Hochschule St.Gallen, Switzerland (1993).
Hid93	Hidding, Gezinus J., Johan K. Joseph and Gwendolyn M. Freund, "Method Engineering at Andersen Consulting: Task Packages, Job Aids and Work Objects," in *2nd International Summerschool on Method Engineering and Meta Modelling conference binder*, Univ. of Twente, Enschede, the Netherlands (1993).
Kel94a	Kelly, Steven, Veli-Pekka Tahvanainen, "Support for Incremental Method Engineering and MetaCASE," in *Proceedings of the 5th Workshop on the Next Generation of CASE Tools*, B. Theodoulidis (Ed.), Universiteit Twente, Enschede, the Netherlands (1994).
Kel94b	Kelly, Steven, *"A Matrix Editor for a MetaCASE Environment,"* Information and Software Technology 36(**6**) (1994) pp.361–371.
Kot84	Kottemann, J. E., B. R. Konsynski, "Dynamic Metasystems for Information Systems Development," in *Proceedings of the Fifth International Conference on Information Systems* (1984).
Kum92	Kumar, Kuldeep, Richard J. Welke, "Methodology Engineering: A Proposal for Situation Specific Methodology Construction," in *Challenges and Strategies for Research in Systems Development*, Kottermann W. W. and Senn J. A. (Ed.), John Wiley & Sons, Washington (1992).
Lan76	Langefors, B., B. Sundgren, *"Information Systems Architecture,"* Petrocelli / Charter, New York (1976).

LeQ90 LeQuesne, P. N., "*Individual and Organisational Factors in the Design of Integrated Project Support,*" Ph.D. Thesis, London Business School (1990).

Lid93 Liddle, Stephen W., David W. Embley and Scott N. Woodfield, "*Cardinality Constraints in Semantic Data Models,*" Data & Knowledge Engineering (1993) pp.235–270.

Lou92 Loucopoulos, P., "Conceptual Modeling," in *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*, P. Loucopoulos and R. Zicari (Ed.), Wiley, New York (1992).

Lyy94 Lyytinen, K., P. Kerola, J. Kaipala, S. Kelly, J. Lehto, H. Liu, P. Marttiin, H. Oinas-Kukkonen, J. Pirhonen, M. Rossi, K. Smolander, V.-P. Tahvanainen and J.-P. Tolvanen, "*MetaPHOR: Metamodelling, Principles, Hypertext, Objects and Repositories,*" Technical Report TR-7, Department of Computer Science and Information Systems, University of Jyväskylä, Finland (1994).

Nij89 Nijssen, G. M., T. A. Halpin, "*Conceptual Schema and Relational Database Design: A fact oriented approach,*" Prentice-Hall, Englewood Cliffs, NJ (1989).

Poc91 Pocock, John N., "VSF and its Relationship to Open Systems and Standard Repositories," in *Software Development Environments and CASE Technology*, A. Endres and H. Weber (Ed.), Springer-Verlag, Berlin (1991).

Rol92 Rolland, C., C. Cauvet, "Trends and Perspectives in Conceptual Modeling," in *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*, P. Loucopoulos and R. Zicari (Ed.), Wiley, New York (1992).

Sae94 Saeki, Motoshi, Kuo Wenyin, "Specifying Software Specification & Design Methods," in *CAiSE '94 Proceedings*, Gerard Wijers, Sjaak Brinkkemper and Tony Wasserman (Ed.), Springer-Verlag, Berlin (1994).

Slo93 Slooten, Kees van, Sjaak Brinkkemper, "A Method Engineering Approach to Information Systems Development," in *Procs. of the IFIP WG 8.1 Working Conference on the Information Systems Development Process*, N. Prakash, C. Rolland and B. Pernici (Ed.), North-Holland, Amsterdam (1993).

Smi77 Smith, J. M., D. C. P. Smith, "*Database Abstractions: Aggregation and Generalization,*" ACM Transactions on Database Systems 2(**2**) (1977) pp.105–133.

Smo90 Smolander, Kari, Veli-Pekka Tahvanainen and Kalle Lyytinen, "How to Combine Tools and Methods in Practice: a field study," in *Advanced Information Systems Engineering, proceedings of the Second Nordic*, B. Steinholz, A. Sölvberg and L. Bergman (Ed.), Springer-Verlag, Berlin (1990).

Smo91a Smolander, Kari, Kalle Lyytinen, Veli-Pekka Tahvanainen and Pentti Marttiin, “MetaEdit — A Flexible Graphical Environment for Methodology Modelling,” in *Advanced Information Systems Engineering, Proceedings of the Third International Conference CAiSE'91, Trondheim, Norway, May 1991*, R. Andersen, J. A. Bubenko jr. and A. Solvberg (Ed.), Springer-Verlag, Berlin (1991).

Smo91b Smolander, Kari, “OPRR: A Model for Modelling Systems Development Methods,” in *Next Generation CASE Tools*, K. Lyytinen and V.-P. Tahvanainen (Ed.), IOS Press, Amsterdam, the Netherlands (1991).

Smo93 Smolander, Kari, “*GOPRR: a proposal for a meta level model,*” University of Jyväskylä, Finland (1993).

Sor88 Sorenson, Paul G., Jean-Paul Tremblay and Andrew J. McAllister, “*The Metaview System for Many Specification Environments,*” IEEE SOFTWARE (March 1988) pp.30–38.

Sto92 Storey, Veda C., “*Understanding Semantic Relationships,*” VLDB Journal (1992) pp.455–488.

Tei77 Teichroew, Daniel, Ernest A. Hershey III, “*PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems,*” IEEE Transactions on Software Engineering (January 1977).

Tei80 Teichroew, Daniel, Petar Macasovic, III Ernest A. Hershey and Yuzo Yamamoto, “Application of the entity-relationship approach to information processing systems modeling,” in *Entity-Relationship Approach to Systems Analysis and Design*, P. P. Chen (Ed.), North-Holland (1980).

Wan93 Wand, Yair, Veda C. Storey and Ron Weber, “*Analyzing the Meaning of a Relationship,*” Working Paper 92-MIS-011, University of British Columbia (1993).

War85 Ward, P., S. Mellor, “*Structured Analysis for Real-Time Systems,*” Prentice-Hall, New Jersey (1985).

Wel88 Welke, R. J., “Metabase: A Platform for the Next Generation of Meta Systems Products,” in *Proceedings of the Ninth Annual Conference on Applications of Computer-Aided Software Engineering Tools, May 23–27, 1988*, Meta Systems Ltd., Ann Arbor, MI (1988).

Wel92 Welke, R. J., “The CASE Repository: More than another database application,” in *Challenges and Strategies for Research in Systems Development*, William W. Cotterman and James A. Senn (Ed.), Wiley, Chichester UK (1992).

Wij90 Wijers, G. M., H. E. van Dort, “*Experiences with the use of CASE-tools in the Netherlands,*” Advanced Information Systems Engineering (1990) pp.5–20.

Wij92 Wijers, G. M., A. H. M. ter Hofstede and N. E. van Oosterom, “Representation of Information Modelling Knowledge,” in *Next Generation CASE Tools*, K. Lyytinen and V.-P. Tahvanainen (Ed.), IOS Press, Amsterdam, The Netherlands (1992).

Hal94 Halpin, T., personal communication, 1994.

Appendix 1

In this appendix we show the details of the calculations of the efficiency of two different schemes for representing binding information in metamodels, Scheme 1 (as used in MetaEdit, but extended to be able to model n-ary relationships, and with the specific cardinality and connectivity constraints replaced by more general rules), and Scheme 2 (the scheme put forward in the body of this paper).

- Scheme 1: store binding information with objects, roles and relationships
- Scheme 2: store bindings separately, with sets of objects, roles or relationships at each point

Both these schemes are tested over 3 methods:

- Data Flow Diagrams
- Activity Modelling
- Express-G

The formula for calculating the number of objects used for the metamodel (technically, object pointer consumption) is given for each method.

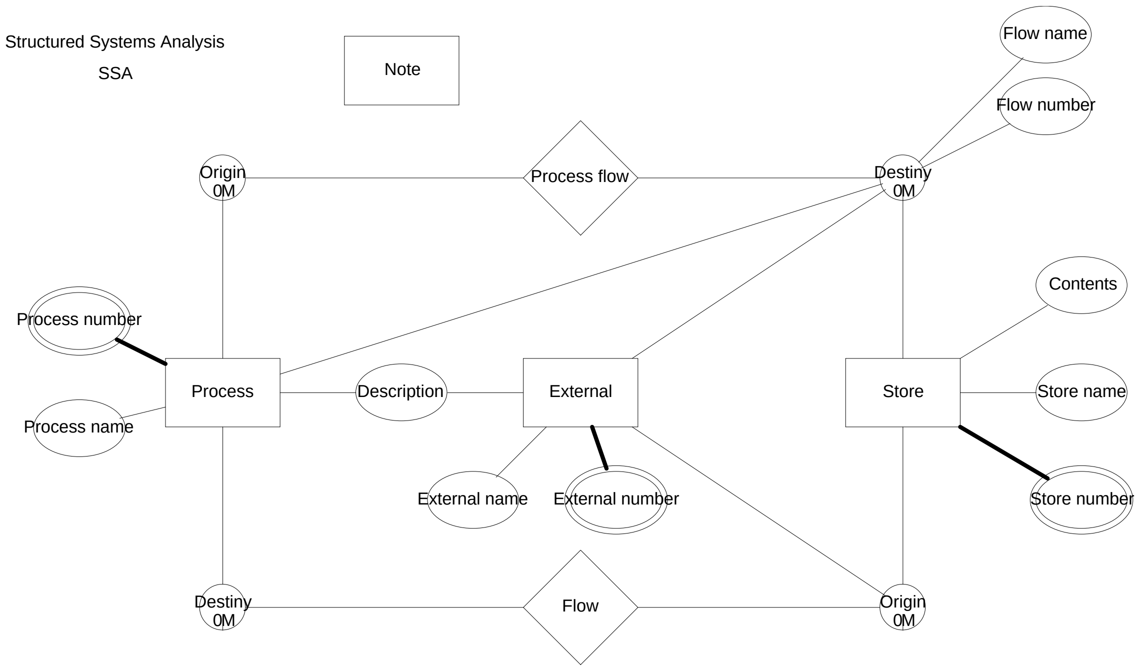
Scheme 1: Binding information stored with objects, roles and relationships

The metamodels are represented here by their OPRR diagrams. In the first diagram the properties are left visible to show how they interact with the binding information. In the other diagrams the properties are hidden as they make the graph difficult to read and do not directly concern the calculations.

The syntax of OPRR is as follows: An oval represents a property type, which is used by the object, role or relationship types it is connected to. A rectangle represents an object type, a circle a role type, and a diamond a relationship type. The legal connections of these types (what we would call legal bindings) can be found by following lines from an object, to a role, through and relationship, to a different role, and finally to an object. For instance, starting from the Process object on the left in the first example, we could go up to Origin, across to Process Flow, across to Destiny, then back down and left to Process, to show that two Processes can be connected in those roles by a Process Flow relationship.

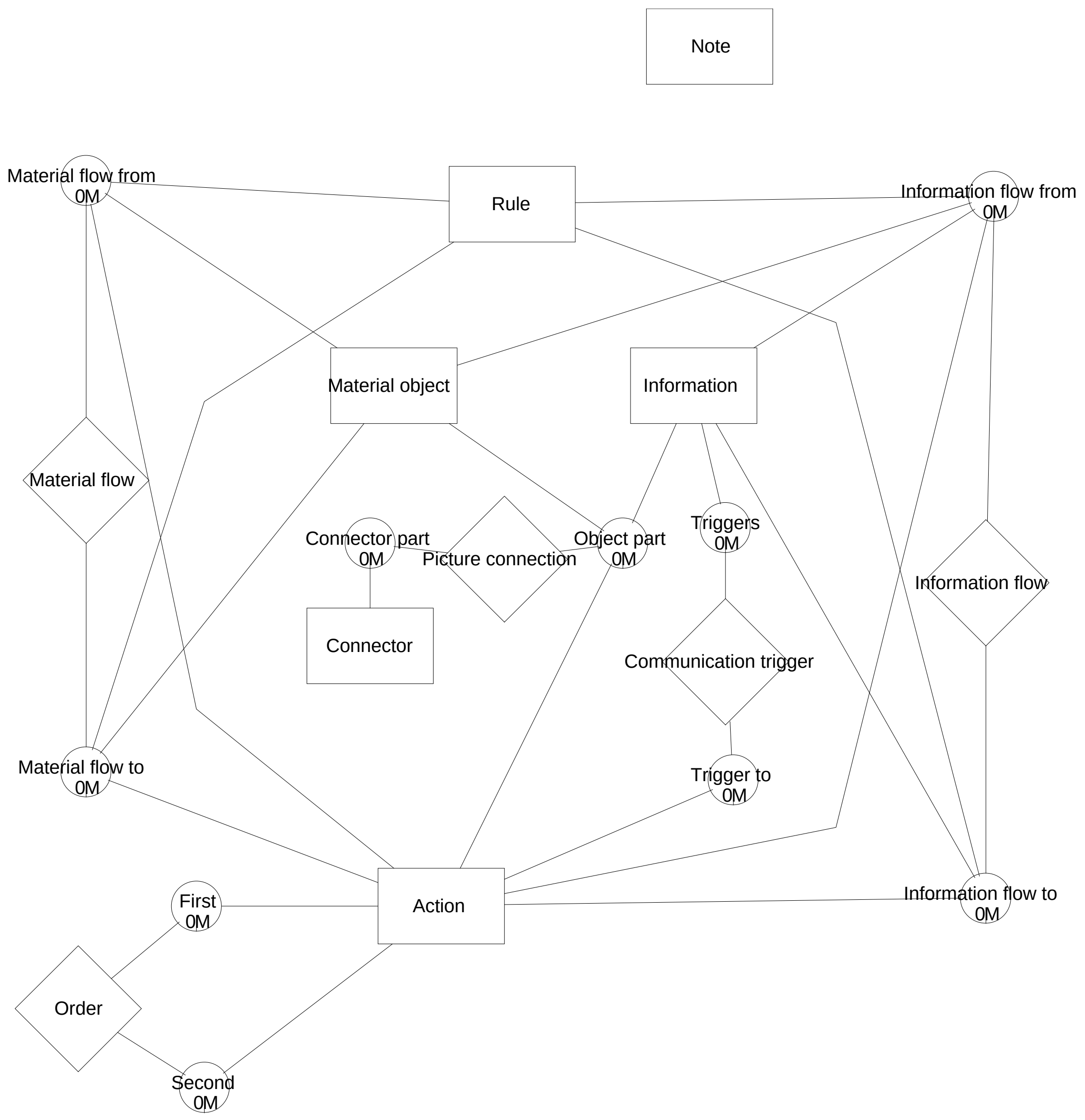
Object pointer consumption: 4+number of objects*2 + number of relationships*6 + number of roles*4 + number of connections between roles and objects*2

Data Flow Diagrams



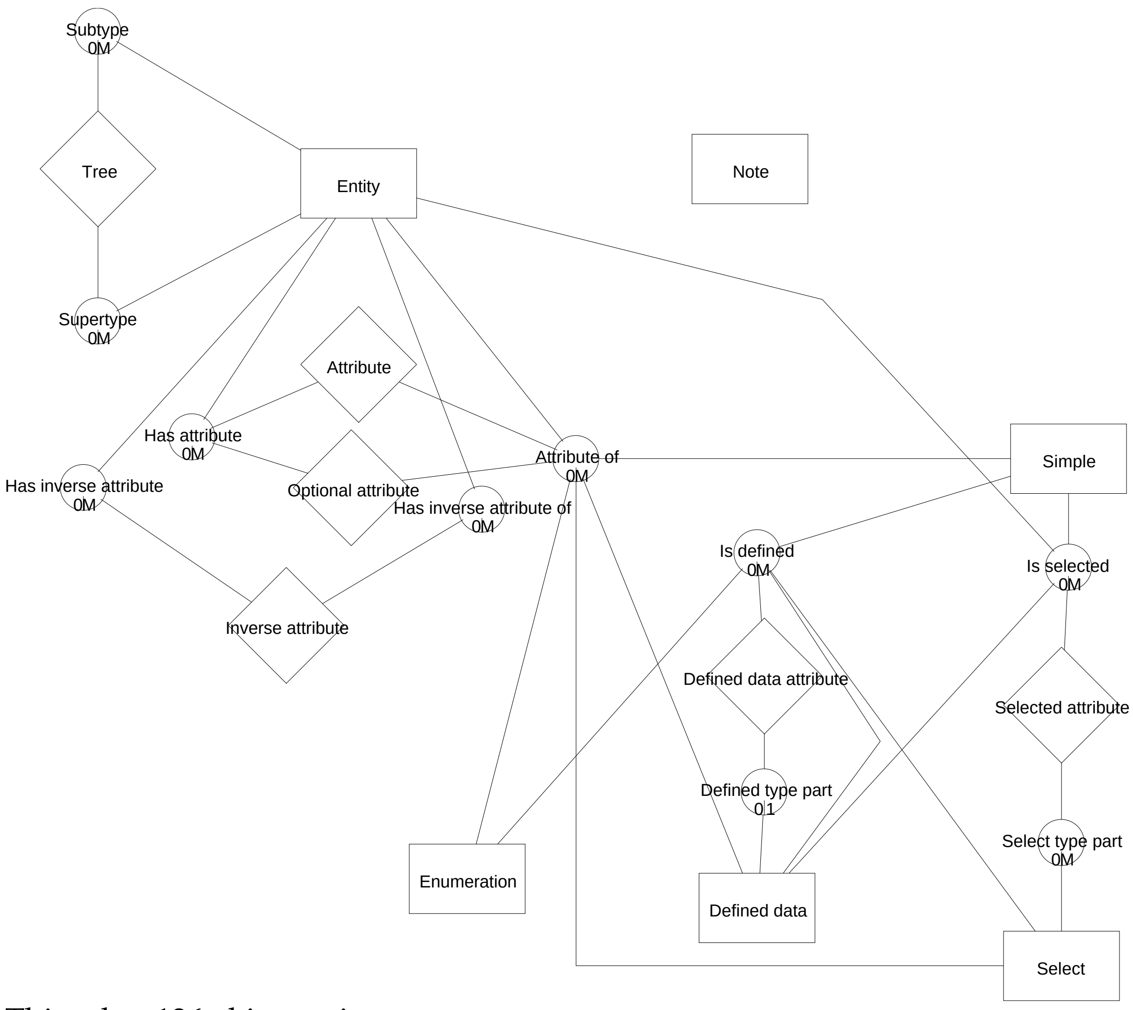
This takes 46 object pointers.

Activity Modelling



This takes 132 object pointers.

Express-G



This takes 136 object pointers.

Scheme 2: bindings with sets of types

Metamodels are presented as sets: {({rels}, {rules}, {({objs}, {roles})})}

Object pointer consumption: 1 + 10*no.of binary relationship rules

Data Flow Diagrams

```
{
 {Flow}, {}, {
            {External Process Store}, {from}
            {Process}, {to}
           }

 {Flow}, {}, {
            {Process}, {from}
            {External Store}, {to}
           }
}
```

this takes 21 object pointers

Activity model

```
{
 {MaterialFlow}, {}, {
                  {Action MaterialObject Rule}, {MaterialFlowFrom}
                  {Action MaterialObject Rule}, {MaterialFlowFo}
                 }
 {InformationFlow}, {}, {
                    {Action MaterialObject Rule Information}, {from}
                    {Action Rule Information}, {to}
                   }
 {PictureConnection}, {}, {
                     {Connector}, {ConnectorPart}
                     {Action MaterialObject Information}, {ObjectPart}
                    }
 {CommunicationTrigger}, {}, {
                        {Information}, {Triggers}
                        {Action}, {TriggerTo}
                        }
 {Order}, {}, {
             {Action}, {first}
             {Action}, {second}
            }
}
```

this takes 51 object pointers

Express-G

```
{
 {Tree}, {}, {
          {Entity}, {SubType}
          {Entity}, {SuperType}
          }
 {Use Reference}, {}, {
          {Schema}, {to}
          {Schema}, {from}
          }
 {InverseAttribute OptionalAttribute Attribute}, {}, {
          {Entity}, {HasAttribute}
          {Entity Simple DefinedData Select Enumeration}, {AttributeOf}
          }
 {DefinedDataAttribute}, {}, {
          {Simple DefinedData Select Enumeration}, {IsDefined}
          {DefinedData}, {DefinedTypePart}
          }
 {SelectedAttribute}, {}, {
          {Simple DefinedData Select Enumeration}, {IsSelected}
          {Select}, {SelectTypePart}
          }
 {InterSchemaReference}, {}, {
          {Entity}, {EntityPart}
          {InterSchemaUsed InterSchemaReferenced}, {InterSchemaPart}
          }
 {ReferenceOntoThisPage ReferenceOntoAnotherPage}, {}, {
          {Entity DefinedData Select}, {ObjectPart}
          {PageReference}, {ReferencePart}
          }
}
```

this takes 71 object pointers

Postscript

The description of GOPRR provided here has proved remarkably resilient. The only significant changes have been in the area of rules, which are covered only on a high level in this article. Rules for properties are described in 'MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment'; here we briefly describe the rules for bindings.

There are two major additions to the rules for bindings given here. Both are applicable only on the metamodel level: thus it appears that, in object-oriented terms, bindings for metamodels are a specialisation of bindings for the model level.

The first addition is the possibility to specify a multiplicity range for a Connection (i.e. a Role and set of Objects). The multiplicity range specifies how many times that Connection may be duplicated in creating instances for this binding. The default is 1..1, in other words that Connection must occur once and only once within each binding. A lower limit of 0 models the case where the Connection is optional, for instance the Refinement relationship from UML. The upper limit can be any number greater than or equal to the lower limit, and also may be set to N (infinity), allowing the Connection to be duplicated as many times as needed. This is useful for modelling many simple n-ary relationships, where there are two role types involved but one may occur multiple times, e.g. in an Inheritance relationship there is often one superclass but any number (>=1) of subclasses.

The second addition attempts to allow modelling of methods where each object of a certain type may only take part in a limited number of roles or relationships of certain type(s). This is generally referred to as a connectivity constraint. There were three somewhat conflicting requirements for the solution: 1) it must be easy to use in metamodelling, 2) it must have reasonable performance for run-time checking, and 3) it must enable modelling of as many such constraints as possible. There is not room here to give a full explanation of these problems, but a brief overview is in order.

1) is important when comparing GOPRR with other meta-metamodels: these invariably are forced to resort to programming or logic languages to specify such constraints, and indeed many use such languages even for the basic kinds of binding described in the paper. We believe that such languages introduce too much complexity: the observed range of possibilities in methods is small enough that it should be possible to specify them without resorting to languages with such great expressive power.

The importance of 2) becomes apparent when we consider how metamodel bindings are used: when a relationship is being created, the user often selects just the objects he wishes to collect, and allows the program to decide what are the legal relationships and roles between them. The set of objects must then be compared with the sets of objects from each binding in the metamodel. For such a comparison to be possible, it is necessary to make sure the order of objects is the same in both sets (because objects A B C may correspond to a binding B A C). If 7 objects are chosen, there are 7! = 5040 ways of ordering them. If there are 10 bindings in a metamodel, there will be over 50,000 comparisons to be performed (before the pruning of the possible 'game tree' by the algorithm). If bindings use the Connection duplication possibilities, there will probably be even more (we must try all possible duplications that give 7 Connections: if we have multiplicity ranges of 0..N, 1..N and 0..N that amounts to 28 comparisons for this binding. If the same were true for each binding, we would have about 1,500,000 comparisons!). This then is the situation before this constraint: for each of these comparisons we must also check that the connectivity constraint holds for each object involved.

3) means we should look at all existing methods, and see what kinds of constraints there are that are similar to this. The most common constraint is that an object may take part in a certain role or relationship only once, e.g. each Class may often only take part in one ‘Subclass’ role (ruling out multiple inheritance). There are however also cases where objects may only take part once in *either* of a pair of roles (or relationships), or where an object may be related only once to another object of a certain kind, with or without specification of the types of relationship and role in between. Most often, methods do not specify exactly what they mean, in particular whether the constraint on an object is related to a role, relationship or object type.

A generic solution would be to specify a constraint for an object type as a range and a partial binding (i.e. one where only some parts are filled in, e.g. no relationship is specified). Checking the constraint would involve checking the partial binding against the existing bindings involving the object, plus the new binding whose legality is to be checked. For the new binding to be legal, the number of matching bindings must be within the range specified. This solution would however probably not satisfy conditions 1) or 2), being both complicated to specify and requiring significant time to check.

Our current solution is to allow graph types to specify for each object type a set of role and relationship types with a maximum legal number of occurrences for each. The absence of an object, relationship or role type means there is no constraint: that combination may occur any number of times. This constraint is significantly easier to specify for the metamodeller, and also to check at run-time. By first calculating the bindings that are legal by the other constraints, we can reduce the number of times these constraints need to be checked to manageable proportions. Further, the number of occurrences of each object in each role and relationship type of interest can be calculated once for the graph before the addition, and then this information can be temporarily augmented with each candidate binding in turn. Whilst this solution is not as powerful at representing all possible types of constraints as the partial binding solution mentioned above, it covers the vast majority of constraints actually found in existing methods, and is much faster and easier to use.