Chapter 5:

Support for Incremental Method Engineering and MetaCASE

This paper is published in *Proceedings of the 5th Workshop on the Next Generation of CASE Tools,* CAiSE'94, Utrecht, the Netherlands, *Memoranda Informatica 94*–25, 1994, pp. 140–148.

Support for Incremental Method Engineering and MetaCASE

S. Kelly and V.-P. Tahvanainen

Department of Computer Science and Information Systems
University of Jyväskylä
P.O. Box 35
FIN-40351 JYVÄSKYLÄ
Finland

Email: {stevek|vpt}@hyeena.jyu.fi

1. INTRODUCTION

In this paper we discuss the problems related to supporting *incremental method engineering*, i.e. the crafting of a design method for a specific IS development project 'on the fly'. A necessity for this approach is a reliable and efficient metaCASE environment, a tool or a set of tools that can be adapted to support different IS design methods, in a manner similar to that of a CASE tool dedicated to that particular method, whilst retaining basic 'look and feel' and functionality across different methods. In addition, a somewhat different set of tools is needed for working on *metamodels*, i.e. descriptions of information system development methods (see e.g. Wel92 for details).

We start the discussion by briefly explaining the principles of incremental method engineering and outlining the specific requirements that they impose upon the computerised support environment.

In Section 3 we discuss metamodels, i.e. models of the design methods that are crafted and used during the incremental method engineering process. We show how the GOPRR data model can function as a basis of method component integration and illustrate the issue with a small example. Finally, we recapitulate the most important results in the conclusions section.

2. INCREMENTAL METHOD ENGINEERING AND ITS CONCEPTUAL PREREQUISITES

The concept of method engineering — analogous to software engineering — was first introduced by Kumar and Welke (Kum88). After Heym (Hey93a) we define it to be *the systematic and structured process of developing, modifying and adapting information systems (IS) development methods*[1] *by describing the components of the method and their relationships*. (See van Slooten & Brinkkemper (Slo93) for a slightly different definition that focuses on the contingencies and purpose of method engineering.) The modelling of the components and their relationships is commonly known as *metamodelling* (cf. Bri90), and it has obvious uses even outside the area of method engineering (Bri89, Hon92). By incremental method engineering we mean the process of method engineering as defined above *applied during the process of IS development in order to craft a specific IS design method for the purposes of the project and taking into account the specific contingencies that affect the current development situation*. The need for method engineering has also been observed in industry (Hid93).

Incremental method engineering obviously requires support by flexible CASE tools, often referred to as metaCASE tools or CASE shells, (analogous to expert system shells, see Bub88) that can be easily adapted to support any IS design method or method component. Some existing tools for this purpose are RAMATIC (Ber89), MetaEdit (Smo91), Virtual Software Factory (Poc91, Hey93b), ToolBuilder (Ald91), and MetaView (Sor88) (see also Mar93 and Har93). In addition to this, however, tools are needed which can create and manipulate models of methods and their components (henceforth referred to as *metamodels*) e.g. MetaEdit or the VSF-based MEET (Hey93b). Further, for incremental method engineering we need at least some way of modelling the

[1]The vocabulary used in referring to these concepts is unfortunately not very well standardized. A *method* in the sense we are employing here is often — especially in American usage — referred to as a *methodology*. For method components, terms like *method* (Lyy89), *technique* (e.g. Bri90) or *fragment* (Har93) have been proposed.

software development process (cf. Wij91), and — to link arbitrary concepts between method components — a way of describing the semantic relationships of concepts (and their respective notations) across metamodels (called route maps by Harmsen & Brinkkemper). Thus, a repository to support incremental method engineering needs at least three levels:

1. **models**: The IS models created with the metaCASE tool(s).

2. **metamodels**: The models of IS design method components that are reused and reconfigured to make the IS design method employed in a project.

3. the **meta-metamodel**: The datamodel with which the metamodels and their relationships can be described and manipulated.

In the following section we take a closer look at the requirements these conceptual models and their efficient manipulation impose. Another approach based on using an existing metaCASE tool is described in (Har93).

3. CONCEPTUAL SUPPORT FOR METHOD ENGINEERING

To provide support for users on both the model level (CASE) and the metamodel level (method engineering), we must maintain consistency of design information along at least two dimensions, namely (cf. Mar93):

1. *Horizontal consistency*, the consistency between semantically equivalent descriptions on the same level, for example between an Entity Relationship model and a Data Flow diagram;

2. *Vertical consistency*, the consistency between descriptions on different levels, for example between an Entity Relationship model and the equivalent metamodel.

We shall look at these two areas in turn, considering first how we can make the necessary links between different design objects which are in some way considered to represent the same real world object, and secondly how we can provide the method engineer with tools which make his job — creation and maintenance of the metamodels which define these links — easier and more efficient.

The GOPRR meta-metamodel (Smo93), based on Welke's OPRR (Wel92), will be used to describe methods. GOPRR can be used to model both the metamodel (type) level and the model (instance) level. It has 5 concepts, set out below with their representations in metamodelling (in modelling, their representations of course change according to the method) and their function:

- Properties, represented as ovals, which contain single data entries such as a name, text field or number (there are also complex properties, represented by double ovals, which can contain a number of properties as a list);

- Objects, represented as rectangles, which contain properties and model concepts like ER Entity and DFD Process (there are also complex objects which contain a breakdown of their internal structure, modelled with the same concepts);

- Relationships, represented as diamonds, which contain properties and model concepts like DFD Data Flow;

- Roles, represented as circles, which contain properties and model concepts such as which end of a relationship is 'to' and which 'from';

- Graphs, represented as rectangles with a Windows-like title bar, which have their own set of properties, and also contain collections of Objects, Relationships and Roles, and even other Graphs as the contents of complex Objects. A Graph type can be used to model a method or method

component specification, e.g. Object-Oriented Analysis (Coa90) or Data Flow Diagram; a Graph could be a Data Flow Diagram describing an order system.

Any Property, Object, Relationship or Role can explode to a Graph. In an OPRR diagram, these symbols are linked to show which Properties go with each Object etc., and the legal ways of connecting Objects of various types via Relationships are shown by a chain of Object-Role-Relationship-Role-Object, referred to as a *binding*.

3.1 Method Integration

To support method integration, we need a meta-metamodel which is first able to cope well with links within a single Graph or Graph type. GOPRR is particularly satisfactory in this respect, as it has advanced capabilities for linking Objects via Relationships and Roles, including n-ary Relationships (i.e. Relationships with more than two Roles), and the ability to define properties for both Relationships and Roles. Objects can have a truly recursive structure, containing other Objects, Relationships and Roles, and maintaining the distinction between these sub-Objects and Properties of the parent Object. In attempting to remove this distinction, Petry (Pet88) falls in to the problem that his approach loses the parent-child connection, and cannot support copying of complete complex objects. Further, instances of all GOPRR meta-types may explode to a Graph, and the conceptual-representational distinction allows re-use of model components: for instance, an Object may have representations in several Graphs, but all of these will reference the same set of Properties.

These, however, are not yet sufficient: we need to integrate between Graphs of different types. GOPRR uses class-based inheritance to ease the task on the metamodelling level, and Wilkes (Wil88) goes one stage further by allowing *instance inheritance*: a model component can inherit values as well as type structure. However, Wilkes links this inheritance to the class inheritance

structure, thus removing the generalised possibility of sharing values between design components whose types are unrelated. The approach we suggest is to allow sharing of Properties: any design component may have Properties, and each component stores its Properties by reference, not value. Hence several components may have references to the same Property, and if it is changed via one component, then all the other components will see it with the new value. This approach has the further benefit of allowing components with different GOPRR meta-types to have the same sets of Properties: for instance, an Entity Relationship Diagram 'Relationship' is actually a GOPRR Object, but may well model the same real world concept as a Data Flow Diagram 'Data Flow', which is a GOPRR Relationship. In the example diagram in Figure 1, three Graphs contain Objects or Relationships, all referring to a real world mercury delay line, and hence all sharing the same properties, and also all exploding to the same decomposition Graph.

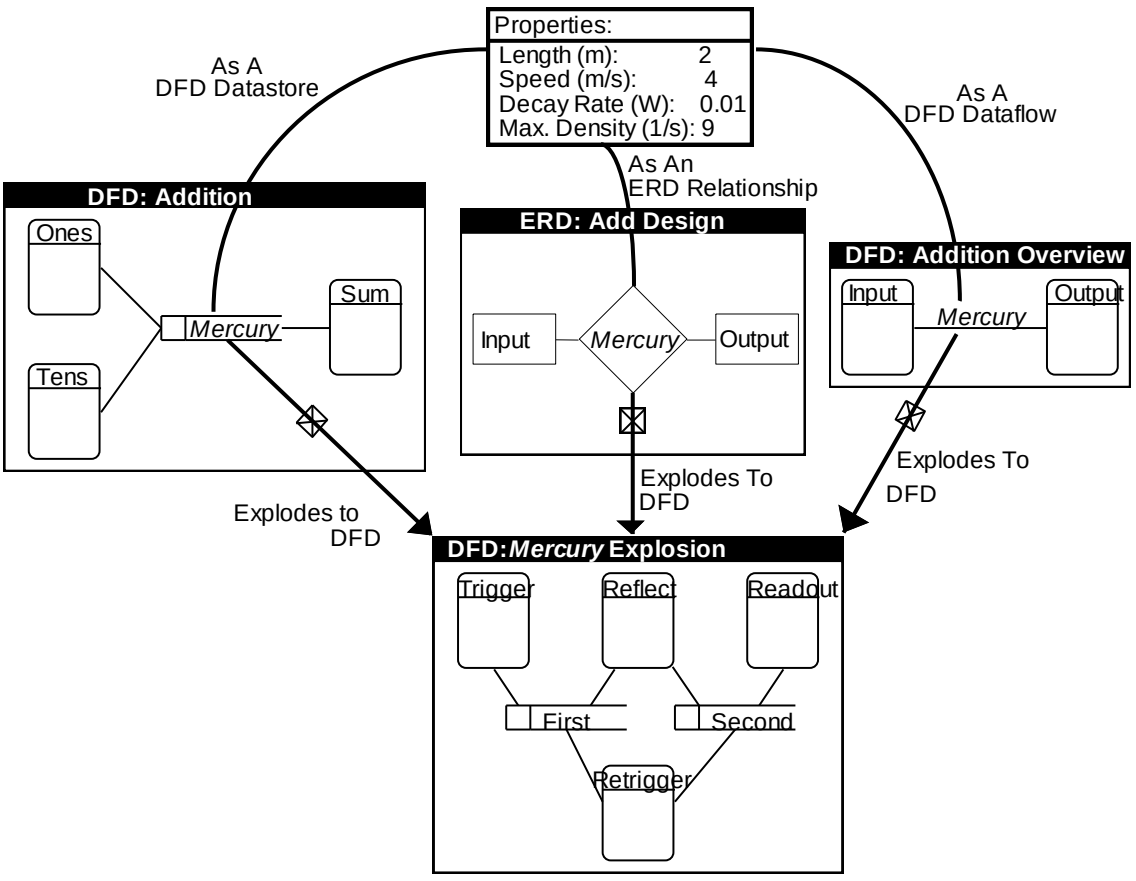

Figure 1: Method Integration by Property Sharing

This approach is sufficient in many cases, but cannot of course handle the general case, where there is some relationship between Properties of two model components, but not one of simple equality. For example, there could be another concept modelling the mercury delay line, which had a Property 'delay'. The value of this can of course be calculated as Length / Speed, but the question of how to provide the method engineer with tools to create such links is a difficult one. For complete generality, nothing short of a functionally complete programming language would suffice. The suggestion of Rumbaugh (Rum88) to tag relationships between object types to show which operations on those objects should spread to objects connected by the relationships could be extended, so that Properties could be connected by various links, and an update to a Property could be propagated to linked Properties in an appropriate way. Dayal et al. (Day88) suggest that Event-Condition-Action rules should be used as first-class objects in the repository, and such rules could be used with an active database to calculate and propagate updates to linked Properties appropriately.

3.2. Method Component Reuse

The purpose of method integration is to allow connections to be made and maintained between dissimilar Graph types. Clearly, when metamodelling an existing method from a book this presents few real problems: we know from the start which connections should be there, and can easily make such links as are necessary. The problems arise in the unfortunately more common situation that an organisation wants to use a variety of methods or method components that were not necessarily designed to work as a coherent set. In this case, it should be possible to use existing Graph types to form a metamodel. In fact, what we need are *links* to existing Graph types: if we merely copy the contents of those Graphs, we create a consistency time bomb, as changes made to one copy cannot be easily propagated to the other copies. Although there will be times when a simple copy is wanted (and hence this functionality too must be

provided), a system which allows the method engineer to see which method components have been used where, and update them collectively or individually, will be invaluable. Further, the inter-Graph links to and from a given Graph type will differ depending on the metamodel, and thus should be defined in the metamodel. Hence only the metamodel 'knows' both ends of the link, and each Graph type 'knows' only its internal information, and can thus be re-used without changes in many metamodels.

A problem arises in trying to represent this metamodel to the user: the sheer size of the methods used in real projects. This rules out the simple approach of showing all the contents of the component Graph types at the metamodelling level. A more reasonable approach is to make a metamodel Graph that contains the constituent Graph types as complex objects, each containing only those types that are part of some inter-Graph link. The ability to use complex objects at this stage is another indication of the power of using the same GOPRR meta-metamodel schema at both the metamodel and model levels: the functionality and user interface need only be made once, and learned once by the user. A further benefit of complex objects in metamodelling is that an Object, Role or Relationship can be collected together with its Properties as a complex Object, and a library compiled of these method building blocks: other still larger blocks can of course be made in the same way. Thus the user is not limited to one definition of 'reusable method component', but can create his own system, modifying it according to need. Hence the incremental approach is not only applied to methods, but also to the process of method engineering itself.

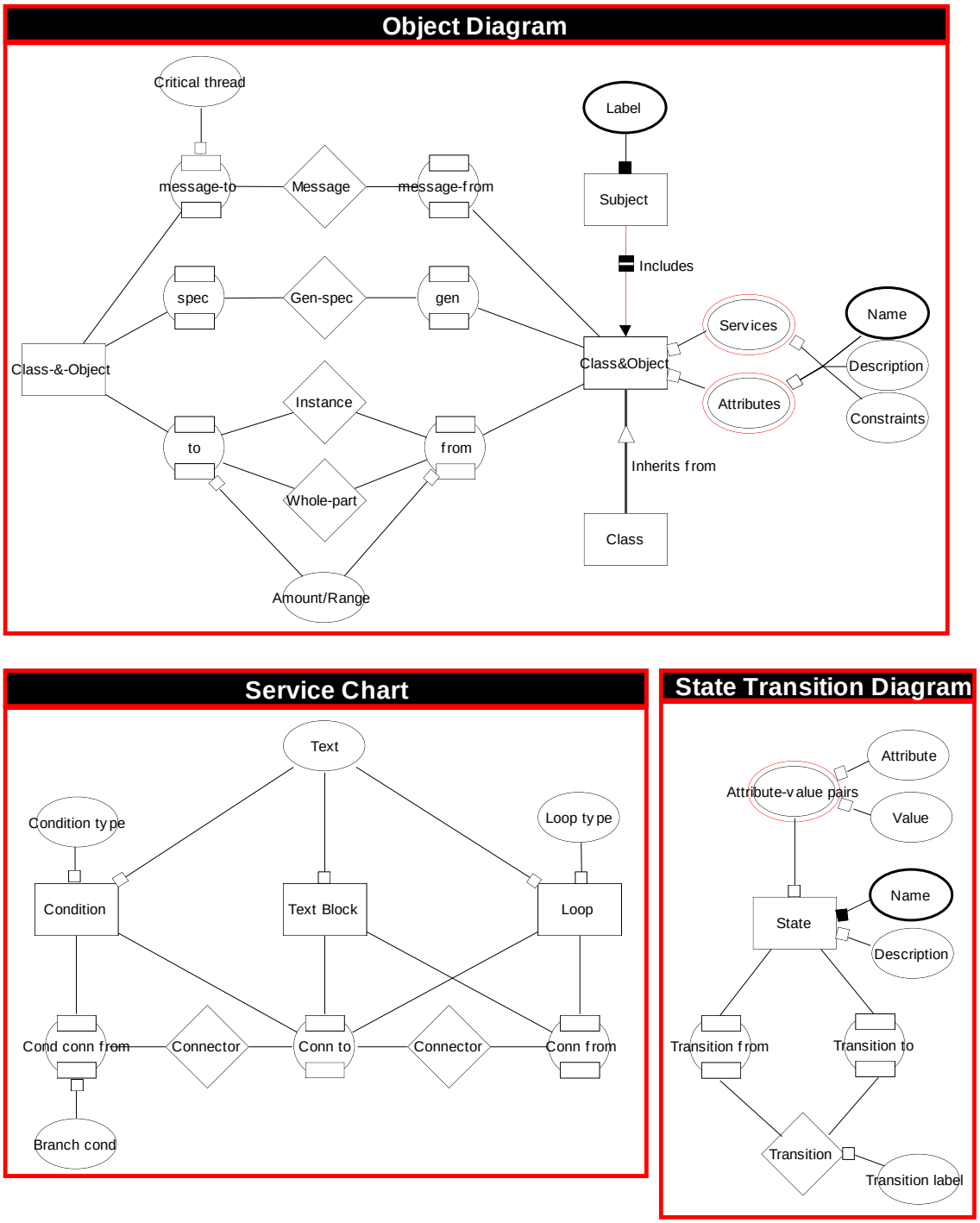

Figure 2: Coad & Yourdon Object-Oriented Analysis Graph Types

The Coad and Yourdon Object Oriented Analysis method (Coa90) is presented here in Figures 2 (Graph types) and 3 (metamodel) as an example, modelled according to this approach. The reader should note particularly that each of the Graph type definitions has been made without any information as to which types would take part in method integration, and therefore could be re-used without copying or changing in another putative metamodel which included,

say, Object Diagrams and Data Flow Diagrams. Also, to change how the method integration works, or to add another Graph type to the method, we would only need to change the final diagram, probably only by adding the Graph type and maybe one object type. This example was deliberately chosen as it is quite small, and thus easy to assimilate. With a larger example, the metamodelling Graph (Figure 3) could be split over several Graphs: the same Graph types could appear more than once as complex Objects in these metamodel Graphs.

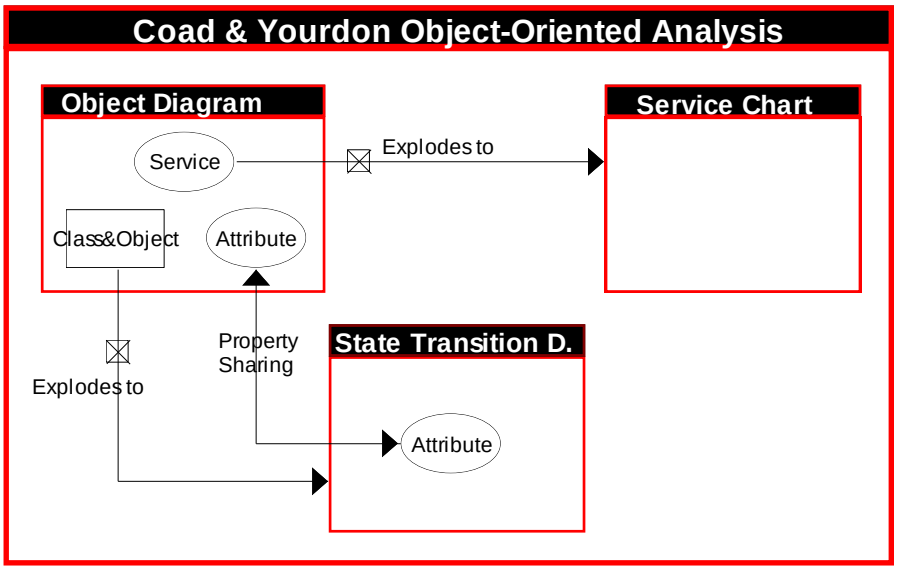

Figure 3: Coad & Yourdon Object-Oriented Analysis Metamodel

As can be seen from Figure 3, the method consists of three Graph types, and the inter-Graph links are as follows: in Object Diagram, Service can explode to Service Chart, and Class&Object to State Transition Diagram, and the Property 'Attribute' in the Object Diagram is the same as that in the State Transition Diagram. In practical terms, the last connection means that when the Property 'Attribute' is being filled in in either of these Graph types, the user can choose to create a new Attribute, or select from among those already created in either diagram.

4. CONCLUSIONS

GOPRR was shown to be a good basis on which to build support for method integration, and extensions were presented to support this and incremental method engineering. In particular, it was shown how method integration could be performed by Property sharing, even between model components of different meta-types; how metamodels could be built of Graph types, without altering the Graph types and thus restricting them to use within that particular metamodel; and how Objects, Relationships and Roles and their Properties could be stored in a metamodel as complex Objects, thus allowing their reuse in different Graph types. Coad and Yourdon's Object-Oriented Analysis was then presented as a practical example, metamodelled according to this approach.

REFERENCES

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

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).

Bri89 Brinkkemper, S., M. de Lange, R. Looman and F. H. G. C. van der Steen, "*On the Derivation of Method Companionship by Meta-Modelling,*" Imperial College, London, UK (July 17–21,1989).

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

Bro91 Brown, Alan W., "*Object-oriented Databases: their applications to software engineering,*" McGraw-Hill, Maidenhead UK (1991).

Bub88 Bubenko, Janis A., "*Selecting a strategy for computer-aided software engineering,*" SYSLAB University of Stockholm, Stockholm (June 1988).

Coa90 Coad, P., E. Yourdon, "*Object-Oriented Analysis,*" Englewood Cliffs, New Jersey (1990).

Day88 Dayal, Umeshwal, Alejandro P. Buchmann and Dennis R. McCarthy, "Rules Are Objects Too: A Knowledge Model for an Active, Object-Oriented Database System," in *Advances in Object-Oriented Database Systems: 2nd International Workshop on Object-Oriented Database Systems*, Springer-Verlag, Berlin (1988).

Har93 Harmsen, Frank, Sjaak Brinkkemper, "Computer Aided Method Engineering based on existing Meta-CASE technology," in *Proceedings of the Fourth Workshop on The Next Generation of CASE Tools*, Univ. of Twente, Enschede, the Netherlands (1993).

Hey93a Heym, Michael, "*Methoden-Engineering Spezifikation und Integration von Entwicklungsmethoden für Informationssysteme,*" Hochschule St.Gallen, St.Gallen, Switzerland (1993).

Hey93b Heym, M., H. Österle, "*Computer-aided methodology engineering,*" INFORMATION AND SOFTWARE TECHNOLOGY 35(**6/7**) (June/July 1993) pp.345–354.

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).

Hon92 Hong, Shuguang, Geert van den Goor and Sjaak Brinkkemper, "A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies," in *Comparison of Object-Oriented Analysis and Design Methods: a Collection of Papers*, University of Twente, Netherlands (1992).

Kat84 Katz, Randy H., "Transaction Management in the Design Environment," in *New Applications of Databases*, Academic Press, London UK (1984).

Kum88 Kumar, Kuldeep, Richard J. Welke, "Methodology Engineering: A Proposal for Situation Specific Methodology Construction," in *Proceedings of CASE Studies 1988*, Meta Systems, Ann Arbor (1988).

Lyy89 Lyytinen, Kalle, Kari Smolander and Veli-Pekka Tahvanainen, "Modelling CASE Environments in Systems Development," in *Proceedings of the first Nordic Conference on Advanced Systems*, SISU, Stockholm (1989).

Mar93 Marttiin, Pentti, Matti Rossi, Veli-Pekka Tahvanainen and Kalle Lyytinen, "*A Comparative review of CASE shells: A preliminary framework and research outcomes*," Information & Management 25 (1993) pp.11–31.

Pet88 Petry, E., "A Model for an Object Management System for Software Engineering Environments," in *Advances in Object-Oriented Database Systems: 2nd International Workshop on Object-Oriented Database Systems*, Springer-Verlag, Berlin (1988).

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

Rum88 Rumbaugh, J., "Controlling Propagation of Operations using Attributes on Relations," in *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, San Diego, California (1988).

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*, North-Holland, Amsterdam (1993).

Smo91 Smolander, Kari, Kalle Lyytinen, Veli-Pekka Tahvanainen and Pentti Marttiin, "MetaEdit — A Flexible Graphical Editor for Methodology Modelling," in *Advanced Information Systems Engineering CAiSE'91 proceedings*, Springer-Verlag, Berlin (1991).

Smo93 Smolander, K., "GOPRR: a proposal for a meta level model," in *MetaPHOR internal technical document* (24.8.1993).

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

Wel92 Welke, Richard J., "The CASE Repository: More than another database application," in *Challenges and Strategies for Research in Systems Development*, Wiley, Chichester UK (1992).

Wij91 Wijers, Gerard, "*Modelling Support in Information Systems Development,*" Thesis Publishers, Amsterdam (1991).

Wil88 Wilkes, Wolfgang, "Instance Inheritance Mechanisms for Object-Oriented Databases," in *Advances in Object-Oriented Database Systems: 2nd International Workshop on Object-Oriented Database Systems*, Springer-Verlag, Berlin (1988).