5

# A Functional Framework for Evaluating Method Engineering Environments: the case of Maestro II/ Decamerone and MetaEdit+

*P. Marttiin [a], F. Harmsen [b] , M. Rossi [a]*
*[a] Department of Computer Science and Information Systems, University of Jyväskylä, P.O.Box 35, 40351 Jyväskylä, Finland*
*E-mail: {ptma, mor}@jyu.fi*
*[b] Department of Computer Science, University of Twente, P.O.Box 216, 7500 AE Enschede, The Netherlands,*
*E-mail: harmsen@cs.utwente.nl*

**Abstract**

CASE environments with method customisation capabilities and Computer Aided Method Engineering (CAME) environments have emerged during the last few years. While many research papers discuss the principles of method engineering and suggest requirements for new environments, we do not have critical evaluations of CAME environments using a wider method engineering framework. The aims of this study are twofold: 1) to build a preliminary framework for comparative studies of CAME environments, and 2) to increase the knowledge of the 'state of the art' in CAME by evaluating two CAME environments. We adapt a functional framework – originally built for CASE technology – to examine the following two research questions: *How well can a method be defined in a CAME environment?,* and *How well is the defined method supported in a customisable CASE environment?* The environments chosen for evaluation are *Maestro II /Decamerone*, and *MetaEdit+*. As an outcome, we will describe what framework aspects these environments support, and discuss the aspects not supported.

**Keywords**

CASE evaluation, metaCASE, method engineering, systems development, co-ordination

S. Brinkkemper et al. (eds.), *Method Engineering*

# 1 BACKGROUND AND MOTIVATION

Despite extensive comparative research on information system development methods[1] during the 1980's (Olle et al., 1982, 1983, 1986), research interest in assessing methods is still strong. According to Norman and Chen (1992), methods have evolved in parallel with changing application domains and to the need to improve method support through tools. Recently, the object-oriented paradigm has yielded numerous new methods and new contexts for applying methods (e.g., business engineering) have emerged. These have given rise to the continuing method evolution. Furthermore, we believe improved mechanisms for method customisation in CASE environments will increase organizations' willingness to adjust and to improve their method practices.

To obtain a better understanding of the study domain and the environments compared, we need first to discuss some basic concepts. Because various definitions and views of CASE abound (e.g., Henderson and Cooprider, 1994), we do not try to suggest clear and exact definitions for CASE and CASE environment. We consider CASE to be a design aid technology for ISs, and a CASE environment a collection of design aid tools for ISs. While traditional CASE environments – based on the fixed repository structure, and containing editors and other tools only for a fixed method – support a few popular methods, a customisable CASE environment – having a flexible repository structure and technique specific editors and tools – is capable to support any method specified into it. We can define a customisable CASE environment more precisely:

> Definition 1. *A customisable CASE environment is a CASE environment having mechanisms to support any method specified into it.*

Brinkkemper (1995) has defined *method engineering* as 'a discipline to design, construct and adapt methods, techniques and tools for the development of information systems'. If the method engineering process is supported by specific computer aided tools, we call the engineering discipline *Computer Aided Method Engineering (CAME)*, and the supporting tools *CAME tools*. According to Harmsen et al. (1994), we define a CAME environment as follows:

> Definition 2. *A CAME environment is a collection of CAME tools for 1) specifying methods to be used in CASE environments, 2) comparing, analysing, and selecting methods, and/or 3) storing the accumulated knowledge of methods and situation factors.*

Bubenko (1988) introduced a term *CASE shell* to denote a tool including mechanisms to define CASE support for an arbitrary method, thus, corresponding to the first part of CAME tool definition. The most common term *metaCASE environment* is used to denote either a CASE shell, or an integrated CASE shell and a customisable CASE environment (Marttiin et

---

[1] Wynekoop and Russo (1993) define a method as 'a systematic approach to conduct at least one complete phase (e.g. design, testing) of IS production, consisting a set of guidelines, activities, techniques and tools, based on a particular philosophy of system development and the target system'.

al., 1993). To avoid misunderstandings, we have selected the terms 'a customisable CASE environment' and 'a CAME environment' to be used later in this study.

Historically, in the late 1970's and in the 1980's, metaCASE environments were studied and designed mainly in academic research laboratories. Such environments include *SEM* (ISDOS, 1981), *RAMATIC* (Bergsten et al., 1989), *MetaPlex* (Chen and Nunamaker, 1989), *MetaEdit* (Smolander et al., 1991), *MetaView* (Sorenson, 1988), and *ConceptBase* (Hahn at al., 1991). During the early 1990's commercial metaCASE environments have appeared in the market. Current metaCASE products include e.g. *Graphical Designer* by ASTI (Advanced Software Technologies, *Toolbuilder* by Lincoln Software/IPSYS, *Paradigm Plus* by Protosoft, *ObjectMaker* by Mark V Systems, *Maestro II* by Softlab, and *MetaEdit+* by MetaCase Consulting.

Each metaCASE environment takes a different view on methods, and they employ different mechanisms for defining the supported method. Furthermore, the supported method aspects vary considerably (see Marttiin et al., 1993; Vessey et al., 1992; Verhoef and ter Hofstede 1995). For example, some environments define integration just for techniques, whereas others tie techniques to the IS development process. This divergence of tools requires analysis and evaluation of metaCASE using a wider method engineering framework.

To construct a framework for new research areas (we can consider method engineering as such one), we need to look at available research and research directions. Brinkkemper (1995) presents some research questions for method engineering: *meta modelling techniques, tool coverage and interoperability, situational methods,* and *comparative review of methods and tools*. We now discuss these questions closely.

## 1.1 Meta-modelling techniques

Meta-modelling techniques deal with issues related to both meta-modelling language and process.

A meta-modelling language is a language for specifying techniques (e.g., OPRR [Smolander, 1992]), interrelations of techniques (e.g., PSM [Ter Hofstede and van der Weide, 1993]), consistency rules and transformations (e.g., ETL [Boloix et al., 1991]), development tasks (e.g., Task Structures [Wijers, 1991]), decisions (Jarke et al., 1994), and tools (Sorenson et al., 1988), as examples. Comprehensive criteria to evaluate meta-modelling languages, such as those proposed to conceptual modelling languages (see Venable, 1993), are lacking. The open research issues include, e.g., *the richness, simplicity,* or *granularity of a meta-modelling language* to define various aspects of a method, which are discussed by Verhoef and Ter Hofstede (1995).

Second, meta-modelling process deals with the steps and actors needed in modelling method (i.e. method engineering steps). The important questions include how to manage method evolution, how to use the method knowledge, and who are participating the method engineering process. Different scenarios on meta-modelling process are discussed in (Harmsen et al., 1994).

## 1.2 Tool coverage and interoperability

To provide support for various method aspects, a customisable CASE environment needs to operate with a large set of tools. The art of integrating all the tools – tools for CASE, software engineering, and project management – is called *interoperability* of tools (Brinkkemper, 1995).

Another issue is the *coverage* of the design aid support for specific purposes. For example, matrix techniques require matrix tools, graphical techniques require graphic tools, and some techniques such as *Petri nets* are hardly useful without formal analysis or simulation mechanisms. Moreover, how do we manage techniques that do not focus on modelling, but rather on idea generation, communication or reasoning? The question of tool coverage can be formulated as: What design tasks need to be supported by specific design aid tools, and what tasks can be managed by general purpose tools (such as text editors and electronic mail), or without any tool support?

## 1.3 Situational methods

Methods are always of a generic nature and dependent on the contextual situations – calling for situational methods. For example, Wijers and van Dort (1990) observed that tools built around fixed handbook techniques and process models hindered projects to use their own dialects. Moreover, finding the *best* or *appropriate* method for a *specific* system may currently put a development project into a trial. For these reasons, approaches for situational method engineering have been introduced (Kumar and Welke, 1992; Harmsen and Brinkkemper, 1995) and related supporting CAME environments designed such as *Decamerone* (Harmsen and Brinkkemper, 1995), and *Method Base* (Saeki et al., 1993).

## 1.4 Comparative review of methods and tools

Several frameworks for methods have been constructed (e.g., Essink, 1986, Olle et al., 1991). Also, a number of frameworks and models for CASE and method support are available (Lyytinen et al., 1989; Wijers, 1991; Heym and Österle, 1992; Henderson and Cooprider, 1994). However, only few studies tackle issues related to CAME. Comparisons (Vessey, 1992; Crozier et al., 1989) have been made by focusing on CASE tool capabilities. The comparison of three metaCASE environments (Marttiin et al., 1993) examined the functionality to specify a new technique. Also, Verhoef and Ter Hofstede (1995) evaluated the feasibility of the conceptual basis of metaCASE environments. Notable is the lack of empirical evaluation of CAME tools: we found only Cronholm and Goldkuhl (1994), which describes five cases of method modification.

This research issue (1.4.) motivated us to evaluate two environments, while the others guided us to focus on prominent method engineering aspects in the CAME framework. Aims of this study are to build a framework to study *How well can a method be defined in a CAME environment?*, and to increase knowledge of the 'state of the art' in CAME by evaluating two CAME environments. A cornerstone in CAME is its meta-modelling language, which raises another research question: *How well is the defined method supported in a customisable CASE environment?* The environments in comparison are: *Maestro II* (Merbeth, 1991) with *Decamerone* (Harmsen and Brinkkemper, 1995) and *MetaEdit+* (Kelly et al., 1996). Analysis and comparison of these environments are carried out and reported based on the two questions above and a detailed framework presented in Section 2.

The study is structured as follows. In Section 2 we introduce the framework and adapt it into CAME. Section 3 introduces the basic architecture and tools of the environments. Section 4 evaluates the CAME part of these environments, and Section 5 discusses shortly customised

CASE by applying the framework. Finally, in Section 6 we draw conclusions and discuss our future work.

## 2 FRAMEWORK FOR CUSTOMISABLE CASE AND CAME

A number of CASE technology frameworks have been proposed (Lyytinen et al., 1989; Misra, 1990; Crozier et al., 1989; Wijers, 1991; Heym and Österle, 1992; Fuggetta, 1993; Henderson and Cooprider, 1994). We selected the 'functional model' for CASE (Henderson and Cooprider, 1994) to be applied for CAME technology. The selection is based on the following reasons: First, the framework takes a comprehensive view of CASE technology. Second, it can be easily adapted to any design aid domain including CAME. The rationale for adapting the framework as such into CAME is found in (Auramäki et al., 1988; Nijssen, 1989). According to their architectural principles, CAME technology – a design aid technology for methods and CASE tools – can be similarly treated with CASE technology. Third, the model is based on empirical studies and contains several strictly formulated questions for CASE. This is the main limitation of other framework candidates. Furthermore, our earlier comparison (Marttiin et al., 1993) based on the framework of Lyytinen et al. (1989) did not focus extensively on CAME aspects, and it concentrated on issues not relevant for this study (e.g. portability to use various operating systems, and DBMSs).

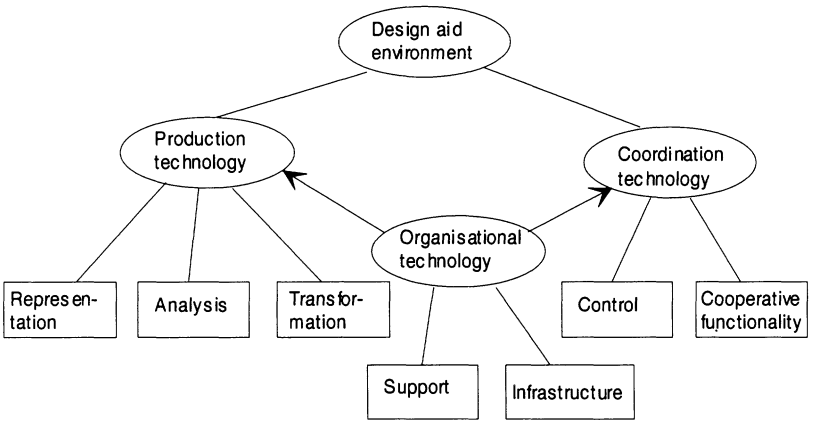


**Figure 1** A functional model of CASE technology (Henderson and Cooprider, 1994).

Henderson and Cooprider (1994) conceptualise design aid technology as a combination of *production*, *co-ordination*, and *organisational* technology (Figure 1). Each of these main functions divides into sub-functions. In the following we present the main issues of each sub-function, discuss shortly the functions for customisable CASE, and try to capture the basic functions a future CAME environment need to supply.

## 2.1 Production technology

Production technology (as an individual's point of view to analyse, design and generate products) is divided into components of *representation*, *analysis* and *transformation* (see Figure 1). Representation focuses on abstraction and conceptualisation of phenomena into models. Analysis reflects the problem solving and decision making aspects of development. Transformation calls for rules and mechanisms to transform models into another form.

### *Customisable CASE*

Representation in customisable CASE calls for the possibility to model ISs using various techniques, and to support modelling using various notations (textual, diagram, matrix, tabular) by corresponding editors (Chen and Nunamaker, 1989). As a modelling support, representation function deals with issues on creating, editing, composing, integrating, retrieving, and viewing IS models and components of them.

Analysis requires verification, validation and simulation support for IS models. Verification deals with issues such as consistency checking, rules, equivalencies and redundancies in IS models, change analysis, and querying IS models (Henderson and Cooprider, 1994). Validation can be achieved by using metrics, decision aids, requirements tracing, or supporting versions of models. Simulation is used for testing the completeness and performance of IS model by running it.

Transformation considers, for instance, how to transform a logical model into a physical one. The other issues include reverse engineering, change propagation, generation of reports, documentation, and code, and the generation of screen mock-ups and executable code for prototyping.

### *CAME*

While the representation function for customisable CASE focuses on how to model ISs, the representation function for CAME focuses on how to specify methods, and how to manage their use in CASE. This is related to meta-modelling techniques discussed in Section 1.1. We consider here *meta-modelling languages*, *abstraction mechanisms* and *notations* used in modelling methods.

- According to earlier studies (Brinkkemper, 1990; Heym and Österle, 1992; Jarke et al., 1994, Marttiin et al., 1995), the suggested method fragments (i.e. a method specification or part of it) contain some features of all the CASE framework functions. Thus, in practice the primary focus has been on the production: defining a method for representing, analysing, and transforming IS models. As noted in our earlier comparison (Marttiin et al., 1993), we need a richer meta-modelling language to be able to capture more details of any method aspect. This will lead to consider both the feasibility and the possible granularity of any supported method aspect (Verhoef and Ter Hofstede, 1995).
- In some cases the method fragments are produced from 'scratch', but the need to store them into a method base, and reuse and recompose them requires the use of abstraction principles. Basic abstraction principles used in conceptual modelling (Brodie, 1984): *generalisation – specialisation* and *aggregation – decomposition* are suitable for method fragments. Further, the research on meta-modelling hierarchies (Oei and Falkenberg, 1994) introduces also *restriction* and *degeneration* principles when discussing changes in the meta-modelling language and its effects on the power of a modelling language.

- CAME representation deals with editing and viewing method fragments in various notations: structured text, forms, matrices and graphics, as examples. CAME tools have basically supported textual notations (see Marttiin et al., 1993), although graphics are used in *MetaEdit* (Smolander et al., 1991), and *GraphicalDesigner*.

Analysis covers *verification, validation* and *simulation* of *method fragments*.

- Verification for methods is mostly situation independent, and thus it can be better formalised and implemented than validation. Verification includes consistency requirements for methods including precedence, input/output, and granularity consistency or the checking of duplicate concepts and uncompleted relationships in methods.
- Kumar and Welke (1992) pointed out that method engineering follows the incremental learning strategy: every time a project starts the experience and 'wisdom' from earlier successful and unsuccessful projects are accumulated and included into the method fragments. Validation of the method fragments can be done by comparing them with a number of project factors such as available domain knowledge, the technology used, the organisational size, and the amount of resources available (Slooten and Brinkkemper, 1993). Some references (e.g., Euromethod, 1994; Harmsen et al., 1995) give high-level heuristics to match situations with suitable method fragments.
- A specific simulation can be accomplished when following the 'method engineering by example' strategy, where method components are modelled in the same form as they will appear in CASE as in 'query-by example'.

Transformations in method base can be divided into *generations between various levels of method fragments*, *implementation transformations* and *document generation*.

- Transformations between method fragments can be used to partly automate the production of situational methods or to transform a method fragment from the coarse-featured form into the detailed one. Situational methods may require a possibility to combine a set of rules for selecting elementary method fragments and composing them into a rough 'method template'.
- Implementation transformation occurs when we transform a method fragment into an executable form for a customisable CASE environment, or into a form required by another CAME environment. An example of the former is the transformation from *Decamerone* to the *Prolan* language used in *Maestro II* (Harmsen and Brinkkemper, 1995). The bridge between *MetaEdit* and *RAMATIC* (Rossi et al., 1992) serves as an example of the latter case. Transformations between CAME and CASE have mostly been uni-directional. However, to manage IS model updates when changing a method may require more seamless solutions to integrate the two levels.
- Creation of method documentation belongs to CAME representation. However, its generation into a CASE tools' help is a specific transformation issue.

## 2.2 Co-ordination technology

Co-ordination technology includes functions of *control* and *co-operation*. Henderson and Cooprider discuss control in terms of *resource management* and *access control*. Resource

management enables managers to utilise project resources consistently with project goals. Access control implies, e.g., ways to manage access rights to user groups that participate in the development. It is closely related to database technology and its mechanisms.

Alternatively, co-operation enables to exchange information between developers (co-operative modelling) and users (user involvement) for the purpose of influencing the ISD process, or product. Design aid tools and methods themselves can be used for co-operative purposes: the use of graphical models in co-operation is a case in point. On the other hand, using a specific functionality can increase co-operative support for the design aid: both for CASE and CAME.

### *Customisable CASE*

To establish control mechanisms for the customisable CASE environment, we need to deal with, for instance, the concepts of *process*, *project* and *user roles* (Curtis et al., 1992). Development process is both project and method dependent. A process model constructs development tasks into precedence order, integrates techniques and tasks, and allocates users to participate in the tasks. Process models in customisable CASE are discussed in, among others, (Jarke et al., 1994; Marttiin, 1994). The characterisation of project and user roles is discussed in (Hahn et al., 1991, Curtis et al., 1992, Marttiin et al., 1995). Other control issues relate to project management such as schedules, deadlines, project complexity metrics, and quality assurance.

Co-operation for customisable CASE calls for possibility to use co-operative tools, which support messages, notes, anonymous feedback, announcement of changes, and design rationale concerning models or development tasks. We can divide the support into asynchronous communication (electronic mail as an example) and synchronous modelling (several developers editing the same model simultaneously). Other more advanced group interaction mechanisms include brainstorming and other group development techniques that are currently managed using separate tools. The possibility to integrate these into CASE is a challenging future task.

### *CAME*

Method engineering requires the co-ordination and organisational functions, though these do not have such a prominent position as in the production function. The current CAME literature does not tackle the co-ordination issues involving multiple method engineers or stakeholders.

We can discuss, however, control and co-operation aspects of method engineering. Controlling method evolution is critical, in particular when the changes to a method take place during the IS development and create effects on CASE and IS models. Such effects arise for example when one deletes a concept in a technique or an attribute of a concept. Co-operation in CAME can be seen as an exchange of method engineering experiences and a use of the collected expertise (Kumar and Welke, 1992). As a note to user involvement aspects, method users may include system designers, software engineers, and project managers.

## 2.3 Support and Infrastructure

All production and co-ordination functions can be supported by using organisational technology: the *infrastructure* catching standard operating procedures and quality standards, and the *support* functionality dealing with organisational guidelines, on-line helps, learning aid, 'user friendliness' and 'easiness', which can assist users to understand and use design aid effectively.

**Table 1** A functional framework of customisable CASE and CAME

| *Function* | *Customisable CASE* | *CAME* |
|---|---|---|
| Representation | • Modelling of ISs using various methods: *concepts, notations, and abstraction mechanisms to represent IS models* | • Modelling of methods using a meta-modelling language: *concepts, notations and abstraction mechanisms to represent methods* |
| Analysis | • Verification of IS models: *consistency checking for IS models*<br>• Validation of IS models: *traceability to requirements, design rationale of IS models*<br>• Simulation of IS models | • Verification of methods: *consistency checking for methods*<br>• Validation of methods: *situational methods, analysis on earlier methods, the use of method base, design rationale of methods*<br>• Simulation of methods: *'method engineering by example'* |
| Transformation | • Transformations between IS models<br>• Code and report generation from IS models<br>• Generation of screen mock ups and executable code for prototyping | • Transformation from situation factors into methods<br>• Generation of methods into CASE: *generation of method documentation*<br>• Generation of CASE tools & management of repository mappings |
| Control | • Resource management: *resource capacities, organisational goals, deadlines, priorities, managerial control for CASE*<br>• Access and change control for IS models<br>• ISD process models: *management of ISD tasks and deliverables, automation of CASE* | • Resource management: *resource capacities, organisational goals, deadlines, priorities, managerial control for CAME*<br>• Access and change control for methods: *change effects into CASE*<br>• Method engineering process models: *management of ME tasks and deliverables, automation of CAME* |
| Co-operation | • CASE as co-operative aid: *notes, design rationale support for IS models*<br>• Technical support for co-operative CASE: *group interaction mechanisms, asynchronous/ synchronous CASE* | • CAME as co-operative aid: *notes, design rationale support for methods*<br>• Technical support for co-operative CAME: *group interaction mechanisms, asynchronous/ synchronous CAME* |
| Support & infrastructure | • Help, learning aid and organisational policies for the areas of customisable CASE representation, analysis, transformation, control, and co-operation | • Help, learning aid and organisational policies for the areas of CAME representation, analysis, transformation, control, and co-operation |

*Customisable CASE and CAME*

Organisational support for CASE involves help, learning aid and organisational policies for any CASE framework functions: representation, analysis, transformation, control, and co-operation. As we discussed earlier, method guidance can be modelled during the method engineering process and generated into customisable CASE. In this manner the maintenance of method guidance will be easier when methods evolve.

Similarly to customisable CASE, organisational support for CAME means help, learning aid and organisational policies for the areas of CAME representation, analysis, transformation, control, and co-operation.

The main issues (a bullet for each) and some illustrative examples (italics after colon) discussed in this Section are summarised into Table 1.

# 3 OVERVIEW OF THE ANALYSED ENVIRONMENTS: MAESTRO II/ DECAMERONE AND METAEDIT+

In this section we take an overview to the architecture and tools of *Maestro II/ Decamerone* and *MetaEdit+* environments.

## 3.1 Maestro II and Decamerone

The analysis in respect to the tools and functions of environment is based on *Maestro II* product (Merbeth, 1991) and the design of *Decamerone* (Harmsen and Brinkkemper, 1995).

Maestro II is a metaCASE environment offering experienced developers all possibilities to develop diagram editors, repository structures, process enactment mechanisms, etc. It is a multi-user environment based on a client/server architecture. It includes the following tools:

- Repository tools: *Object Management System (OMS)*,
- Model editing tools: *Text Developers System* and *Graphics Editor*
- Project and process management system: *Project and Configuration Management System (PCMS)*
- Method management tools: *Data Model Declaration Table (DMDT)*, *Tool Customisation Interface (TCI)* including *Symbol Editor*

Figure 2 depicts the architecture of *Decamerone*, which is implemented in Maestro II, and consists of a CAME and a CASE component.

The CAME component is built upon a *method base management system* (*MBMS*), and provides facilities for specifying, storing, and selecting method fragments, and for assembling method fragments into a situation method. The selected method fragments are stored in a *Selected Method Fragments Repository* (*SMFR*), from which they are retrieved by the assembly functions.

CASE component uses the situational method as a definition for its repository structure, its editors and report generators, and its process engine. Method fragments are specified and manipulated both by entering specifications and issuing commands in a textual method engineering language called *MEL*, and by a tool called *MEL editor*. Both of these translate commands to a *MEL interpreter*, which makes use of the MBMS facilities. Decamerone offers

also graphic tools. *The Concept Structure Diagram (CSD) editor* acts as a graphical user interface to the Maestro II's Data Model Declaration Table. The *Process Structure Diagram (PSD) editor* enables definition of task classes, deliverable classes and their relationships, yielding a task structure that is input to the Maestro II's PCMS. The CSD/PSD/MEL editors are currently being implemented.

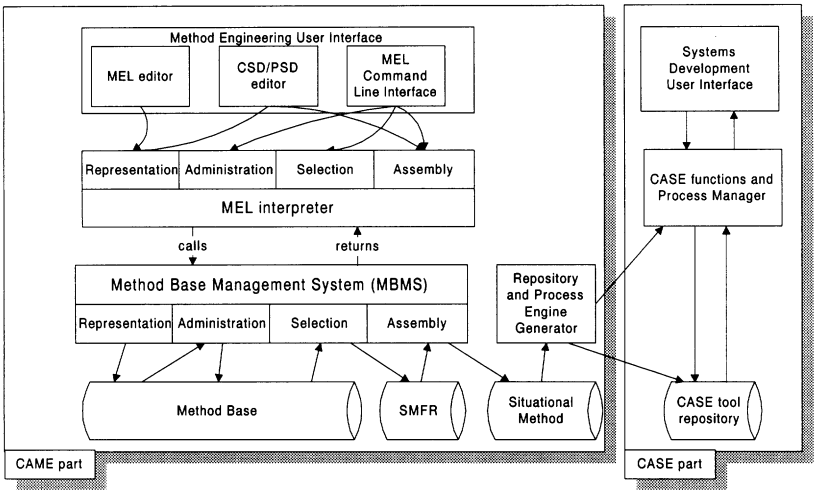


**Figure 2** Architecture of Decamerone.

## 3.2 MetaEdit+

The analysis is based on *MetaEdit+* product (MetaEdit+, 1995; Kelly et al., 1996), and two prototypes: hypertext sub-system (Oinas-Kukkonen, 1995a, 1995b), and process sub-system (Marttiin, 1994; Koskinen, 1996).

MetaEdit+ is a multi-user and multi-tool environment developed in the MetaPHOR-project. It consists of tools for both CAME and CASE. MetaEdit+ can run either as a single-user workstation or simultaneously on many workstation clients connected by a network to a server. Each client contains a running instance of MetaEdit+, including *MetaEngine* and a set of tools. MetaEngine manages all operations on the underlying conceptual data model. Tools communicate with each other only through the MetaEngine, and thereby through the shared data in the repository. MetaEdit+ includes the following tools:

- Environment management tools: tools for managing features of the environment, its main components, and for launching it.
- Model editing tools: tools for creating, modifying, viewing and deleting models or their parts, and deriving new information from existing design information including *Diagram Editor*, *Matrix Editor*, and *Table Editor*.

- Model retrieval tools: tools for retrieving design objects and their instances from the repository for reuse and review including *Repository Browsers*, *Report Tool,* and *Query Editor*.
- Method management tools: a set of form based tools for defining methods and their components (see Figure 3).
- Hypertext subsystem, which gives, for instance, the ability to link design objects for traceability, annotating model instances, and maintain conversations about design issues (*Debate Browser).*
- Process subsystem containing a set of form based tools for defining a process modelling language and *Process Editor* for modelling ISD process and using it for guidance and co-ordination purposes.

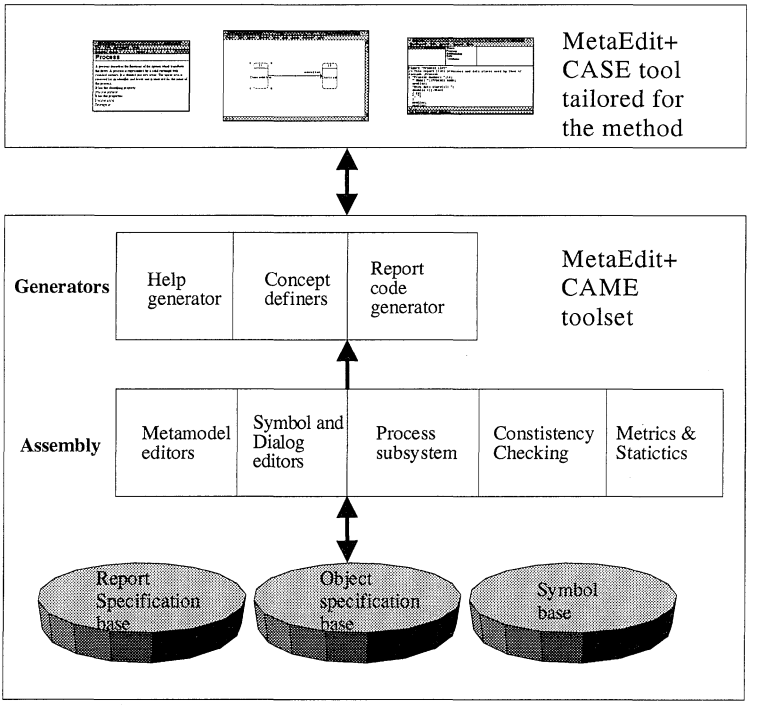


**Figure 3** Method management tools in *MetaEdit+*.

Assembly in Figure 3 includes a set of editors for defining method fragments (method's conceptual structure including basic consistency rules, and method's process structure), method symbols, dialogues, and helps, and tools for producing method specific reports and code. These editors contain mechanisms to generate a method to be automatically used in CASE.

The MetaEdit+ server forms the repository holding all the data contained in models, and also in the method fragments. The MetaEdit+ repository includes *object specification base* containing all the method fragments; *symbol base* containing all symbols needed to represent method concepts; and *report specification base* containing all report and other output specifications. The repository holds also tool related information including spatial co-ordinates in a diagram, and user information including passwords, access rights, and current locks.

## 4 DEFINING A METHOD BY USING THE CAME ENVIRONMENTS

In this section we analyse the CAME functions in both of the environments. Table 2 presents the supported CAME functions.

**Table 2** Supported CAME functions

| *CAME support* | *in Maestro II/Decamerone* | *in MetaEdit+* |
|---|---|---|
| Representation | • *CSD/PSD editor*<br>• *MEL editor* | • Form based meta-modelling tools: *Graph tool, Object tool, Property tool, Relationship tool, Role tool, Binding tool, Symbol Editor*<br>• Other meta-modelling tools: *Table Editor, Matrix Editor and Diagram Editor*<br>• Form based process meta-modelling tools & *Process Editor* |
| Analysis | • *MEL editor*: consistency checking | • Meta-modelling tools: consistency checking<br>• *Process Editor*: task precedence |
| Transformation | • Transformations from CSD/PSD to MEL, and from MEL to MBMS<br>• *Repository generator*<br>• *Process engine generator* | • Form based meta-modelling tools: method generation into customisable CASE, method help generation<br>• *Report Generator, Process Editor* |
| Control | • | • Access control for methods |
| Co-operation | • The use of various notations of method fragments<br>• Co-operative information in MEL | • The use of various notations of method fragments |
| Support & infrastructure | • | • Help for meta-modelling language and tools |

## 4.1 Representation

We evaluate here the meta-modelling language, abstraction mechanisms and notations used in CAME.

### *Meta-modelling language*

*Decamerone* uses MEL (see Harmsen and Saeki, 1996) as a meta-modelling language. MEL is based on the basic type *method fragment*, the instances of which varies by attaching pre-defined or user-definable *property types* and *property values*. MEL allows for an integrated view on both the conceptual view of a method by providing *process fragments* and *product fragments* and the technical view (CASE tool part) by allowing to define *technical method fragments*. Process fragments can denote iteration, parallelism, non-determinism and decisions, whereas product fragments describe both 'high-level' method products (such as Functional Specification) and diagrams, concepts, and associations. Further, MEL offers facilities to anchor method descriptions in an ontology, and contains operations to administrate method fragments in the method base, to query them, and to assemble method fragments into a situational method.

*MetaEdit+* uses GOPRR (Kelly et al., 1996) as a meta-modelling language. GOPRR contains a set of elementary types (*object, role,* and *relationship*), a concept for collecting the elementary types into model (*graph*), and mechanisms to decompose and structure elementary concepts (*binding*). GOPRR focuses on the modelling of the conceptual structure of techniques and various relations between these (explosions, polymorphic modelling concepts). A process modelling language can be defined using an extended GOPRR model called *GOPRR-p* (see Koskinen, 1996). The extensions contain information about the behavioural aspects of a process, e.g., states, precedence and parallelism of process elements. Moreover, related product (models or reports) and tool information can be attached to process elements.

### *Abstraction mechanisms*

*Decamerone* supports aggregation and decomposition mechanisms: method fragments defined by MEL can be decomposed to detailed granularity level, and the detailed method fragments can be composed into various method fragments in general granularity level. Specialisation is supported by the IS_A keyword in MEL.

*MetaEdit+* supports aggregation and decomposition of methods by collecting reusable elementary method types using the concept *Graph*. Further, a set of graphs is collected into a project. Instances of each GOPRR element are modelled using specialisation hierarchies. Therefore, *MetaEdit+* supports different ways of technique development: creation from scratch, where all the parts of the technique are defined as new concepts, component oriented, where techniques are constructed by using prefabricated parts, and reuse oriented, where the goal of technique development is to allow maximal generality of the concepts and then to specialise these general components for different techniques.

### *Notations*

*Decamerone* provides two ways to represent method fragments: graphically, by using the CSD/PSD editor, and textually, by using the MEL editor. The graphical representation format is intended for initial and global specification, whereas the MEL specification contains all details of the method fragment. Process Structure Diagrams support the notions of process, trigger, and product, of which the latter provides the link with the Concept Structure Diagram editor.

In *MetaEdit+* product the primary support for technique component definition and retrieval is available as a set of form based tools, where the attributes of the component are filled and checked for their internal validity. Also, tabular, graphics and matrix support have been designed. Process structure is defined graphically.

## 4.2 Analysis

Validation and simulation of method fragments are not currently supported in either of the CAME environments. Therefore, we only consider the verification capabilities.

The design of *Decamerone* currently does not include a verification component. However, MEL is designed to attach information for checking consistency into its property types. Also, formalised consistency rules are available, and can be transferred to Prolan rules (Hoef and Harmsen, 1995).

The method consistency is ensured in *MetaEdit+* by implementing a GOPRR technique, which prevents the definition of syntactically incorrect methods. *MetaEdit+* approach aims to give the method engineers a maximum degree of freedom, so the tools do not try to do any semantic checking for the models. Anything that can be defined with the meta-modelling tools, is a valid technique specification in this approach. Nevertheless, a number of checks and quality reports have been defined to inform the method engineer about the possible problems in his model. Analysis according to development process, e.g. task precedence, will be supported by using a process model.

## 4.3 Transformation

In the following we discuss both transformations between various forms of method fragments, and transformations into CASE including documentation generation.

### *Transformations between method fragments*

In *Decamerone* graphics (CSD/PSD) are used to define method fragments in earlier phases of CAME. These forms are transferred into MEL specifications, in the form of which the detailed method fragment is constructed.

*MetaEdit+* can use table, matrix and graphics notation in the CAME process. Yet, form based meta-modelling tools can only attach symbols into method concepts and generate a method into CASE.

### *Transformations into CASE*

MEL specifications in *Decamerone* are transformed into MBMS calls in *Maestro II*, enabling storage in the database. The finest granularity product fragments (concepts, their properties and associations) in the situational method are input to the repository generator. The process fragments in the situational method, along with the coarser granularity product fragments (e.g.,

a functional specification report) are transferred to the process engine generator, which creates an instance of *Maestro II's* PCMS.

Diagram editors are specified in *Maestro II*. The method data model and the representational aspects of the technique to be supported are specified separately. The method data model and conceptual definitions of all diagram editors are specified in a Data Model Declaration Table (DMDT). The notational elements are specified by using the Tool Customising Interface (TCI). The DMDT defines the repository structure managed by the Object Management System (OMS).

*MetaEdit+* directly translates the method fragments into parts of its design object repository. Thus, when the user has defined the techniques: concepts and their representations, the method can be tested immediately. The form based tools have a two-way connection to the repository, but the tools using other representation form can only define components, but not retrieve them from the repository. *MetaEdit+* will test the use of pre-defined process templates to be copied and specified for projects' use.

*Document creation*

In *MetaEdit+*, method specific helps can be generated as a by-product of a method, and method specific reports can be created by using a Report Generator. Learning material and examples of *MetaEdit+* are separated, but implementable as external documents using hypertext subsystem.

## 4.4 Co-ordination: control and co-operation

The CAME environments in comparison are focused on production. However, some support for control and co-operation is found.

Control mechanism of *MetaEdit+* multi-user environment is designed also for CAME level. Co-operative aid is supported by the use of various representations of a method in each CAME environment. Also, *Decamerone's* MEL language is designed to attach situation information not generated into CASE, but usable in co-ordination of method fragments. Anyway, specific CSCW tools are not integrated.

## 4.5 Support

*MetaEdit+* contains on-line help including descriptions of GOPRR concepts and guidelines how to use them. Since the environments are still at the early stages, the ways of supporting will improve.

# 5 SUPPORTING A METHOD BY USING THE CUSTOMISABLE CASE ENVIRONMENTS

In this section we shortly discuss the differences of the customisable CASE environments in their method support. Table 3 represents the functions and tools (*italics*) of the customisable CASE environments.

## 5.1 Representation

In contrast to drawing tools, both *Maestro II* and *MetaEdit+* are concept-oriented environments. Concepts are implemented as classes, which store information into their properties. In *Maestro II* the integration between techniques is achieved by using the DMDT, which contains references to object classes, relationships and attributes. Attributes can be shared by referring to the defining template. The following integration mechanisms are used in *MetaEdit+*: concept reuse between models, property sharing between concepts (e.g., object appears as a relationship in another technique), and explosions from an element to a model of different type.

Both CASE environments separate the conceptual and representational (notations) information. Representation forms for *MetaEdit+* are structured graphics, matrices and tables and for *Maestro II* structured graphics and text. Both of the CASE environments handle the basic graphical semantics. However, they are limited to deal with, for instance, layered complex models in the same diagram, and specific graphical rules and constraints.

**Table 3** Supported CASE functions

| *CASE support* | *in Maestro II* | *in MetaEdit+* |
|---|---|---|
| Representation | • *Tool Customisation Interface (TCI)* including *Diagram Editor*<br>• *OMS Interface* | • Model editing tools: *Diagram Editor, Matrix Editor, Table Editor*<br>• *Repository Browser* |
| Analysis | • *Prolan* language: user defined rules<br>• Query mechanism | • Model editing tools: consistency checking<br>• *Report Editor*: reports for checking<br>• *Query Editor*<br>• *Linking Ability*: requirement tracing |
| Transformation | • Report generation<br>• Code generation<br>• Links to other CASE tools | • *Report Editor*: generation of reports and code |
| Control | • *Project Configuration and Management System (PCMS)*<br>• *Function Point Analysis* -tool | • *Project tool*<br>• *Process Editor* |
| Co-operation | • *PCMS*<br>• E-mail facilities | • Hypertext tools: *Linking Ability, Debate Browser*<br>• *Process Editor* |
| Support & infrastructure | • On-line help | • Hypertext tools: *Debate Browser*<br>• Tool guidance<br>• Context specific method guidance |

## 5.2 Analysis

*Maestro II* allows to define a wide spectrum of method specific rules by using a Prolan rule language. In *MetaEdit+* GOPRR manages a set of general rules (cardinality, connectivity and composition rules) and offers possibilities to define checking reports.

Validation of IS models in *MetaEdit+* means traceability to requirement documents. This functionality is provided by the Linking Ability. *Maestro II* offers no support for validation.

Simulation is not used in conceptual modelling domains, but essential, if we want to support process and behaviour modelling by using, for instance, State Transition Diagrams or Petri nets. Both environments do not offer simulation mechanisms.

## 5.3 Transformation

*Maestro II* provides transformation facilities such as advanced report generation, code generation, screen mock-ups, and reverse engineering. It has facilities to generate reports of the database (OMS) and project (PCMS) contents. Besides, there exist tools that provide links with other CASE tools such as Knowledgeware's ADW (Bosua and Brinkkemper, 1995).

*MetaEdit+* contains a *Report Tool* to define report models and programming language structures. Therefore, transformations from a higher level model to a detailed one or a data structure of programming language can be specified.

## 5.4 Control

As noted before, *Maestro II* contains a Project Configuration and Management System (PCMS), which guides the user in applying a method by offering references to required tasks and deliverables. In addition, the state automaton of PCMS defines the dynamics of a project by keeping track of the various states of deliverables as well as their state transitions. Freeze and unfreeze of artefacts, and configuration and version management can be achieved. A project manager can keep track of the current state of all the activities performed by project members and compare their activities with the project plan by using the project scheduling tool incorporated in the PCMS. For estimating, a Function Point Analysis tool is developed as an extension to PCMS.

*MetaEdit+*'s repository is an object base, which stores the objects as such. Repository provides the locking mechanism to avoid simultaneous editing of the same diagram. However, current implementation does not provide versioning of models. GOPRR provides the automatical change propagation between the model elements that share properties. Further, the Process Editor for the guidance and co-ordination of the project specific development process is currently being implemented.

## 5.5 Co-operation

The CASE environments contain minor differences in their co-operative support. Both are multi-user environments having locking mechanisms for asynchronous development (one developer at a time). *Maestro II* supports communication by its e-mail facility. *MetaEdit+* co-operative functionality is found in its hypertext subsystem containing, e.g., annotations and debates attached to either modelling concepts or their representations. Moreover, by using a *DebateBrowser*, structured conversations can be managed. Still, specific group interaction

mechanisms and functionality for synchronous modelling are lacking in these CASE environments.

## 5.6 Support

Dependent on the method or the tool used, *Maestro II* provides on-line help screens in various levels of detail. Help screens are laid out as hypertexts, enabling the user to cross-reference other topics.

*MetaEdit+* offers on-line help to its general CASE tool functions. Context specific aid, descriptions of concepts, techniques and tasks, are generated as a by-product of their creation in method engineering tools. Explanations and discussions according to project tasks, models, model elements, model versions can be attached by using the hypertext support discussed above. Furthermore, the joint menus and functions of various CASE tools are designed similarly.

# 6 CONCLUSIONS AND FUTURE WORK

In this study we developed a framework for evaluating CAME and customisable CASE properties of current 'state of the art' environments. We selected the framework for CASE functionality presented by Henderson and Cooprider (1994) and tailored it for customisable CASE and CAME. We used the comparison framework to evaluate the properties of two environments *Maestro II* with *Decamerone*, and *MetaEdit+*.

The aim of this study was to answer the questions*: How well can a method be defined in a CAME environment?,* and *How well is a defined method supported in a customisable CASE environment?* As a result we presented the functionality and tools that are available in two environments, and functionality required after the framework.

If we consider the framework used, we can conclude that current CAME technology focuses on production, but does not deal adequately with co-ordination and support. Further, in production technology there exists several issues that have not been sufficiently examined, such as specifying the methods in detailed granularity, reuse of method fragments, and change propagation to customisable CASE.

If we look at the second question, we can conclude that in the evaluated environments method support can be achieved by representing and storing models and by checking their consistency. Also, the development process is either supported or under construction. Validation support is limited and behavioural semantics of modelling elements for, e.g., simulation of Petri-net diagrams is not available. Moreover, both environments are limited in their integration of tools for specific tasks, including co-operation, project management, and learning issues.

The main purpose of the environments examined is to improve IS development support by tailoring a method specific CASE for project needs. The quality of such a tailorable environment means the support for the methods required by customers, the methods without any hard-coded CASE support, and the freedom to modify methods. Further, this study explained the framework functions supported. It did not focus on usability or integrity of the tools supporting these functions. Therefore, one should not draw a straightforward conclusion that an environment supporting all framework aspects is a good one.

In the future we would need to use a more comprehensive approach to evaluate these environments. First, we will generate a set of exact questions for each CAME function. Second, we will select an example method to be modelled for the CASE environments. By using this comprehensive approach for evaluating the environments we will obtain a detailed view of the environments. The evaluation of other environments should also be performed.

## 7 ACKNOWLEDGEMENTS

We would like to thank the other members of the projects MetaPHOR (University of Jyväskylä) and Method Engineering Group (University of Twente), and in particular Sjaak Brinkkemper for his assistance and co-operation, and Steven Kelly, Kalle Lyytinen, and Tuuli Rossi for the improvements on this paper.

## 8 REFERENCES

Auramäki, E., Leppänen, M. and Savolainen, V. (1988) Universal framework for information systems. *Data base*, **19**, 1, pp. 11-20.

Bergsten, P., Bubenko, J., Dahl, R., Gustafsson, M. and Johansson, L.-Å. (1989) RAMATIC - a CASE shell for implementation of specific CASE tools. TEMPORA T6.1 report, SISU, Stockholm, Sweden.

Boloix, G., Sorenson, P.G. and Tremblay, J.P. (1991) On Transformations Using A Metasystem Approach To Software Development. Technical report, The University of Alberta, Edmonton, Alberta, Canada.

Bosua, R. and Brinkkemper, S. (1995) Realisation of an Integrated Software Engineering Environment through Heterogeneous CASE-Tool Integration. *Software Engineering Environments* (Ed. M.S. Verrall), IEEE Computer Science Press, pp. 152-159.

Brinkkemper, S. (1990) Formalisation of Information Systems Modelling. Ph.D. Dissertation, University of Nijmegen, Thesis Publishers, Amsterdam.

Brinkkemper, S. (1995) Method engineering: engineering of information systems development methods and tools. *Information and Software Technology*, **37**, 11, pp. 1-6.

Brodie, M. (1984) On the developments of data models. *Perspectives from Artificial Intelligence, Databases and Programming Languages* (Eds. M. Brodie, J. Mylopoulos and J. Schmidt), Springer-Verlag, pp. 19-47.

Bubenko, J.A. jr. (1988) Selecting a strategy for computer-aided software engineering (CASE). SYSLAB Report No 59, SYSLAB, University of Stockholm, Sweden.

Chen, M., Nunamaker, J.F. jr. (1989) MetaPlex: an integrated environment for organization and information systems development. *Proceedings of the 10th ICIS* (Eds. J.I. DeGross, J.C. Henderson and B.R. Konsynski), ACM Press, New York, NY, pp. 141-151.

Cronholm, S. and Goldkuhl, G. (1994) Meanings and Motives of Method Customizations in CASE Environments. *5th Workshop on Next Generation of CASE Tools*, June 6-7. Utrecht, The Netherlands.

Crozier, M., Glass, D., Hughes, J., Johnston, W. and McChesney, I. (1989) Critical analysis of tools for computer-aided software engineering. *Information and Software Technology*. **31**, 9, pp. 486-496.

Curtis, B., Kellner, M.I. and Over, J. (1992) Process modeling. *Communications of the ACM*, **35**, 9, pp. 75-90.

Essink, L. (1986) A modelling approach to information system development. *Information Systems Design Methodologies: Improving the practise* (Eds. T.W. Olle, H.G. Sol and A.A. Verrijn-Stuart), North-Holland, Amsterdam, pp. 55-86.

Euromethod (1994) Euromethod Architecture. Euromethod project deliverable Work Package 3, 1994.

Fuggetta, A. (1993) A classification of CASE Technology. *IEEE Computer*, **26**, 12, pp. 25-38.

Hahn, U., Jarke, M. and Rose, T. (1991) Teamwork Support in a Knowledge-Based Information Systems Environment. *IEEE Transactions on Software Engineering*, **17**, May, pp. 467-481.

Harmsen, F. and Brinkkemper S. (1995) Design and Implementation of a Method Base Management System for a Situational CASE Environment. *Proceedings of the 2nd Asian-Pacific Software Engineering Conference (APSEC'95)*, IEEE Computer Society Press, Los Alamitos, CA, pp. 430-438.

Harmsen, F., Brinkkemper S. and Oei H. (1994) Situational Method Engineering for Information System Projects. *Proceedings of the IFIP WG8.1 Working Conference CRIS'94* (Eds. T.W. Olle and A.A. Verrijn-Stuart), North-Holland Publishers, Amsterdam, pp. 169-194.

Harmsen, F., Lubbers I. and Wijers G. (1995) Success-driven selection of Fragments for Situational Methods - The S cube model. *Proceedings REFSQ'95 Workshop* (Eds. P. Peters and K. Pohl), Aachener Berichte zur Informatik, pp. 104-115.

Harmsen, F., and Saeki, M. (1996) Comparison of Four Method Engineering Languages. *Proceedings IFIP WG8.1/8.2 Working Conference on Principles of Method Construction and Tool Support (ME'96)*, Atlanta, Georgia, USA.

Henderson, J.C. and Cooprider, J.G. (1994) Dimensions of IS Planning and Design Aids: A Functional Model of CASE Technology. *IT and the Corporation of the 1990's: Research studies* (Eds. T. Allen and M. Scott-Morton), Oxford University Press, pp. 221-248.

Heym, M. and Österle, H. (1992) A reference model of information systems development. *The Impact of Computer Supported Technologies on Information Systems Development* (Eds. K.E. Kendall, K. Lyytinen and J.I. DeGross), Amsterdam, North-Holland, pp. 215-240.

Hoef, R. van de and Harmsen F. (1995) Quality requirements for situational methods. *Proceedings of the NGCT'95 Workshop*, Jyväskylä, Finland.

ISDOS (1981) An introdution to the System Encyclopedia Manager, ISDOS Ref #81 SEM-0338-1, ISDOS Project, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan.

Jarke, M., Pohl, K., Rolland, C. and Schmitt, J.-R. (1994) Experience-Based Method Evaluation and Improvement: A process modeling approach. *Proceedings of the IFIP WG8.1 Working Conference CRIS'94* (Eds. T.W. Olle and A.A. Verrijn-Stuart), North-Holland Publishers, Amsterdam, pp. 1-27.

Kelly S., Lyytinen, K. and Rossi, M. (1996) MetaEdit+ A Fully Configurable Multi-User and multi-Tool CASE and CAME Environment. *Proceedings of the CAiSE'96 conference*, 20-24 May, Heraklion, Crete, Greece.

Koskinen, M. (1996) Designing Multiple Process Modelling Languages for Flexible, Enactable Process Models in a MetaCASE Environment, *Proceedings of the 7th European Workshop on Next Generation CASE Tools (NGCT'96)*, Heraklion, Crete, Greece.

Kumar, K. and Welke, R.J. (1992) Methodology Engineering: A proposal for Situation-specific Methodology Engineering. *Challenges and Strategies for Research in Systems Development* (Eds. W.W. Cotterman and J.A Senn), John Wiley and Sons Ltd., pp. 257-269.

Lyytinen, K., Smolander, K. and Tahvanainen, V.-P. (1989) Modelling CASE Environments in Systems Work. *CASE'89 conference papers*, Kista, Sweden.

Marttiin, P. (1994) Towards Flexible Process Support with a CASE Shell. *Advanced Information Systems Engineering* (Eds. G. Wijers, S. Brinkkemper and T. Wasserman), LNCS#811, Springer-Verlag, pp. 14-27.

Marttiin, P., Lyytinen, K., Rossi, M., Tahvanainen, V.-P., Smolander, K. and Tolvanen, J.-P. (1995) Modeling Requirements for Future CASE: modeling issues and architectural considerations. *Information Resource Management Journal*, **8**, 1, pp. 15-25.

Marttiin, P., Rossi, M., Tahvanainen, V.-P. and Lyytinen, K. (1993) A Comparative Review of CASE Shells: a preliminary framework and research outcomes. *Information and Management*, **25**, pp. 11-31.

Merbeth, G. (1991) Maestro II - das integrierte CASE-System von Softlab. *CASE Systeme und Werkzeuge* (Ed. H. Balzert), BI Wissenschaftsverlag, pp. 319-336.

MetaEdit+ (1995) MetaEdit+: Method Workbench User's Guide (version 2.0). MetaCase Consulting, MicroWorks Finland.

Misra, S.K. (1990) Analysing CASE system characteristics: evaluative framework. *Information and Software Technology*, **32**, 6, pp. 415-422.

Nijssen, G.M. (1989) An Axiom and Architecture for Information Systems. *Information System as an In-Depth Analysis* (Ed. E.D. Falkenberg), Elsevier Science Publishers B.V. (North-Holland), IFIP, pp. 157-175.

Norman, R.J. and Chen, M. (1992) Working together to integrated CASE. *IEEE Software*, March, pp. 13-16.

Oei, J.L.H. and E.D. Falkenberg (1994) Harmonisation of Information System Modelling and Specification Techniques. *Proceedings of the IFIP WG8.1 Working Conference CRIS'94* (Eds. T.W. Olle and A.A. Verrijn-Stuart), North-Holland Publishers, Amsterdam, pp. 151-168.

Oinas-Kukkonen, H. (1995a) Linking Ability - a Model Linking Tool for MetaEdit+ Environment. Working paper series B39, Department of Information Processing Science, University of Oulu, Finland.

Oinas-Kukkonen, H. (1995b) Debate Browser - a Design Rationale Tool for MetaEdit+ Environment. Working paper series B40, Department of Information Processing Science, University of Oulu, Finland.

Olle, T.W., Hagelstein, J., MacDonald, I.G., Rolland, C., Sol, H.G., Van Assche, F.J.M. and Verrijn-Stuart, A.A (1991) *Information Systems Methodologies: A framework for understanding*. Addison-Wesley Publishing Company, Wokingham, England.

Olle, T.W., Sol, H.G. and Tully, C.J. (Eds.) (1983) *Information Systems Design Methodologies: A Feature Analysis*. Elsevier Science Publishers, North-Holland, Amsterdam.

Olle, T.W., Sol, H.G. and Verrjin-Stuart, A.A. (Eds.) (1982) *Information Systems Design Methodologies: A comparative review*. Elsevier Science Publishers, North-Holland, Amsterdam.

Olle, T.W., Sol, H.G. and Verrijn-Stuart, A.A. (Eds.) (1986) *Information Systems Design Methodologies: Improving the practise*. Elsevier Science Publishers, North-Holland, Amsterdam.

Rossi, M., Gustafsson, M., Smolander, K., Johansson, L.-Å. and Lyytinen, K. (1992) Metamodeling Editor as a Front End Tool for a CASE Shell. *Advanced Information Systems Engineering* (Ed. P. Loucopoulos), LNCS#593, Springer-Verlag, Berlin, Germany, pp. 546-567.

Saeki, M., Iguchi, K., Wen-yin, K. and Shinohara, M. (1993) A meta-model for representing software specification & design methods. *Proceedings of the IFIP WG8.1 Conference on Information Systems Development Process* (Eds. N. Prakash, C. Rolland and P. Pernici), Como, pp. 149-166.

Slooten, K. van, and Brinkkemper S. (1993) A Method Engineering Approach to Information Systems Development. *Proceedings of the IFIP WG8.1 Conference on Information Systems Development Process* (Eds. N. Prakash, C. Rolland and P. Pernici), Como, pp. 167-186.

Smolander, K., Lyytinen, K., Tahvanainen, V.-P. and Marttiin P. (1991) MetaEdit - A flexible graphical environment for methodology modelling. *Advanced Information Systems Engineering,* (Eds. R. Andersen, J. Bubenko and A. Sølvberg), LNCS #498, Springer-Verlag, pp. 168-193.

Smolander, K. (1992) OPRR - A Model for Methodology Modeling. *Next Generation of CASE Tools* (Eds. K. Lyytinen and V.-P. Tahvanainen), Studies in Computer and Communication Systems, IOS press, pp. 224-239.

Sorenson, P.G., Tremblay, J-P. and McAllister, A.J. (1988) The Metaview system for many specification environments. *IEEE Software*, **30**, 3, March, pp. 30-38.

Ter Hofstede, A. H. M. and Weide, Th. P. van der (1993) Expressiveness in data modeling. *Data & Knowledge Engineering*, **10**, pp. 65-100.

Venable, J. (1993) CoCoA: A Conceptual Data Modelling Approach for Complex Problem Domains. Ph.D. dissertation, State University of New York, Binghampton.

Verhoef, T.F. and Ter Hofstede, A.H.M. (1995) Feasibility of Flexible Information Modelling Support. *Advanced Information Systems Engineering* (Eds. J. Iivari, K. Lyytinen and M. Rossi), LNCS #932, Springer-Verlag, pp. 168-185.

Vessey, I., Jarvenpaa, S. and Tractinsky, N., Evaluation of Vendor Product: CASE Tools as Methodology Companions. *Communications of the ACM*, **35**, 4, pp. 90-105.

Wijers, G. and Dort, H. van (1990) Experiences with the use of CASE tools in the Netherlands. *Advanced Information Systems Engineering* (Eds. B. Steinholz, A. Sølvberg and L. Bergman), LNCS#436, Springer-Verlag, pp. 5-20.

Wijers, G. (1991) Modelling Support in Information Systems Development. Ph.D. dissertation, Thesis publishers, Amsterdam.

Wynekoop, J.D. and Russo, N.L. (1993) System development methodologies: unanswered questions and the research-practice gap. *Proceedings of 14th ICIS* (Eds. J.I DeGross, R.P Bostrom and D. Robey), Orlando, USA, pp. 181-190.

## 9 BIOGRAPHY

Pentti Marttiin is a researcher in the MetaPHOR project funded by the Academy of Finland. He received his M.Sc. (1991) and Econ.Lic. (1994) at the Department of Computer Science and Information Systems, University of Jyväskylä, Finland. He has written articles on method engineering and metaCASE environments published in Information and Management, Information Resource Management Journal, and several conferences. He has participated ICIS'92 doctoral consortium, served as a program committee member in CAiSE'95, and involved in the development of metaCASE tools for Meta Case Conculting. His current research interest focuses on process and agent modelling aspects in metaCASE.

Frank Harmsen is a researcher in the Information Systems Design Methodology Research Group at the Computer Science Department of the University of Twente in the Netherlands. He holds a B.Sc and M.Sc in Mathematics and Computer Science from the University of Nijmegen. His research interests are information system methodology, meta-modelling, method engineering, and CASE tools, about which he has published several papers. Current research activities focus on defining formalisms and tools for representation and assembly of method fragments for Situational Method Engineering. He was co-editor of the 1993 edition of the Workshop on Next Generation of CASE Tools (NGCT), and served on the organisation committee of CAiSE'94 (Conference on Advanced Information Systems Engineering). He is a member of the Netherlands Society for Informatics.

Matti Rossi is a researcher in the MetaPHOR project funded by the Academy of Finland. He received his M.Sc. (1994) and Econ.Lic. (1996) at the Department of Computer Science and Information Systems, University of Jyväskylä, Finland. He has published in Information and Management, Information Systems, and Information Resource Management Journal, and participated CAiSE, ECOOP, WITS and SERF conferences. He has served on the Workshop, Poster and Exhibition Chair of the CAISE'95 conference. He is also a member of the board in Meta Case Conculting, and has involved in implementing the report generation facility of MetaEdit and CAME tools of MetaEdit+. His research interests include database management, object-oriented data representation, metamodelling, transformations in metamodelling, and the applications of the previous items to software engineering.