Scandinavian Journal of Information Systems

Volume 5 | Issue 1 Article 5

1993

# FLEXIBLE METHOD ADAPTATION IN CASE: The Metamodeling Approach

Juha-Pekka Tolvanen
*University of Jyväskyä, Finland*, jpt@jyu.fi

Kalle Lyytinen
*University of Jyväskylä, Finland*, KalleLyytinen@emailaddressnotknown

Scandinavian Journal of Information Systems, Vol. 5, pp. 51–77, 1993

# FLEXIBLE METHOD ADAPTATION IN CASE
## The Metamodeling Approach

JUHA-PEKKA TOLVANEN & KALLE LYYTINEN
Department of Computer Science and Information Systems
University of Jyväskylä
P.O. Box. 35, SF-40351 Jyväskylä, Finland
jpt@jyu.fi

**Abstract**

There is an obvious need to improve the adaptation of methods into Computer Aided Software/Systems Engineering (CASE) tools. This paper presents a new approach to adapting methods into CASE environments, called metamodeling. It applies a new generation CASE tool or CASE shell to offer flexible mechanisms to specify and implement methods, or to modify existing ones in tools. This allows customization of CASE support for local needs. Metamodeling is a key step in such a customization and adaptation task. During metamodeling a formal model of the method to be supported is derived. This paper offers guidelines for a method adaptation process based on metamodeling. The goal of the process is to examine and improve methods so as to adapt them flexibly and successfully into the contingent local needs in order to achieve a sufficient fit between users' cognitive skills, special methods, and tool support. The practicality of the method adaptation guidelines is demonstrated by reporting a case study where a 'manual' method called Activity Modelling was adapted into a CASE shell called MetaEdit. We also suggest criteria to evaluate the adaptation outcome and illuminate them in our adaptation case. The paper ends up with a speculation of how the nature of IS development methods is likely to change due to increased computer support. This will make the dominating 'paper-and-pencil' mentality obsolete, and introduce more flexible, complex and versatile methods which are supported by powerful analytical tools offering unprecented functionality such as simulation or hypertext features. We believe that the metamodeling approach will form an essential core of method development and use in years to come, as it can be used to extend and modify organizations' knowledge about methods and to make them learn more rapidly.

# 1 Introduction

In the area of Computer Aided Software/Systems Engineering (CASE) we face a continuous need to integrate tools and methods better. On the one hand tools support and mechanize operations prescribed by methods by storing system representations, transforming representations from one form to another, and displaying representations in varying forms. By a method we mean a set of steps and a set of rules that define how a representation of an information system is derived and handled (Smolander *et al.* 1990a).[1] On the other hand tools empower users by enhancing correctness checking and analytical power, by freeing them from tedious documentation tasks, and by providing multi-user coordination (access and version control). None of these features could easily be available in manual method use.

A major focus in developing CASE has so far been on integrating existing methods into tools. We call this a tool-driven CASE approach, and it can be seen to build 'method-enhanced tools'. As a result we currently have available a large number of CASE products which strive to mechanize manual methods mostly developed in the 70s. These tools offer simple technical support which implements the 'pen and paper' mentality of these methods. One major consequence of this approach has been high method-dependency in available CASE tools which makes method customization difficult, if not impossible. As empirical studies reveal this creates difficulties in introducing CASE, because the selected CASE tool changes methods already deployed in the organization as well as current working practices (Aaen *et al.* 1992, Loh & Nelson 1989, Smolander *et al.* 1990b). Hence, limited possibilities to customize a tool often lead to users' dissatisfaction and resistance (Wijers & van Dort 1990).

In this paper we explore another direction for integrating methods and tools. We shall call this approach method-driven CASE, as it works through CASE-enhanced methods. We argue that this view has not been fully examined in CASE research so far, though it offers several advantages which warrant research effort. In this approach customizable CASE tools, also called as metasystems (Sorenson *et al.* 1988) or CASE shells (Bubenko 1988) establish a key technological component. The approach aims to integrate CASE and methods in a more flexible way, by providing facilities to tailor tools for selected methods with some desirable properties.[2] Using such a capability CASE shells can be 'programmed' and thereby adapted to local methods. By method adaptation we mean a representation of a given method for a CASE tool in such a way that the CASE tool can support development tasks as prescribed by the method. More interestingly, CASE shells in the method-driven CASE are seen to offer novel ways to create new methods and enhance them in previously unimaginable ways. It seems that the capability to customize tools gives more degrees of freedom in method selection, and thereby provides a balanced way to integrate methods and tools which can overcome some of the current problems in CASE adaptation.

In order to customize methods with tools we have to understand how method

adaptation can be done in practice and what factors affect its success. Although there has been a growing interest in examining technical capabilities of CASE shells, there is a paucity of research that deals specifically with problems of method adaptation. Few discussions available, see e.g. (Bubenko 1988, Tagg 1990) focus mainly on alternative methodology engineering approaches. What these studies lack, however, are tried out guidelines and 'methods' in adapting methods to CASE. Therefore, in this study our essential question will be: how to integrate methods and CASE tools, given a set of constrains, so that the method meets selected use criteria? To address this question we develop metamodeling approach for method adaptation and evaluate its usefulness. In the metamodeling approach we apply data modeling theory to derive a formal model of a method (a metamodel), which helps us to make informed decisions about issues faced during method adaptation. We examine what method knowledge can be captured into metamodels to guide the adaptation process. To demonstrate the practical value of the proposed adaptation approach we introduce a case study of a specific method adaptation. In this case we apply a CASE shell (MetaEdit) and adapt a manual method called Activity Modeling (Verksamhetsanalys med handlingsgrafer) (Goldkuhl 1990, 1992) to it. As a result we obtain a functional tool for activity analysis. In this process we identify several decision alternatives which could be followed to implement the method into a tool. We also take some of them into a closer examination and point out how specific criteria can be used to make informal decisions about method adaptation. Based on this we shall suggest a set of criteria to make informed choices between the available alternatives. Suggested evaluation criteria are general and not dependent of specific circumstances, such as the tool applied.

The paper is organized as follows. Section 2 introduces the metamodeling approach and delineates stages of the adaptation process. We also outline the evaluation framework for estimating results of method adaptation. In Section 3 we explore our method adaptation case and use the evaluation framework to estimate its success. In Section 4 we probe what possibilities the metamodeling approach and CASE shells can provide in the future, and discuss how the nature of methods is likely to change due to increased computer support.

# 2 Metamodeling Approach to Method Adaptation

## 2.1 Modeling of Method Knowledge

Two types of knowledge are essential in IS development: knowledge of the application system and knowledge of the system design. CASE tools must represent and utilize both types of knowledge. On the one hand CASE tools assist modeling of the application: its current or desired state. With these models IS developers can simplify the design problem and thereby capture its essential features. On the other hand tools embed knowledge about methods and their use. This knowledge is captured in CASE tools in some sort of a model of the method,

i.e. a metamodel, see also (Kottemann & Konsynski 1984). In its simplest form we can say that a metamodel is a conceptual model of a development method (Brinkkemper 1990).[3] Consequently, metamodeling can be defined as a modeling process, which takes place one level of abstraction and logic higher than the standard modeling process (van Gigch 1991). Clearly, no modeling is possible without some sort of (explicit or implicit) metamodeling. The metamodel captures information about the concepts, representation forms, and use of a method. IS developers use this kind of knowledge—although often unconsciously—in IS modeling tasks, see (Smolander *et al.* 1990a). For example, in data-flow diagrams the concepts used to model the systems are processes, data stores, external entities and various relationships between them. Moreover, the metamodel of data flow analysis defines how each object (concept) is represented (e.g. the symbol definition for an external entity is a square), and in what order the system model should be created (e.g. top-down structure). The relationships between modeling and metamodeling are illustrated in Figure 1.

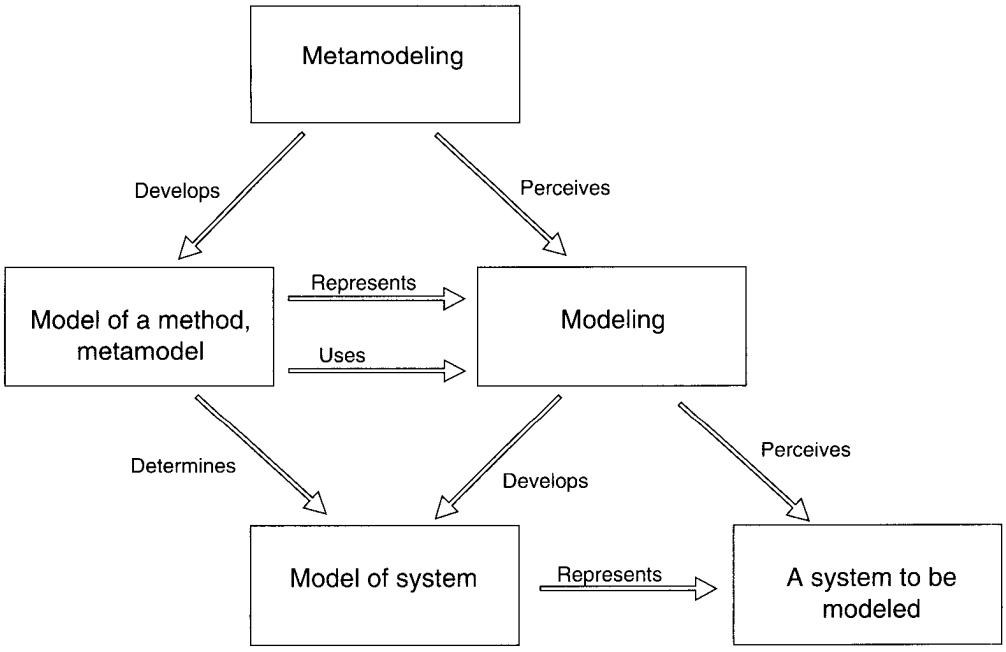


Figure 1: *Metamodeling and modeling (Brinkkemper 1990)*

In CASE the need to render model knowledge explicit and formalize it grows. Metamodels in CASE form an 'in-coded' part of CASE repositories, i.e. they act as a schema for the design database. Currently most widely-used metamodels are higher level data models, like ERA (CASE 1989). Moreover, some parts of the code that manage the design objects and their behavior in the repository need to

be included into the metamodel. CASE shells use higher level semantic modeling constructs, and thus provide mechanisms to implement such metamodels quickly. An additional advantage of using such metamodels is that we can apply them to help make informed decisions about the methods' use. In the other words, using metamodels we can examine in a systematic and rigorous fashion how IS developers perceive the IS, in what notation the system is described, and how the modeling process will proceed.

Though Figure 1 describes all modeling levels, it does not reveal the dynamics of the metamodeling process. Metamodeling also uses its own methods and tools which, in turn, can be described on one level higher in metametamodels (and so ad infinitum). In general, the study of principles, tools and methods for metamodeling and their management is called methodology engineering. Finally, a person responsible for developing method specifications and implementing them is called a methodology engineer. The art of methodology engineering is still in its infancy, though some recent research efforts have recognized the need to systematically develop metamodeling tools and compare methods using 'sound' metamodeling principles, see (Welke & Forte 1989, Heym & Österle 1992, Tolvanen *et al.* 1993).

The model of a method, i.e. a metamodel, can be further divided into a meta-datamodel and a meta-activitymodel (Brinkkemper 1990). A meta-datamodel describes the static aspects of a method, like its concepts and terms, and a notation or a representation of the method (i.e. a graphical description language). Widely applied meta-datamodeling methods include the ER-model (Chen 1976), NIAM (Verheijen & Bekkum 1982), and OPRR (Welke & Forte 1989). Meta-activitymodels describe dynamic aspects of a method's use, like its stages, tasks or steps. Available (meta)methods for meta-activitymodeling include e.g. data-flow diagrams, and Petri-nets (Curtis *et al.* 1992).

Overall, we can distinguish three dimensions in modeling methods for CASE environments. The first dimension deals with the two abstraction levels needed to build up a model hierarchy: the metamodel and its model instantiations. This we call a type-instance dimension in metamodeling (Smolander *et al.* 1990a). Types included in a metamodel determine what one can describe or observe on the instance level while using the method. The second dimension makes a difference between the conceptual structure of a method and its representational form. This is called the conceptual-representational dimension (Smolander *et al.* 1990a), which has been recognized since the ANSI/SPARC proposal for a three-level data base architecture (ANSI 1978). This helps to achieve in CASE what we call the "representation independency", i.e. that the method knowledge can be conveyed in a varying representations covering graphical, matrix or textual form. The third dimension recognizes time as an important feature of metamodeling and thereby introduces dynamics into the modeling domain. This is called the static-dynamic dimension of metamodeling and it permits us to distinguish between meta-datamodels and meta-activitymodels, where only the latter involve time. Hence, meta-datamodels declare what a method allows one to perceive, while

meta-activitymodels specify how a 'method' should be followed, i.e. how and in what order of tasks (in time) a method produces its products (representations). The three dimensions are further clarified in Figure 2 (note that not all regions in the cube are illustrated).

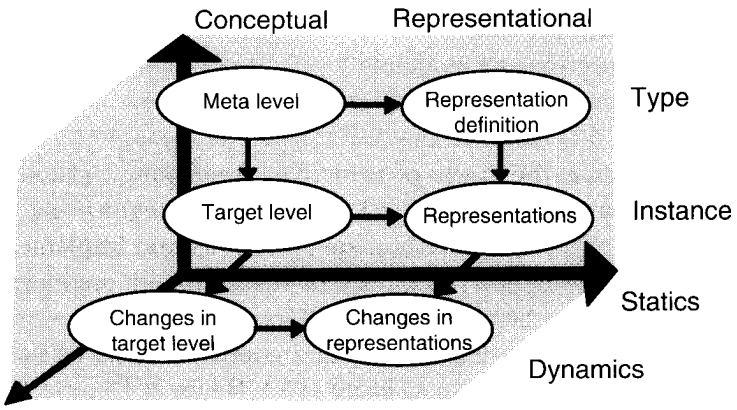


Figure 2: *Three dimensions of metamodeling*

To summarize the three dimensions allows us to discern three key concerns in method engineering. These are: (1) what knowledge is conveyed, stored and manipulated in the CASE tool while using the method, (2) how the knowledge is represented for different stakeholders of the IS development, and (3) how the method knowledge is produced and consumed in the development process. Moreover, the dynamics allows us to analyze the dynamics of the method evolution and its modifications in conceptual/representational dimensions.

## 2.2 The Process of Method Adaptation

Although several studies, see (Bergsten *et al.* 1989, Brinkkemper *et al.* 1990, Bubenko 1988, Declerfayt & Milgrom 1989, Lustig & Ersson 1989, Patel 1989, Shapiro 1991) have shown that there are real benefits to be gained from CASE tool customization, little attention has been paid to the content and structure of the customization process. Though, some studies, see (Brinkemper *et al.* 1989, Pelly 1990, Teichroew *et al.* 1980) discuss method adaptation at some length, they do this in the context of particular tools. Furthermore, some studies in method-tool integration aim to integrate several CASE tools by specifying the data content of the methods to be supported through a shared repository (A/D Cycle). Clearly, this allows only a limited customization possibility because the meta-datamodel is fixed, i.e. customization mainly deals with representations and static object types. Another problem is the high complexity of the technical environment and the metamodels used because the metamodel must cater for all

possible users and environments (Mercurio *et al.* 1990). In this paper we shall study the method-tool integration from another angle which will increase the flexibility by using meta-datamodels. Unfortunately, our current understanding of how to adapt methods successfully by using metamodels is inadequate. In particular, we do not know how to adapt methods for different development contingencies. This section aims to develop some guidelines for method adaptation.

We divide a method adaptation process into four steps: *a selection of methods, modeling of the methods, modeling method use and method linkages, and evaluation of the metamodeling results.* Each step is discussed in more detail below.

### Selection of methods

This step is important, because without a proper selection of methods and understanding of the needs they seek to satisfy the adaptation process is difficult to carry out. Because methods form a widely studied area with a rich and varying literature, see e.g. (Olle *et al.* 1982, 1983, 1986, 1988), we will not examine this task in great detail. Instead, we shall consider three issues that need to be addressed in forming a methodology. In our opinion these questions reveal critical knowledge of IS development as it is currently practised in the organization. These questions are:

1. What tasks must be supported by IS computer-aided tools? Often it is unnecessary to cover the whole system life-cycle with computer supported methods. Therefore, one must focus on those critical tasks that need to be supported by CASE tools and which can improve process quality and its effectiveness. Selecting specific computer-aided methods will govern what 'knowledge' must be captured, derived and handled in the CASE environment.

2. What is the type of the system being developed? Methods must be helpful for the type of IS being developed. For example, in developing production planning systems, we can be interested in the dynamic system behavior represented in production plans and schedules. In developing interactive data intensive systems we are often interested in the static structure of the universe of discourse as represented in entities and their relationships.

3. What is the developers' familiarity with the method? The organizations' own method base and standards must guide the method selection. It is obvious that potential users will approve a new tool more easily if it supports widely-used methods and working practises.

### Modeling method structure

This step specifies the concepts involved in a method and their representation from a static perspective (cf. Figure 2, dimensions of metamodeling). During

this phase a method engineer models a method formally using some metamodeling language. The result specifies which concepts, relationships and properties the method leads one to perceive in the system under development, and what representation form it suggests to represent the system. This phase is essential in the method adaptation because the resulting metamodel will largely dictate how easy the method is use in the tool. This step will determine e.g. when and how the system will check the model consistency. The method modeling step is discussed in more detail in Section 3.

### Modeling method use and method linkages

This step specifies the dynamic behavior of IS model instances (see Figure 2). Hence, in this step a method engineer models how the method will guide and restrict developers' activities and tasks (Brinkkemper 1990). Moreover, the method engineer defines connections with other methods as to form a larger methodology. These connections define method interactions in two ways: in the horizontal and the vertical dimension (Lyytinen *et al.* 1991). In the horizontal dimension connections or constraints between objects in different model representations are specified. For example, data stores in data-flow diagrams are redefined for cross-checking with an ERA-model. The vertical dimension denotes linking of 'semantically' equivalent descriptions at two consecutive levels of abstraction, such as connecting an ER-model with its representation in a relational schema. Because vertical system descriptions must be maintained for consistency, a methodology engineer's attention concentrates here on maintaining consistency over several representations.

### Evaluation of the metamodeling results

This step is needed to analyze and assess customized methods. In this step we shall use the framework proposed by Brinkkemper[4] (1990) and assess the results using twelve partly overlapping criteria. These criteria will help to evaluate how well the 'knowledge' within the method is captured in the metamodel. We do not extensively address issues related to the CASE tool's functionality here, because we do not develop issues related to meta-activitymodeling in this paper.

The criteria suggested here are general and they focus on diverse dimensions to evaluate the adaptation outcomes. Due to the multiple perspectives applied some criteria are contradictory. For example, some method adaptations need to support professional developers' design tasks thus emphasizing completeness and consistency, whereas some other situations need to focus on preserving users' natural concepts which may not be always consistent, but more complete. The criteria suggested are presented in the form of a series of questions. To demonstrate their usefulness we relate each criterion to the three-dimensional metamodeling framework suggested above:

1. Are all relevant aspects of the universe of discourse described completely?

This requirement sets out the goal that the method must capture essential 'objects' in the design problem and convey relevant information about them. This is the most important requirement in identifying metalevel types and their representations.

2. Are system descriptions consistent? Systems descriptions are often large and complex. Hence, system descriptions must be checked, for their consistency using the available metamodel data. This will result in well-defined and complete model instances.

3. Are system descriptions valid? The metamodel must help to validate the system descriptions in relation to IS users' or designers' desires and needs. This requirement is partly overlapping with the consistency criterion. There is, however, a marked difference: validity deals mostly with the semantic adequacy, whereas consistency focuses mainly on the syntactic properties of the models. Therefore, validity can not be assessed by exploring the metamodel alone.

4. Are concepts well-defined? This requirement is satisfied only partly in most development methods, because they are mostly defined informally, thus providing high degrees of freedom in interpreting the modeling results. In order to use the method fully in a CASE tool its structure and behavior must be well-defined, i.e. in adaptation the need to formalize a method is increased. This relates mainly to the semantics of the type system in the metamodel.

5. Is method use deterministic? This requirement is difficult to meet in the metamodel, because formalization of all rules related to method use is difficult, if not impossible. This relates to the dynamics of the target systems descriptions.

6. Does the method correspond with users' natural concepts? Development methods are developed to satisfy developers' cognitive needs related to design tasks. Therefore, it would be an advantage if methods were similar to users' native concepts and thinking patterns, as this requires less learning. This question focuses mainly on the type system and the representation definitions in the metamodeling framework.

7. Are all developer's information requirements satisfied? This requirement closely relates to the first question. All relevant information of the IS as needed by developers must be captured so as to satisfy developers' and users' needs. Therefore, the adaptation process must take into account developers' experience, skills and view of the development process and situations. In other words, the adaptation should strive to describe concepts related to the different views and on different abstraction levels. These

should cover programming and data base management concepts, as well as concepts closer to end users' experience.

8. What aspects of the method are formalized? Formalization is often highly useful—although it is impossible to accomplish fully—because it reduces inconsistent uses of methods by forcing the method to be used in a more predictable way. Formalization improve consistency checking and use of more analytical methods to analyze descriptions. Finally, formalization makes it possible, for example, to suggest more rigid versions of methods for novices, and thereby to guide them to derive system descriptions 'correctly'. In this sense formalization deals with all aspects of the metamodeling framework.

9. Are IS descriptions useful in communication? This aspect should be also considered during the adaptation as metamodels can help to examine the semantics and the representation forms of the methods.

10. Does the method use help to reduce the complexity of the system development? The method should contain rules which ease modeling of various design situations by helping to simplify them. In other words, the adaptation process should introduce rules such as functional decomposition principles to reduce the design complexity. This issue relates mostly to the dynamic dimension in the metamodeling framework.

11. Can method use be divided into steps? The method adaptation process should divide the method use into smaller tasks. This simplifies the management of the dynamic dimension and often improves the quality of descriptions.

12. Do the methods work together as to form an integrated methodology? This aspect emphasizes conceptual method connections (both vertical and dynamic integration) which must be maintained and checked for correctness in the CASE tool. This is only possible if the metamodel specifies and maintains such knowledge. The resulting integrated methodology can also guide CASE tool users more effectively by offering transformation support and reuse of system descriptions. This criterion in the adaptation process concerns all three dimensions, albeit with the main focus here on the type dimension.

An outcome of a successful method adaptation process will specify what developers' can store in a CASE tool's repository, how system descriptions can be represented, retrieved, verified and validated, transformed, and how descriptions are managed. Furthermore, using the metamodel we can anticipate how descriptions will be used in communications and how design decisions are reached (through a meta-activitymodel). The adaptation process will help to make more informed choices between different system development strategies and representation forms by considering alternative solutions in each of the three dimensions.

# 3 A Case Study on Adaptation to CASE: Implementing Activity modeling in a CASE shell

In this section we describe how one method was adapted using a metamodeling approach into a CASE tool. The goal of the exercise was to develop tool support for a user-oriented method without any tool support. At the same time opportunities for extending the method were sought to benefit from the increased analytical power of the computer support. In other words we aimed to develop a CASE-enhanced method. In our case we shall concentrate mainly on analyzing the static parts of a method, i.e. on the 'data' gathered, represented and handled by a particular method (i.e. the type-instance and the conceptual-representation dimensions). The modeling of method use and method interaction, i.e. the dynamic part of the adaptation, is excluded from our considerations, see e.g. (Tolvanen *et al.* 1993) because the technological platform used did not provide any support for this. Moreover, the dynamic area of metamodeling is less developed (Curtis *et al.* 1992). In particular, we shall examine some method support alternatives identified during the adaptation process and discuss reasons for choosing specific solutions for the method specifications. Factors affecting these choices include the users' experience and the content of the IS development domain. Finally, we assess the results of the adaptation process using the twelve questions of the framework presented above.

## 3.1 Presentation of Adaptable Development Method

In order to find a useful case study for a metamodeling exercise we picked up an unknown method which had currently only a manual version: contextual activity modeling with action diagramming[5] (Kontextuell verksamhetsanalys med handlingsgrafer) (Goldkuhl 1990, 1992). Activity Modeling is a process-oriented method which uses graphical diagrams to analyze information system's role in an organizational context. Further reasons for selecting Activity Modeling were its rich vocabulary, quite complex graphical notation, availability of method developer's comments and thereby easy validation of the metamodeling results, and its user-oriented character with high participatory ambitions which stresses simple yet flexible task support. Accordingly, the method use can alter from one situation to another. Use of Activity Modeling does not depend on any sophisticated or technical principles, like functional decomposition. Thus the method serves well in communicating between developers about the role of IS in the organizational context.

The basic concepts of the method are: an action, an information set, and a material set. Their representations are depicted in Figure 3. Instances of these objects are connected to other objects by information or material flows. Furthermore, activity graphs include logical operators representing alternatives or choices between actions. It can also contain texts to provide more detailed information about flows, actions, or triggers. It is possible to use an 'order flow'

to describe an order between actions which are not connected through information or material flows. In the manual version of the method the size of one activity graph is limited to the size of a paper sheet. Therefore, graphs in different sheets are connected through symbolic references. This feature, however, makes the use of the activity graphs—like many other 'pen and paper' -oriented methods - labour-intensive and tedious.

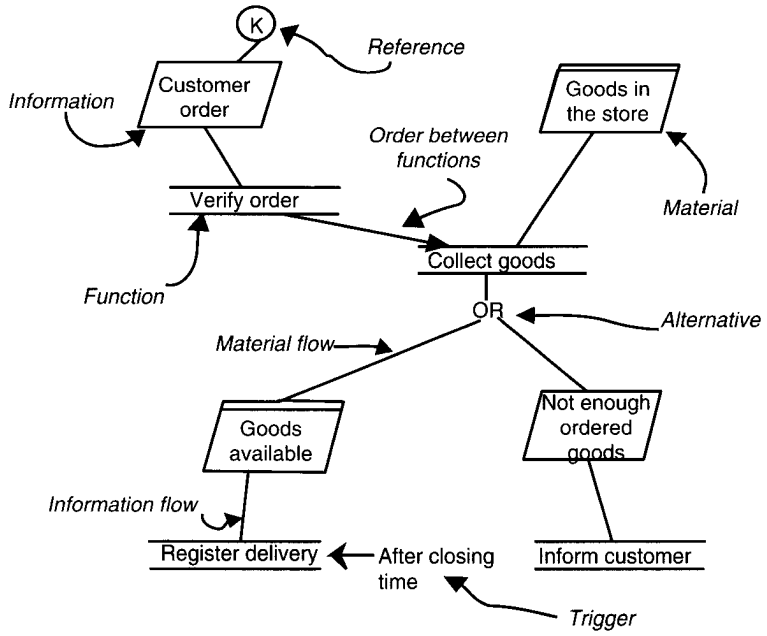


Figure 3: *Basic concepts of Activity Modeling*

In principle, it is possible to define several attributes for all objects in an Activity graph. Goldkuhl's (1990) original version of the method assumes, however, that all objects are identified only by their name (attribute). Only actions are described with additional attributes including their place and actor.

Important features in the graph, such as the role of activities which are supported by ISs, are represented in text. The texts thus solicit knowledge of a particular IS, including its actions, inputs and outputs. These descriptions are used in later phases to design the system functionality in more detail.

## 3.2 Introduction to the Metamodeling Method

To integrate the method and the tool smoothly presumes that a metamodeling method is directly supported by a tool. In this sense the tool and metamodeling language dictates the semantic primitives used. In our case we used a tool called MetaEdit (Smolander *et al.* 1990a) which necessitated at the meta level the use

of the OPRR[6] data model (Smolander 1991). This model forms the specification language for method representations in MetaEdit. A unique and useful feature of MetaEdit is that a graphical OPRR representation can be build up and compiled in this environment. Therefore the implementation of the CASE tool is easy and straightforward after the graphical representation of the method in OPRR has been achieved.

The basic primitives of the OPRR model are:

- An object is a 'thing' which exists on its own. It is represented by its associated properties.
- A property is a describing/qualifying characteristic associated with other object types (object, relationship, or role).
- A role is a link between an object and a relationship. A role may have properties that clarify the way in which 'things' participate in a certain part of a relationship. The role defines what 'part' an object plays in a relationship.
- A relationship is an association between two or more objects. It cannot exist without its associated objects. Relationships can also have properties.

Formally, a metamodel can be defined in OPRR as a seven-tuple, see (Smolander 1991).

$$\langle O, P, R, X, rv, r, p \rangle$$

$O$ is a finite set of object types
$P$ is a finite set of property types
$R$ is a finite set of role types
$X$ is a finite set of relationship types
$rv$ is a mapping:$R \mapsto \Omega$, where $\Omega = Powerset(O), and$
$r$ is a mapping
$r : \mathcal{X} \mapsto \mathcal{X}_{i=1}^{n}(R_i \times A_i)$, where $A_i = rv(R_i)$, where $R_i \in R, n \geq 2$
$p$ is a mapping
$p : NP \mapsto Powerset(P)$,
where $NP = O \cup R \cup \mathcal{X}$ is the set of non-property types

The function $r$ defines the set of object types $A_i$ that can be in role type $R_i$. This set $(A_i)$ is returned by a function $rv$ which is defined from the role types onto subsets of object types. We naturally assume that all sets are non-empty. Because $n \geq 2$, the number of roles in a relationship is always at least two. The mapping $p$ defines the set of property types which are associated with each non-property type (roles, objects, relationships).

### 3.3 Detailed Method Modeling

Metamodeling is an iterative process in which alternative ways of using methods are collectively tried out, analyzed and debated using metamodeling results. We distinguish the following eight metamodeling steps:

1. Identify object types.
2. Define properties for each object type.
3. Identify relationships between object types.
4. Define roles and connections in identified relationships.
5. Inspect properties allocated to object types, relationship types and role types, and define the identifying properties for object types and relationship types.
6. Define the complete conceptual model.
7. Analyze the conceptual model.
8. Define the representational part of the method.

*Identify object types.* Using these steps we first identify object types in the Activity Modeling, which are described in Figure 3. These are: an *action*, a *material set*, an *information set* and an *operation*. The operation object type is used to illustrate alternative flows.

*Define properties.* According to the available version of the Activity Modeling (Goldkuhl 1990) an *information set*, a *material set* and an *operation* have only one property: their description, which conveys the meaning of each object (information set and material set descriptions, and a mark of the operator). If CASE tool users would like to perceive and collect more data about *information sets* or /em material sets, some additional properties, such as information about volume or use, could now be added into these object types. As noted above, in the original version of Activity Modeling, only the *action* type is described with some additional properties.

Due to the need to integrate and manage several graphical system descriptions in a computer environment, the metamodel must also include a mechanism to *reference objects*, i.e. a reference that acts as an identifying property between objects in other models. In CASE tools references between graphs can be defined using identifying properties of the referred object type: i.e. an *action*, a *material set* or an *information set*. Using system descriptions to communicate with other environments (e.g. to allow the use of paper documents) requires that references can be easily identified and followed. Hence, references are needed to maintain an order between incomplete graphs using an object type *reference* with the property identifier of an other graph. Moreover, the need for defining system descriptions contextually with a possibility to attach free form text to descriptions demands a

new object type, a *specifier* with a property type *description.* It could, however, be modeled, like the reference type, as a property of another object or relationship type instead of an object type.

*Identify relationship types.* In Activity Modeling, we distinguish four relationship types (see Figure 3) which are the following: *information flow, material flow, order link*and*reference link.* A metamodel for Activity Modeling can be presented in the form of a set of object types, a set of property types and a set of relationship types as follows:

| | | |
|---|---|---|
| O (object types) | = | {Action, Information set, Material set, Operator, Reference, Specifier} |
| P (properties) | = | {Name of the action, Agent, Place, Trigger, Information description, Material description, Mark, ID of graph, Description of specifier} |
| X (relationships) | = | {Information flow, Material flow, Order, Refers} |

Although the identification of four relationship types is a straightforward process, their use to model connections between object types is not necessarily simple. Due to the current ambiguous and informal specifications of the method there are several alternatives to specify this part of the metamodel. In the description of the method (Goldkuhl 1990) it is not described, for example, whether it is possible to add information to the material set, or how freely the method user can create relationships between operators, information sets, or material sets. Therefore, metamodeling involves increasing the formality of the Activity Modeling method.

*Define the connections and role types.* In the OPRR model a specification of relationships between object types defines role types and their mappings onto object types. According to the OPPR model the relationship types must always be specified with two role types. Therefore, these must be defined, including the connectivity of each relationship type. The role types are used to model direction of relationships, their semantics and representations of connections between object types and relationship types. The connectivity of a role type can be one of the following: zero to one (0:1), one to one (1:1), zero to many (0:M), one to many (1:M) or many to many (M:N). Zero in the connectivity means that instances of an object type do not have to be connected to instances in another object type; and one to one that an instance of an object type must be connected at most to one instance of another object type. Hence, participation in only one relationship in a selected role is allowed. In the Activity Modeling all connectivities are type: zero to many, and can be defined as follows:

| | | |
|---|---|---|
| R (roles) | = | {Information flow from, Information flow to, Material flow from, Material flow to, Order from, Order to, Reference from, Refers to} |

*Analyze properties.* Finally, we inspect properties aligned to object, relationship and role types. We need to define properties to identify object types at an

instance level. Although it would be possible to define some properties, such as triggers, specifiers and references, in alternative ways, i.e. either as a property of some object type, or as an independent object type, or in both ways, we have selected one solution in this case. For example, the motivation to model a trigger as a property of an action type is that only actions have triggers. The question about specifiers and references is a more difficult one. Therefore, advice should be sought from the method or CASE users. In our present case both specifiers and references were modeled as object types to allow a better usage of system descriptions in user-developer communication. In this case the identification of object instances is easier. Overall, most properties act as identifiers of object types (information set, material set, reference, operator and specifier). In the case of object type action, we use the name of the action as the identifying property.

Define the complete conceptual model. We can now define the mapping from the relationship types (r) as following:

r={⟨Information flow, ⟨⟨Information flow from, {Action, Information set, Material set, Operator ⟩, ⟨Information flow to, {Action, Information set, Operator}⟩⟩⟩,
⟨Material flow, ⟨⟨Material flow from, {Action, Material set, Operator}⟩, ⟨Material flow to, {Action, Material set, Operator}⟩⟩⟩,
⟨Order, ⟨⟨Order from, {Action}⟩, ⟨Order to, {Action}⟩⟩⟩,
⟨Refers, ⟨⟨Reference from, {Reference }⟩, ⟨Refers to, {Action, Information set, Material set}⟩⟩⟩⟩}

All definitions presented will be needed in the implementation of the OPRR-based CASE environment. Because the metamodeling environment applies a graphical modeling tool, Figure 4 illustrates a graphical OPRR model of the Activity Modeling as it would be presented in MetaEdit. The figure follows the OPRR notation used in MetaEdit: an object type is represented by a rectangle, a relationship type by a diamond, a role type by a circle, and a property by an oval.

*Analyze the conceptual model.* So far, we have interpreted Activity Modeling quite freely. For example, we have no connectivity limits with the relationship types. Moreover, many to many relationships are allowed in material and information flows between the object types. Due to the informal definitions of the Activity Modeling, it is also possible to model the methods more rigorously based on developers' experience or demand. For example, in the description of the Activity Modeling (Goldkuhl 1990) the relationships between *material sets* and *information sets* are preferred to be limited to one to reduce the complexity of the system descriptions. Thus, for novice users the method can be defined in a more restrictive fashion by adding specific relationship types between object types. If all relationships between object types, excluding the *operator* are limited to one instance, the OPRR definitions of the metamodel will include in total 11

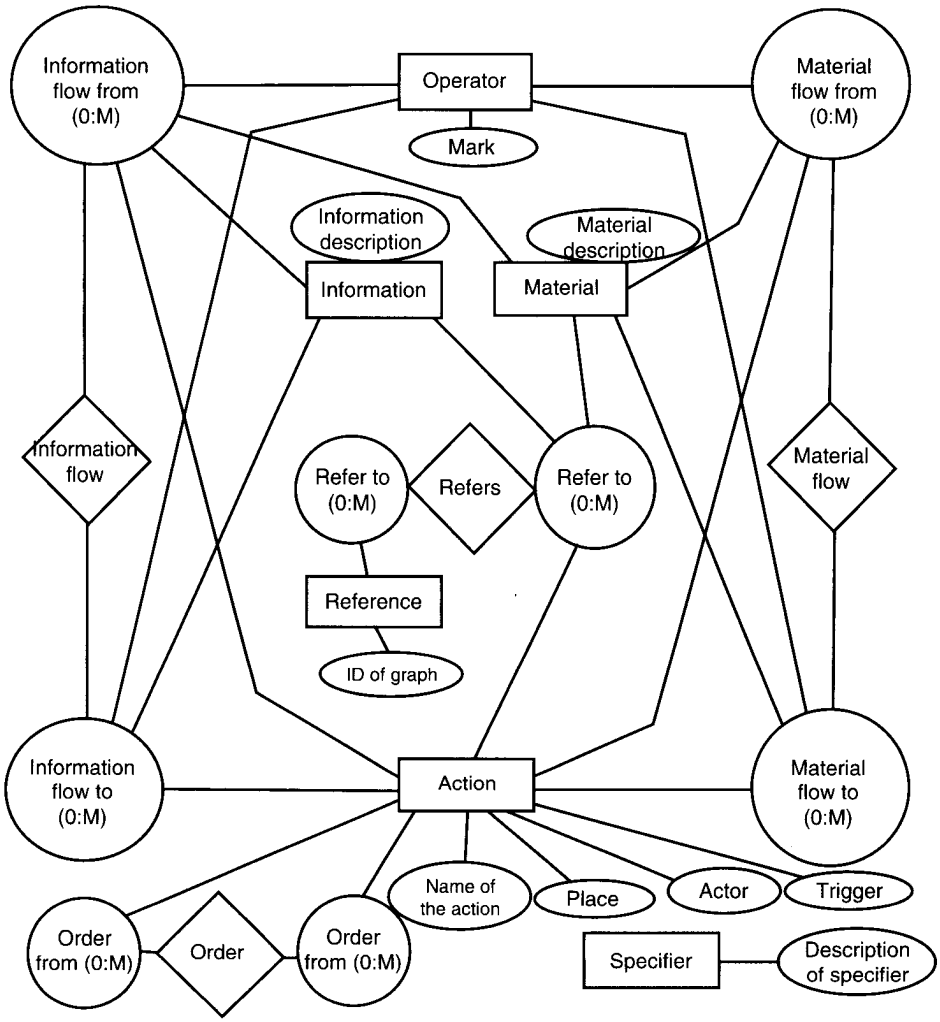


Figure 4: *OPRR model of the action diagramming part of the Activity Modeling*

relationship types. This more rigorously specified version of the method leads to more uniform descriptions, but it will also limit the system developers' freedom in modeling situations by reducing the opportunities for contextual modifications of the method. Similarly, in some design situations a need can arise for a method which will channel developers' observations into dynamics of system behavior, such as the *duration* times or turn-around times. By defining additional properties such as duration, and linking them into actions, the method could also be used to analyze and simulate turnaround times and critical paths of time-sensitive systems. Moreover, by including a reference object in the metamodel we can obtain hypertext features to the method. In case of Activity Modeling a reference object, for example, allows links between 'flat' IS models, and therefore offers tools for managing descriptions. In practise, the property type *ID of graph* links *material sets, information sets,*and*actions* to each other via a reference object (see Figure 4).

Different modeling situations and developers' varying experience motivate alternative versions of methods. Versioning would include, for example, adding new property types or relationship types into object types with formal metamodels. Method support can also be strengthened by adding new functionality. For example, formal metamodels permit development of simulation tools (such as in Activity Modeling predicting process turnaround times or flow sizes), or to augment developers' intellect with associative memory which dynamically links different components of IS descriptions (like developing design rationale for Activity Modeling). The search for new method alternatives will also lead to a creation of wholly new methods. In a metamodeling environment new requirements could be more easily matched with a CASE tool by defining new metamodels on the fly. Such a process will lead to a more thorough understanding of the role of methods in systems development by giving freedom to experiment with alternative method uses. For example, using metamodeling tools, object-oriented approaches can be implemented rapidly and prototyped. At the same time such experiments may add new features to methods. Metamodeling thereby offers a flexible way to assess the functionality desired from CASE tools, and to enhance and clarify methods and their use in different tool environments.

*Define the representational part of the method.* The derived meta-datamodel covers the conceptual part of the conceptual-representational dimension, the type part from the type-instance dimension, and the static part from the static-dynamic dimension in Figure 2. The use of graphical methods in CASE tools, however, necessitates specifications for graphical symbols. The graphical specification is needed because descriptions derived by the method are created, manipulated and analyzed in a computer supported environment (Harel 1988). Accordingly, we must define the representational part of the conceptual-representational dimension. As depicted in Figure 3, an *information set* is represented by a parallelogram, a *material set* by a parallelogram with a double upper line, an *action* instance by two separate lines, and a *reference* is represented by a circle. Both an *operator* and a *specifier* are represented by a text. A *material flow* is described

by a thick solid line. Both an *information* flow and *refers* are shown by a thin solid line, and an order by a dashed line ending to an arrow (see Figure 3).

### 3.4 Implementing the Method in a CASE Shell

We have discussed so far only the static part of the development method, i.e. the way that the method specification relates to the stored data in a CASE tool. The full use of the method, however, requires the development of a process model of the method's use, transformations and reporting facilities. In our case we have not fully taken into account these aspects of method's use due to the current inadequate support for meta-activitymodeling in the CASE shell used. The transformation and reporting facilities in CASE shells or metasystems are implemented by query facilities or report generators. In the case of Activity Modeling, reporting facilities could analyze IS models for their consistency and examine parallelity or redundancy in input-output flows. This requires only that appropriate and specific algorithms are developed for these tasks.

By developing a metamodel of an Activity model and complete it with appropriate tools we obtain a computer-supported tool for activity-analysis and design. In MetaEdit the adaptation is done by compiling graphical metamodels (see Figure 4) into executable method definitions. Figure 5 describes an example of a resulting activity graph and its use in MetaEdit. It also describes a dialog of properties for 'invoicing' activity.

### 3.5 Evaluation of the Method Adaptation

In principle, by using the metamodeling approach we can customize a large majority of current methods into CASE shells. The success of the method adaptation, however, depends on how completely the knowledge of the method and its use can be represented in the metamodeling language. Due to the poor or semi-formal definitions of currently used manual methods obtaining sufficient formal representation of a method is rarely a straightforward process. Thus, the adaptation process faces several obstacles in increasing the formality of the modeled method.

Although a more formal specification of the method is required in the adaptation, the degree of formalization is not the only factor affecting the success of method adaptation. We apply in the following the evaluation framework discussed in Section 2.2 to estimate how successfully the modeling of the Activity Modeling was done. In the evaluation we concentrate only on the static part of the method, and therefore estimate only how the meta-datamodel fulfils the requirements set out in the evaluation framework.

Table 1 summarizes results of the evaluation. A cross denotes that the metamodel meets the requirement and a cross in brackets that the current support is limited. Additionally, the analysis has been extended by specifying whether the metamodel satisfies each requirement during a method's run-time use (intra), or through transformations, i.e. by running analysis reports from produced system

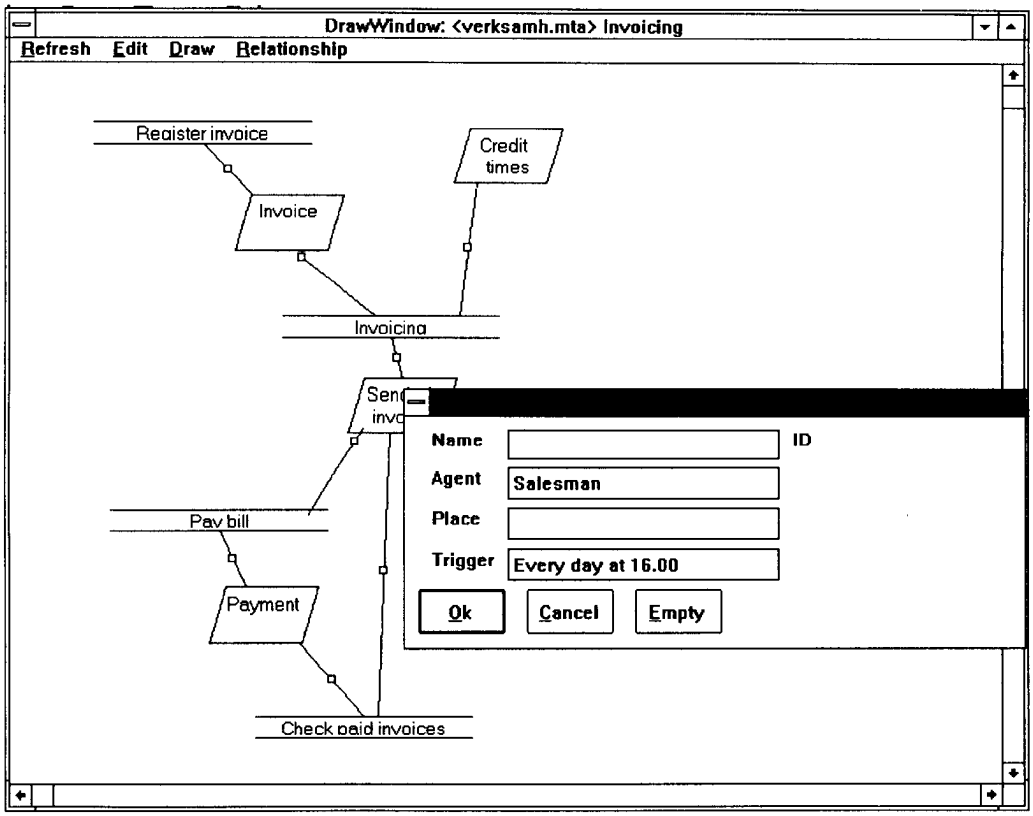


Figure 5: *An example of activity modeling in the MetaEdit CASE shell*

descriptions after the use of the tool (post). Finally, it is obvious that the evaluation of the metamodeling process is not as straightforward as presented here, and clearly not as discrete as table 1 depicts. The results are based on our own personal estimation of the level of the use.

| Evaluation criteria | Value | Type of satisfaction |
|---|---|---|
| (1) All relevant aspects of the universe of discourse can be described completely | X | intra |
| (2) System descriptions made by the method are consistent | (X) | intra/post |
| (3) System descriptions made by the method are valid | (X) | post |
| (4) The concepts of the method are clearly defined | X | intra |
| (5) The use of the method is deterministic | | |
| (6) The method corresponds with user's natural concepts | X | intra/post |
| (7) All developer's information requirements are covered | * | |
| (8) The use of the method is formalized | (X) | intra/post |
| (9) System descriptions described by the method are useful in communication | (X) | intra/post |
| (10)The method use reduces the complexity of the system | (X) | post |
| (11)The method use can be divided into steps | | |
| (12)The method is an integrated part of a larger methodology | X | intra/post |

*Depends on developers' needs in specific situations

Table 1: *The evaluation framework*

When examining the table we can conclude the following. First, the metamodel captures both graphical and textual features related to the Activity Modeling, i.e. identify the action, information and material objects, and alternative flow types between these objects. Secondly, the metamodel includes some constraints to check the consistency of system descriptions, such as connectivity and identification. Due to the weak support of the OPRR model for some other constraints the descriptions must also be analyzed afterwards. For example, in Activity Modeling references between objects in various descriptions must be analyzed by collecting all reference types and their relationships to other object types, i.e. to *action, material set*and*information set.* How the metamodeling process meets the third requirement concerning the validity of system descriptions is difficult to assess, because validity on a semantic level is impossible to capture in the metamodel. Here, the repository aids developers to check conflicts, omissions and errors, which is likely to increase the validity of the results. Fourth, due to the limited number of concepts in the Activity Modeling—six object types and four relationship types—a rigorous definition of the metamodel is possible. The only modeling alternatives are whether to add a *trigger* as a property type into an *action* instead of creating an object type of its own, and whether to create an object type *specifier* instead of adding it as a property type into relationship types. Fifth, because the deterministic use of the method is not included into

the meta-datamodel it is not discussed here. Because the terms in the Activity Modeling were given, the appropriateness to user's natural concepts can not be evaluated. In contrast, if the evaluation is based on the requirements defined by Goldkuhl (1990), the metamodel fills this requirement. Similarly, because Activity Modeling was selected as is, no extensions were made to create an OPRR specification of the method. Appropriate use of the method in specific situations is difficult to assess, and therefore not included in the evaluation.

Due to the fairly simple concepts introduced, formalization of the static part in Activity Modeling was easily achieved. Accordingly, operations related to the method, such as connecting properties to objects or relationships between objects, can be carried out. The contextual use of the Activity Modeling however, is not possible or even appropriate to formalize. Ninth, the communicational requirement is satisfied by using reports generated, such as collecting all input)output of information and material sets from each actor of an action. In the original Activity Modeling this report is used in selecting ISs, which are analysed further. Tenth, Activity Modeling does not apply decomposition as many structured methods do. Instead, it includes a reference object, which aids in connecting system descriptions produced by different developers. The content of separate descriptions, however, such as the amount of material sets or content of actions, is fully dependent on requirements revealed by method users. Eleventh, the static knowledge captured into the metamodel does not assist in splitting the method use into smaller tasks. One alternative phasing could be to divide the process roughly into two steps: generating graphical system descriptions and transforming them into textual reports needed in later stages. Finally, the input to Activity Modeling is a list of development goals. By using these the developers concentrate on the essential organizational issues and represent these in graphical models. The possible output is a list of 'important' actions, i.e. possible ISs to be designed, and material and information sets related to them. By using automatic reports this labour-intensive and error-prone analysis task is eased.

# 4 Conclusions

In this paper we have analyzed how to improve the companionship between methods and CASE tools. We have suggested a metamodeling approach to adapt methods into CASE, which seeks to develop a more balanced perspective on method-tool companionship. We believe that the metamodeling approach can decrease weaknesses observed in methods with a 'pen and paper' mentality, such as: labour-intensive, error-prone, and unsystematic use. Similar results have been attained in method adaptations that have already been done with method-dependent CASE tools (Aaen *et al.* 1992); in these cases, however, no evaluation and reflection is possible for a potential CASE user concerning the possible benefits other adaptations could offer. Therefore, the metamodeling approach not available in tool-driven CASE gives the opportunity to incrementally develop

methods for local needs. Furthermore, the incremental development can be iterative and use quick-and-dirty tactics to revise metamodels based on repeated attempts.

We argue that by applying incremental method development the nature of development methods will change. Because there are many alternatives available during the adaptation process this will rapidly change methods locally. First, the methods' concept structures will change into richer and more formalized structures. In the methods with a 'pen and paper' mentality the conceptual structures have been formulated narrowly, and often loosely, for practical reasons. The situation differs in computer-aided environments, where the repository can capture more complex concepts, and tools can aid in the use and enforcement of methods. Second, an incremental approach to method modeling will lead to flexible development, selection and use of methods. The use of methods and experience with them will create method libraries that help to adjust and select methods quickly for specific needs. Third, it allows the use of user-specific methods. Instead of defining frameworks for methodology use at an organization or a project level it is possible to tailor methods to be used according to individual developers' experiences or likings. On the individual level the acceptance of a new CASE tool can therefore be more easily achieved. Furthermore, it is possible to model different method versions for specific tasks. For example, a schema definition can be done collaboratively with users applying a simple ER method which helps in gathering basic entities and their attributes. Later on, system developers can use this information using a more formalized method which can identify relationships between entities, and defines candidate keys and data types.

It must be pointed out, however, that the use of customized tools based on incremental method development requires an organization to have a certain maturity in methods use. Thus, when studies show surprisingly low knowledge and use of development methods, see (Brodie 1987, Chikofsky 1988), and poor capabilities to manage IS development, see (Humphrey 1988), the use of customized CASE tools is not necessarily realistic on a large scale. Partly for these reasons the adaptation and modeling of development methods should be done on a smaller scale and gather experience gradually. Too radical a variation in methods will also lead to dysfunctional outcomes which should also be taken into account.

The outlined adaptation process suggests several alternatives for modeling a development method in a computer-aided environment. Basically, all static knowledge of the method is modeled in a meta-datamodel. The evaluation presented reveals, however, that some aspects of the method were not included in the metamodel, and thus can not be supported by the current CASE tools. Because methods have not been developed solely for CASE environments achieving a well-founded companionship is not a straightforward process. According to our experience of adaptations methods must be defined more formally, and they must be free from conflicts. Moreover, methods' concept structure will grow more richer and complex. Finally, adaptation using CASE shells can be done in a ways

which honors an organizations' standards, but it can foster the development of system-specific or user-specific methods.

The growing use of CASE tools emphasizes the role of methods in CASE introduction, because the use of the tool is not possible without a proper understanding of methods. We need to understand and address the challenges of developing and selecting IS development methods which can best be supported by CASE tools, and fit into the technical, organizational and cultural environment of organizations: this process, however, is still in its infancy. Method adaptation offers one feasible approach to improve the integration between tools and methods. To achieve full companionship between methods and CASE tools also necessitates modeling of methods' dynamic aspects, such as their behavior and use. Therefore, we will continue to develop more suitable methods for metamodeling and try to implement them into a customizable CASE tool. Furthermore, in order to develop methods incrementally, we need to study how method changes can be managed in the customized CASE environments.

## Notes

1. Method is used in the same meaning as in the CRIS-conferences (Olle *et al.* 1982, 1983, 1986). A methodology is defined as an organized collection of methods and a set of rules which state by whom, in what order, and in what way a chosen set of method is used (Smolander *et al.* 1990a).
2. In such environments a method has more specific meaning: it is limited to denote those parts of 'a method' which can be formally represented in a CASE tool.
3. In fact Brinkkemper (1990) uses a term technique in lieu of a method. In this study we use a term technique to refer to a procedural part of a method. For example, functional decomposition is a technique used as a part of data-flow analysis method, which forms one method in a larger methodology—Structured Analysis (Yourdon 1989).
4. Originally Brinkkemper has presented these requirements for assessing procedures of modeling. We assume here that these requirements suit also to the evaluation of modeled methods with some modification.
5. Hereafter referred to as Activity modeling.
6. OPRR stands for Object, Property, Role, Relationship model.

## References

ANSI, (1978). *The ANSI/X3/SPARC DBMS Framework report of the Study Group on Database Management Systems.* Information Systems 3, Pergamon Press. Pages 173–191.

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

Brinkkemper, J., (1990). *Formalisation of Information Systems Modelling.* Thesis Publishers.

Brinkkemper, S., A. ter Hofstede, D. Verhoef & G. Wijers, (1990). A CASE-tool Shell to Facilitate the Customisation of Requirements Engineering Support, Position paper. In: R. J. Norman & R. van Ghent, editors. *CASE'90 Fourth International Workshop on Computer-Aided Software Engineering.* Irvine, California, December 5-8. IEEE Computer Society Press, Los Alamitos. Pages 302–303.

Brinkkemper, S., M. de Lange, R. Looman & F. H. G. C. van der Steen, (1989). On the derivation of method companionship by metamodelling. In J. Jenkins, editor. *CASE'89 Third International Workshop On Computer-Aided Software Engineering.* London, United Kingdom, July 17-21. IEEE Computer Society Press. Pages 266–286.

Brodie, M.L., (1987). *Automating Database Design and Development.* GTE Laboratories Inc., Waltham.

Bubenko, Jr., J., (1988). *Selecting a Strategy for Computer-Aided Software Engineering (CASE).* SYSLAB-report Nr. 59, University of Stockholm, Sweden.

CASE, (1989). Where Do Repositories Come From? *CASE Outlook*, 4:20–29.

Chen, P., The entity-relationship model—toward a unify view of data. *ACM Transactions on Database Systems*, 1(1):9–36.

Chikofsky, E., (1988). Software Technology People Can Really Use. *IEEE Software* (March):8–10.

Curtis, B., M. Kellner, J. Over, (1992). Process Modeling. *Communications of the ACM*, 35(9):75–90.

Declerfayt, O., & E. Milgrom, (1989).. Requirements for the Next Generation of CASE tools. In J. Jenkins, editor. *CASE'89 Third International Workshop On Computer-Aided Software Engineering.* London, United Kingdom, July 17-21. IEEE Computer Society Press. Pages 575–587.

van Gigch, J.P., (1991). *System design modeling and metamodeling.* Plenum Press, New York.

Goldkuhl, G., (1990). *Kontextuell verksamhetsanalys med handlingsgrafer.* Intention AB.

Goldkuhl, G., (1992). Contextual activity modeling of information systems. In: *Proceeding of the Third International Working Conference on Dynamic Modelling of Information Systems.* Noordwijkerhout, June 9-10.

Harel, D., (1988). On visual formalism. *Communications of the ACM*, 31(5):514–530.

Heym, M. & H. Österle, (1992). A Reference Model of Information Systems Development. In K. E. Kendall, K. Lyytinen & J. I. DeGross, editors. *The Impact of Computer Supported Techologies in Information Systems Development.* Amsterdam, North-Holland. Pages 215–240.

Humphrey, W. S., (1988). Characterizing the Software Process: A Maturity Framework. *IEEE Software*, (March):73–79.

Kottemann, J. & B. Konsynski, (1984). Dynamic Metasystems for Information Systems Development. In: *Proceedings of the 5th ICIS.* Pages 187–203.

Lustig, G. & S. Ersson, (1989). How CASE-tools and methods relate to each other - Experiences. In B. Steinholtz, A. Solvberg & L. Bergman, editors. *CASE89 The First Nordic Conference on Advanced Systems Engineering.* Stockholm, Sweden, May 9-11. SISU.

Loh, M. & R. Nelson, (1989). Reaping CASE Harvests. *Datamation*, 1(July):31–34.

Lyytinen, K., V.-P. Tahvanainen & K. Smolander, (1991). *Computer Aided Methodology*

*Engineering (CAME)—A Research Proposal.* University of Jyväskylä, Department of Computer Science and Information Systems, Jyväskylä.

Mercurio, V., B. F. Meyers, A. M. Nisbet & G. Radin, (1990). AD/Cycle strategy and architecture. *IBM Systems Journal*, 29(2):170–188.

Olle, T. W., H. G. Sol & A. A. Verrijn-Stuart, editors, (1982). *Proceeding of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies.* North-Holland, Amsterdam.

Olle, T. W., H. G. Sol & C. J. Tully, editors, (1983). *Proceeding of the IFIP WG 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies.* North-Holland, Amsterdam.

Olle, T. W., H. G. Sol & A. A. Verrijn-Stuart, editors, (1986). *Proceeding of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the Practise.* North-Holland, Amsterdam.

Olle, T. W., A. A. Verrijn-Stuart & L. Bhabuta, editors, (1988). *Proceeding of the IFIP WG 8.1 Working Conference on Computerized Assistance During the Information Systems Life Cycle.* North-Holland, Amsterdam.

Patel, V., (1989). Experiencies of Using a CASE Tool on a Large Software Project. In J. Jenkins, editor. *CASE'89 Third International Workshop on Computer-Aided Software Engineering*, Supplementary volume. London, United Kingdom, July 17-21. IEEE Computer Society Press. Pages 479–491.

Pelly, P. K., (1990). A CASE for Influencing Systems Development Methodology. In R. J. Norman, & R. van Ghent, editors. *CASE'90 Fourth International Workshop on Computer-Aided Software Engineering.* Irvine, California, Dec. 5-8. IEEE Computer Society Press, Los Alamitos. Pages 502–518.

Shapiro, D., (1991). Introducing CASE at Bell Canada. In: *Quality Engineering Workshop.* Ottawa, Canada, October 16-17. OCRI Publications.

Smolander, K., (1991). OPRR—a model for modelling systems development methods. In V.-P. Tahvanainen & K. Lyytinen, editors. *Proceedings of the Second Workshop on the next Generation of CASE tools.* Trondheim, Norway, May 11-12. Technical Report 1, Department of the Computer Science and Information Systems, University of Jyväskylä, Jyväskylä. Pages 135–151.

Smolander, K., P. Marttiin, K. Lyytinen, V.-P. Tahvanainen, (1990a). MetaEdit—a flexible graphical environment for methodology modeling. In S. Brinkkemper, & G. Wijers, editors. *The next Generation of CASE-tools.* Nordwijkerhout, Netherlands, SERC, Utrecht.

Smolander, K., V.-P. Tahvanainen & K. Lyytinen, (1990b). How to Combine Tools and Methods in Practise—a Field Study. In B. Steinholtz, A. Solvberg & L. Bergman, editors. *Lecture Notes in Computer Science, Second Nordic Conference CAiSE'90.* Stockholm, Sweden. Pages 195–211.

Sorenson, P., J.-P. Tremplay, & A. McAllister, (1990). The Metaview System for Many Specifications Environments. *IEEE Software*, (March):30–38.

Tagg, B., Implementing tool support for box structures. *IBM Systems Journal*, 29(1):79–89.

Teichroew, D., P. Macasovic, E. Hershey III, Y. Yamato, (1980). *Application of the Entity-Relationship Approach to Information Processing Systems Modeling.* ISDOS-project, The University of Michigan.

Tolvanen, J.-P., P. Marttiin & K. Smolander, (1993). An Integrated Model for Information Systems Modeling. In J. F. Nunamaker, Jr., R. H. Sprague, Jr., editors.

*Proceedings of the 26th Annual Hawaii International Conference on Systems Science.* Vol. 3, IEEE Computer Society Press. Pages 470–479.

Verheijen, G. & J. Bekkum, (1982). NIAM: An information analysis method. In (olle *et al.* 1982).

Welke, R. & G. Forte, (1989). Meta Systems on Meta Models. *CASE Outlook*, (4).

Wijers, G. & H. van Dort, (1990). Experiences with the use of CASE tools in The Netherlands. In B. Steinholtz, editor. *Advanced Information Systems Engineering.*Pages 5–20.

Yourdon, E., (1989). *Modern Structured Analysis.* Prentice-Hall, Englewood-Cliffs.

Aaen, I., Siltanen, A., Sørensen, C., Tahvanainen, V.-P., (1992). A Tale of Two Countries — CASE Experiences and Expectations. In: K. E. Kendall *et al.*, editors. *The Impact of Computer Supported Techologies in Information Systems Development.* Amsterdam, North-Holland. Pages 61–94.