Structuring modelling knowledge for CASE shells

T.F. Verhoef [1,2]
A.H.M. ter Hofstede [1,3]
G.M. Wijers [1]

SOCRATES project

[1] Software Engineering Research Centre
P.O. Box 424, 3500 AK Utrecht, the Netherlands
e-mail socrates-info@serc.nl

[2] Delft University of Technology
Faculty of Technical Mathematics and Informatics
Department of Information Systems
P.O. Box 356, 2600 AJ Delft, the Netherlands

[3] University of Nijmegen
Department of Information Systems
Toernooiveld, 6525 ED Nijmegen, the Netherlands

Abstract

In the SOCRATES project, it is claimed that improving automated support of information modelling processes should be realised by the development of a new architecture of CASE tools, which is method independent and as such should be implemented as a CASE shell. A CASE shell should include knowledge of both the tasks to be carried out in the information modelling process and the models resulting from that modelling process. By incorporating this knowledge in a CASE shell, the shell is transformed in a workbench or modelling support system for that specific modelling process. Crucial for the development of a CASE shell is the availability of a suitable technique for the uniform representation of information modelling knowledge. We will refer to such a technique as a meta-modelling technique. In this paper we present a meta-modelling technique in which modelling tasks and the resulting models can be integrated. The development of a CASE shell as such is beyond the scope of this paper. As to show the applicability of the meta-modelling technique presented, it has been used to represent Yourdon's Modern Structured Analysis.

1 Introduction

Nowadays it is widely acknowledged that for the development of information systems the usage of methods is indispensable. Automated support of these methods more and more becomes a necessity due to increasing complexity of applications, more sophisticated verification needs and growing documentation, communication and standardisation needs. Currently many CASE tools are commercially available that provide for this automated support.

Examining the use of these CASE tools one sees that they are primarily used as drawing and reporting tools instead of as modelling support tools [Wijers et al., 1990a]. The reason for this is that they tend to ignore the fact that, especially in the first phases of the development process (commonly referred to as requirements engineering), modelling is a complex problem solving process in which iteration, incompleteness and heuristics are important characteristics.

With respect to *information modelling*, emphasis is currently on modelling techniques instead of on the modelling process ([Knuth et al., 1986], [Lockemann et al., 1986], [Potts, 1989]). Information modelling is in this paper considered to be that part of requirements engineering which is concerned with a high-level, user-oriented and technology independent description of the system to be developed.

Most methods for systems development, such as JSD [Jackson, 1983], IE [Martin, 1986a, 1986b, 1988], Yourdon [Yourdon, 1989] and SDM [Turner et al., 1987], only offer guide-lines with respect to the modelling process to be performed at a global level. Using the terminology of [Seligmann et al., 1989], one can say that emphasis is on the *way of modelling* instead of on the *way of working*. The way of modelling deals with the type of models to be used, while the way of working describes how the modelling process is structured and carried out.

In the SOCRATES project ([Hofstede et al., 1989], [Hofstede et al., 1990]) it is claimed that CASE tools should include knowledge of **both** sides of information modelling (way of modelling and way of working). Knowing where to start, how to continue, what to look for, in other words a clear strategy, will contribute to the quality of the ultimate system. It should be possible to improve automated support of such a process by offering more navigation and adequate verification.

The aim of the SOCRATES project is to realise such improved automated support by the development of a new architecture of CASE tools that incorporates both the tasks within a modelling process and the models resulting from that modelling process. The envisioned SOCRATES architecture is method independent and, as such, will be implemented as a CASE shell. The CASE shell itself is without knowledge of any specific information modelling process, but requires the modelling knowledge to be offered in a specific format. By incorporating this knowledge in the format required, the CASE shell is transformed into a workbench or modelling support system for that specific process.

The concept of a CASE shell is not new. In the literature several CASE shells have been discussed (e.g. RAMATIC [Bergsten et al., 1989], Metaplex [Chen et al., 1989]), even a

tool that supports the modification of meta-models exists (MetaEdit [Smolander et al., 1990]). However, all these shells focus on the support of modelling techniques and hardly pay attention to the modelling process. Furthermore, the degree of support of modelling techniques which they offer is limited, due to the expressive power of the knowledge representation technique used to describe the way of modelling. In the sequel we will use the term *meta-modelling technique* to denote a technique that can be used to represent knowledge about information modelling methods.

In figure 1.1 three abstraction levels are shown that have to be distinguished when discussing CASE shells. At the *application level*, we are concerned with the development of a specific application. As discussed before we distinguish between the process of modelling as prescribed by a way of working and the actual models as prescribed by a way of modelling. A similar distinction can be made at the *method level*. The models constructed at this level, called meta-models, should prescribe both the type of modelling process and the type of models to be used at the application level. These meta-models could be seen as a definition of the method at hand. At the method level we also find the process of meta-modelling: how do you go about creating meta-models? The *axiomatic level* is concerned with defining the method level.

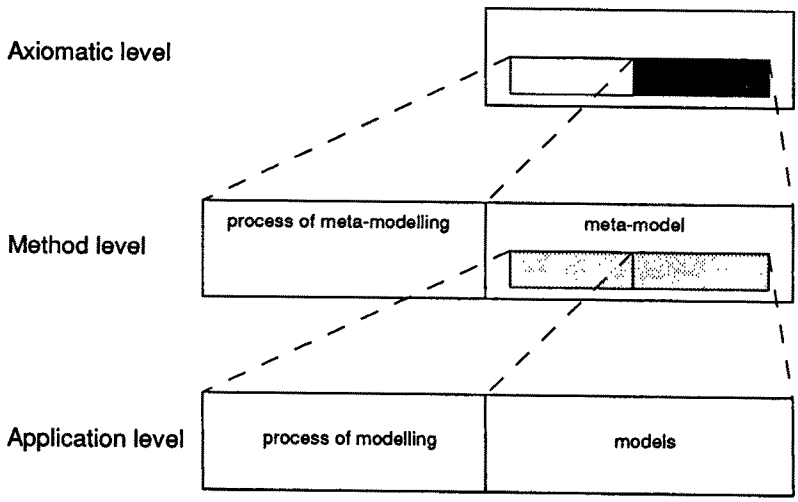

Figure 1.1 Three levels of abstraction

From the previous discussion it is clear that structuring information modelling knowledge for CASE shells has to be realised using a specific meta-modelling technique. In section 2 of this paper we focus on a meta-modelling technique, which can be used to denote meta-models (the black area in figure 1.1). For the sake of brevity we preferred to discuss this technique informally here. A complete formal discussion of the syntax and the semantics of this technique can be found in [Wijers et al., 1990c]. Furthermore, in section 3 we concentrate on the application of the defined meta-modelling technique (the grey area in figure 1.1) as to illustrate the suitability of the technique. The technique

will be applied to model Yourdon's Modern Structured Analysis [Yourdon, 1989]. The complete meta-model can be found in [Verhoef, 1991].

2 Representing modelling knowledge

In this section we present our meta-modelling technique. As stated in the introduction, this technique should represent both sides of information modelling, i.e. the way of modelling and the way of working. Section 2.1 is concerned with the representation of a way of modelling, section 2.2 with the representation of a way of working, and section 2.3 with the integration of the way of modelling and the way of working.

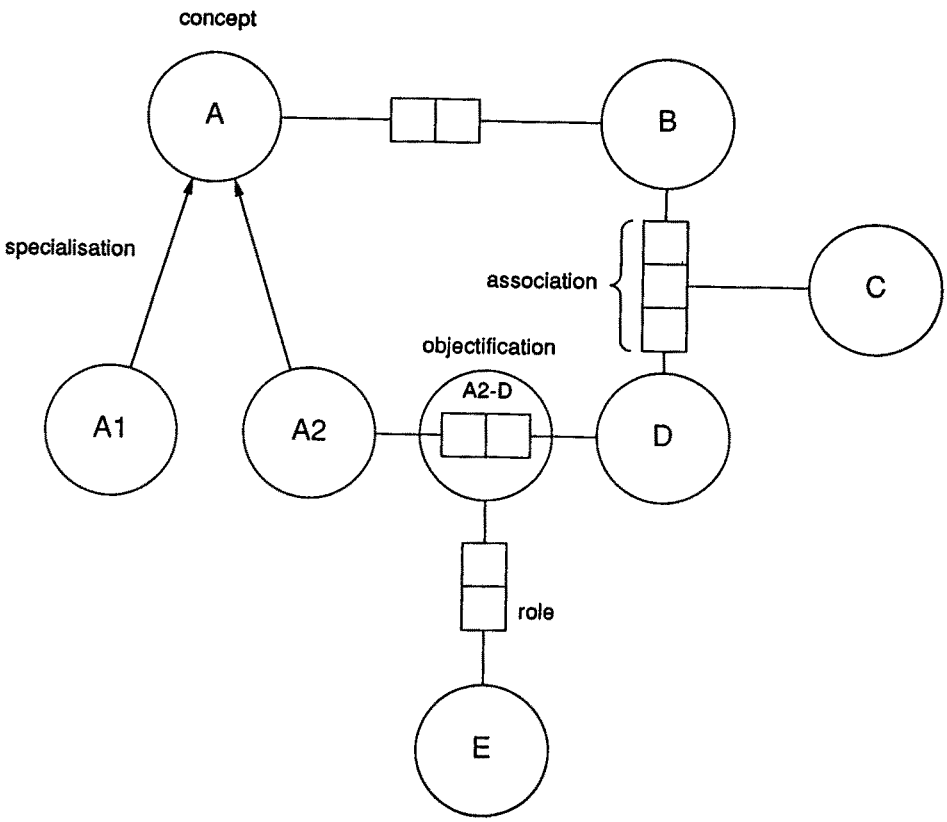

Figure 2.1 A network of modelling concepts with association, specialisation and objectification

2.1 Representing a way of modelling

The way of modelling prescribes the (interrelated) *concepts* to be used in the course of the modelling process. The application model constructed by an analyst forms an instantiation of the concepts prescribed by the way of modelling.The network of modelling concepts as applied within an information modelling process will be called the *concept structure*. The notions used when defining concept structures are illustrated in figure 2.1 using the external representation notation as defined for the NIAM method, see [Wintraecken, 1985] and [Nijssen et al., 1989]. Below, we present these notions in more detail.

The relationships between the concepts that are part of the way of modelling are denoted as *associations* in which concepts play a specific *role*, see also [Welke, 1988]. In other words associations relate two or more concepts, whereas each role that is part of the association clarifies how the specific concept is involved in the association.

Associations themselves can have associations leading to the notion of *objectified associations*. This mechanism offers the possibility to treat associations as concepts themselves. The notion of objectified association, equal to the notion of associative entity in a number of entity-relationship modelling dialects, can also be found in [Nijssen et al., 1989] and [Welke, 1988].
Cyclic objectification structures (see also [Bommel et al., 1990]) are excluded. An example of a concept structure violating this property is presented in figure 2.2.

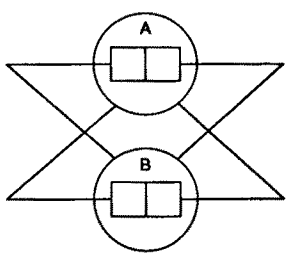

Figure 2.2 Cyclic objectification structure

Another well-known abstraction mechanism applied in concept structures is the mechanism of generalisation versus specialisation. Inheritance of associations can be realised by a *specialisation relationship* between two concepts. The so-called subconcept inherits the associations of the superconcept.

Each concept has exactly one pater familias. A concept is called a pater familias [Troyer et al., 1988] of another concept if it is an ancestor of the other concept and does not have any supertypes. This property implies that each subtype network has exactly one concept as its top. In figure 2.3 a concept structure violating this property is shown.

In figure 1.1 the three levels of abstraction, i.e. application, method and axiomatic, have been presented. Concept structures have to be placed at the method level whereas their definition has to be placed at the axiomatic level. The *semantics* of concept structures

specify how application models constructed at the application level are related to concept structures at the method level. An application model can be thought of as a set of concept instances and a set of association instances. Between these instances the same type of relationships can be found as those defined for concept structures.

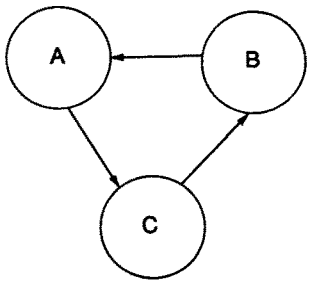

Figure 2.3 Subtype network without a pater familias

2.2 Representing a way of working

An important concept in our specification technique for the way of working is the concept of *task*. A task is perceived as something that has to be performed in order to achieve a certain goal. Tasks can be defined recursively in terms of subtasks meaning that tasks can be decomposed into a hierarchy of subtasks until a desired level of detail has been reached. If performing a task involves choices between subtasks, *decisions* represent this moment of choice. Decisions coordinate the sequence of tasks. We have illustrated the notions of task and decision in figure 2.4 using the external representation notation defined in [Bots, 1989]. The interrelated collection of tasks and decisions will be called a *task structure.*

If we zoom into task A, we arrive at a task structure consisting of the tasks B up till G. This has been established using the notion *decomposition.* When tasks have not been decomposed any further, we refer to them as *basic tasks.* There is exactly one task not the result of a decomposition. We will refer to this task as the *overall task.* If a task is decomposed, there are at least two *task objects* part of that task. A task object is either a task or a decision.

In this figure, task B is an *initial item* of task A, meaning that the flow of tasks part of task A start at task B. Initial items are essential in cyclic task structures. In task structure diagrams we apply the drawing convention to place the initial items at the top of the diagram.

With the help of *triggers* one can express that a certain task object should be performed after termination of another task object. The two task objects related to a trigger are part of the same supertask. Each decision should lead to a task by following a path of triggers. In figure 2.5 a task structure violating this property is shown.

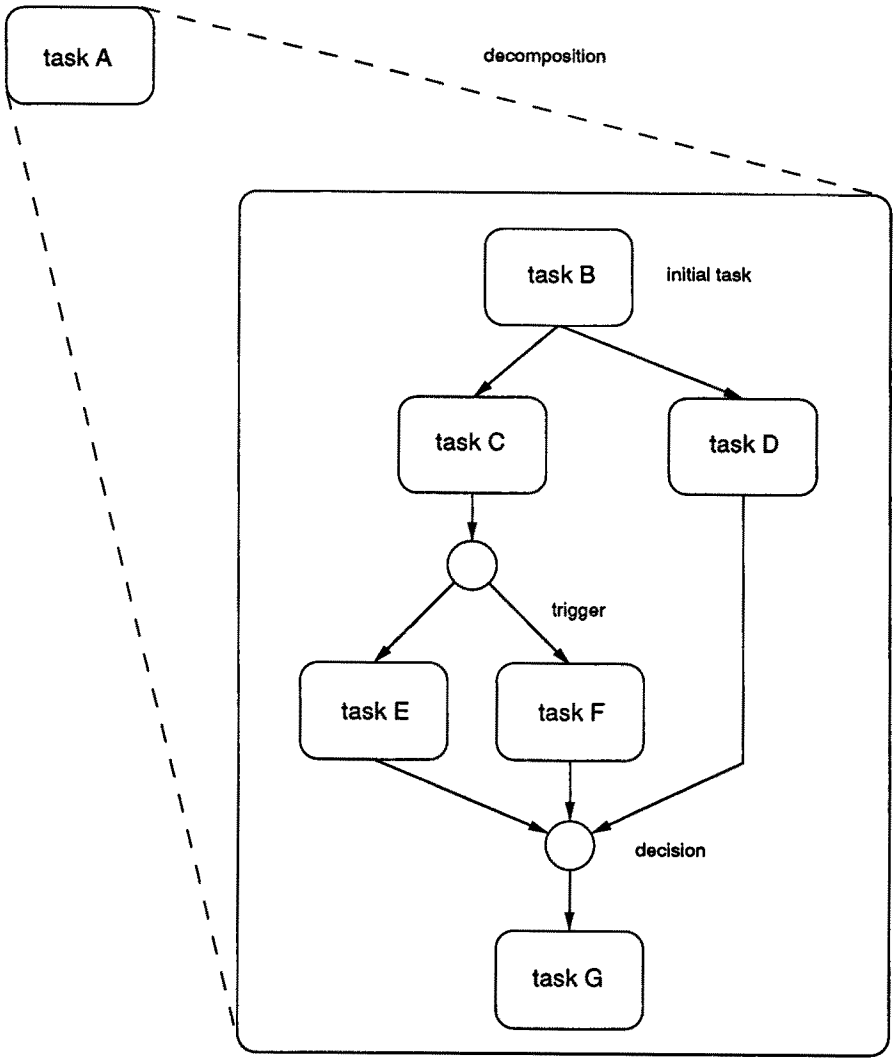

Figure 2.4 Illustration of the interrelationship between task and decision

For the operational *semantics* of task structures, again the distinction between method level and application level should be recalled. Tasks and decisions are defined at the method level. They specify how the design process (or part of it) is to be performed. Actual development of an application can be seen as the execution of task instances and decision instances. This corresponds with the application level. The dynamic interpretation of a task structure is defined in the sense that task instances and decision instances come into being in the course of the design process and can be in certain states such as waiting, active or current.

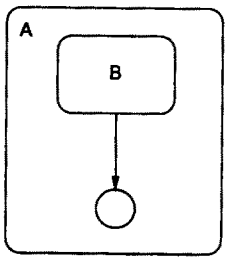

Figure 2.5 Open-ended decision

For the formal definition of the operational semantics of a task structure, a Predicate/Transition net (PrT-net) has been defined in [Wijers et al., 1990c]. The dynamics of PrT-nets themselves have been formally defined in [Genrich, 1987]. In [Wijers et al., 1990b], a prototype tool for the simulation of task structures, based upon this PrT-net, has been described.

2.3 Integrating a way of modelling and a way of working

Finally, the meta-modelling technique deals with the establishment of links between a task structure and a concept structure. Part of this integration comes into existence if we look at our definition of task. A task is something that has to be performed in order to achieve a certain goal. As we focus on modelling we can say that a task can be seen as the creation, the modification, the deletion or selection of a (partial) model. The goal of such a task can then be seen as defining the set of concepts and associations, which can be created or deleted or selected, satisfying a set of verification rules.

Task view - verification rules

That part of the concept structure that can be manipulated by a task in the task structure will be called the *task view* of that task. The set of instances of concepts and associations that are situated outside the task view of a task will never be changed by that task. Because tasks can be decomposed into subtasks in which a part of the task will be performed, the task view of a subtask has to be a subset of the task view of the supertask. On the other hand a task should be able to fully manipulate its task view, so its task view has to be the union of the task views of its subtasks.

The changes to the application model resulting from a task have to satisfy the before mentioned set of verification rules. These rules are called the *a posteriori rules* of the task. A task may only be considered terminated if no a posteriori rules are violated. An *a priori rule* constitutes a rule that may never be violated. In the same way as a posteriori rules, a priori rules are also related to tasks. They should be interpreted as stating an invariant that has to be satisfied during task execution, which implies that at task initiation these rules also have to be satisfied. Inheritance of a priori rules occurs

because of the decomposition of tasks. If a rule is invariant for a given supertask, then it is also invariant for all its subtasks.

In our view, different rules might apply to different stages of the modelling process. Therefore, we have related verification rules to tasks. Another important observation is that we have distinguished between verification at forehand and verification afterwards. Too strong a priori rules burden the analysts' creativity and exploration possibilities whereas too weak a priori rules might result in illogical and unsound models because too little guidance was offered in the course of a modelling process. The solution in the modelling knowledge technique is that verification rules have been related to tasks by which it is possible to activate or deactivate certain rules depending on the stadium of the modelling process.

Verification rules are to be expressed in first order predicate logic, where the predicates used in these rules correspond with the concepts and associations defined in the concept structure.

Basic tasks - basic procedures

Above we indicated that tasks are performed in order to achieve a certain goal which was in fact the manipulation of a partial model constrained by certain verification rules. However, in some way such a model has to be constructed or manipulated by a number of basic actions on models such as the creation of a new instance or the removal of an instance or the selection of an instance. These basic kind of actions are called *basic procedures* and as part of the integration it is required that each basic task is mapped onto a basic procedure. We recall that a basic task is a task not decomposed any further. By the mapping between basic tasks and basic procedures it is uniquely defined what a basic task intends to achieve.

Basic tasks also have a task view, a priori rules and a posteriori rules. However, as the execution of a basic task is "instantaneous", there is no semantic difference between a priori and a posteriori verification rules for basic tasks. Below we define the basic procedures that have been distinguished:

- *create_concept(c)* and *create_association(a)*

  The intended goal of these two basic procedures is to add an instance of a concept or association to the existing application model. As a consequence of the addition of a specific instance, the application model might however violate the corresponding set of verification rules (both a priori and a posteriori). In order to resolve that inconsistency other concept instances and association instances might have to be added to the application model.

- *select(c)*

  Selecting concept instances from an application model is a natural activity during the modelling process, which is also apparent when looking at existing CASE-tools. For example, a process in a dataflow diagram is selected in order to be decomposed. Naturally the selected instance(s) have to be remembered in order to be used by other tasks. These temporary places for selected instances are referred

to as *information places*. Restrictions on the way concept instances can be selected from an application model are reflected in the presence of four different select procedures. Details on the differences are given in [Wijers et al., 1990c].

- *delete_concept(c)* and *delete_association(a)*

  The delete procedures remove a concept instance or an association instance from the existing application model. Which instances can be deleted is in the first place determined by the incoming information places. For example a box is selected from a JSD entity structure diagram, which is placed in an information place. This information place is then input for a delete_concept which then deletes the box from the application model. The selected box is the only box that can be deleted by that delete procedure.

- *classify_concept(c)*

  The classify procedure is defined in order to further specialise an existing instance in the application model. In the same way as with the create procedures this might imply the creation of other new instances.

In the description of the procedures that will support the basic tasks, we have explicitly decided not to discuss the details of how these procedures would behave. It is believed that this is very much dependent on the implementation of an ultimate CASE shell. From the above descriptions it is clear *what* the procedures will do depending on their input, their a priori and a posteriori rules and the state of the application model. *How* they realise this behaviour is considered not to be part of our modelling knowledge representation technique.

Information places

In the previous part we have already mentioned the existence of information places. From the description of the select procedures it becomes obvious that the use of information places offers the possibility to define or restrict the scope or context of a task.

Tasks not only have the possibility to fill an information place but also to empty an information place. Emptying an information place is called *consuming* an information place. It is also possible that a task only *refers* to an information place, which means that the contents of the information place remain unchanged.

The difference between consume and refer can be illustrated as follows: referring to an information place will not change the contents of the information place, consuming an information place is interpreted as emptying the information place. Putting concepts into information places only take effect at the termination of a task.

Summarising, we distinguish between the following sets of information places for a task: (1) the set of information places defined as part of the task, (2) the set of information places consumed by the task, (3) the set of information places referred to by the task, and (4) the set of information places filled by the task. Both the information places

consumed by and the information places referred to by a task will be called input information places for that task.

Decision rules

Decisions can be supported by so-called decision rules. Decision rules are specified as production rules with a condition part and an action part. In the action part, the probability for each task possibly to be performed is indicated. More details can be found in [Wijers et al., 1990c].

The *semantics* of the integration of the way of modelling and the way of working describe the changes allowed to the application model by the execution of task instances. Again, we refer to [Wijers et al., 1990c].

3. Structuring Modern Structured Analysis

This section is concerned with the application of the meta-modelling technique presented in the previous section on Yourdon's Modern Structured Analysis, as to illustrate the suitability of the meta-modelling technique. In section 3.1 the representation of the way of modelling is outlined. Section 3.2 deals with the representation of the way of working. Finally, in section 3.3 the integration between the way of modelling and the way of working is realised.

3.1 The way of modelling in Modern Structured Analysis

With respect to the way of modelling, the application of the meta-modelling technique is restricted to the two main techniques presented, i.e. the data flow diagrams and the entity-relationship diagrams. The explanation of these techniques is based on [Yourdon, 1989]. Meta-models of other Yourdon diagramming techniques can be found in [Verhoef, 1991].

3.1.1 Data flow diagrams

Data flow diagrams (DFDs) constitute a technique which is often used to describe the functions to be performed by an information system to be developed, and the transformations to be carried out by the system. DFDs contain four main component types: *processes*, *flows*, *stores* and *terminators*. Processes show those parts of the system that transform data. Flows are used to denote packets of data moving into or out of processes. Stores model collections of data at rest. Finally, terminators represent external entities with which the system communicates.

Stores, processes and terminators are connected by means of flows. Other characteristics that they have in common are the fact that they can be identified by a name and, obviously, the fact that they are part of a DFD. For these reasons, they can be regarded as subtypes of a generic *DFD Object*.

Different kinds of flows are distinguished. Two examples of these kinds are shown in figure 3.1. From these examples it is clear that it is necessary to discriminate between flows and *flow ends*. Flow ends can be either *incoming flow ends* or *outgoing flow ends*.

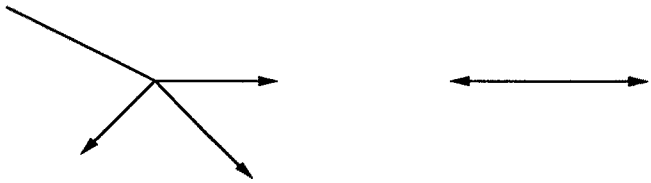

Figure 3.1 An example of a diverging flow and a dialog flow

Examining processes, a distinction should be made between *control processes* and *data processing processes*. Control processes are specified in *state transition diagrams* (STDs). Data processing processes can either be specified in *process specifications* or decomposed into DFDs.

External stores are a special type of stores that can only be part of a special kind of DFD, the context diagram. Terminators can only be part of context diagrams. For identification purposes, DFDs and processes can be numbered, and DFDs, DFD Objects and flow ends can be named.

The concept structure resulting from this description is presented in figure 3.2.

3.1.2 Entity-relationship diagrams

The *entity relationship diagram*ming technique (ERD) is often used to describe the different kinds of data and their relationships within the system to be developed. ERDs contain two main component types: *object types* and *relationships*. Object types represent a collection or set of objects (things) in the real world whose members play a role in the system being developed, can be identified uniquely, and can be described by one or more facts (*attributes*). Relationships represent a set of connections or associations between object types, where object types can occur more than once in a relationship, in different roles (*number of occurrences*).
An attribute is the occurrence of a *data element* in the description of an object type. Data elements, contrary to object types, can be mapped onto concrete data types, such as strings, integers and booleans. Attributes can be part of the identification of object types.

Two more notions are introduced, *subtype indicators* and *associative object type indicators*. Subtype indicators denote object types being categories of other object types, whereas these categories are described into more detail, i.e. by more data elements. Associative object type indicators denote that a relationship itself can be interpreted as an object type.

The resulting concept structure is presented in figure 3.3.

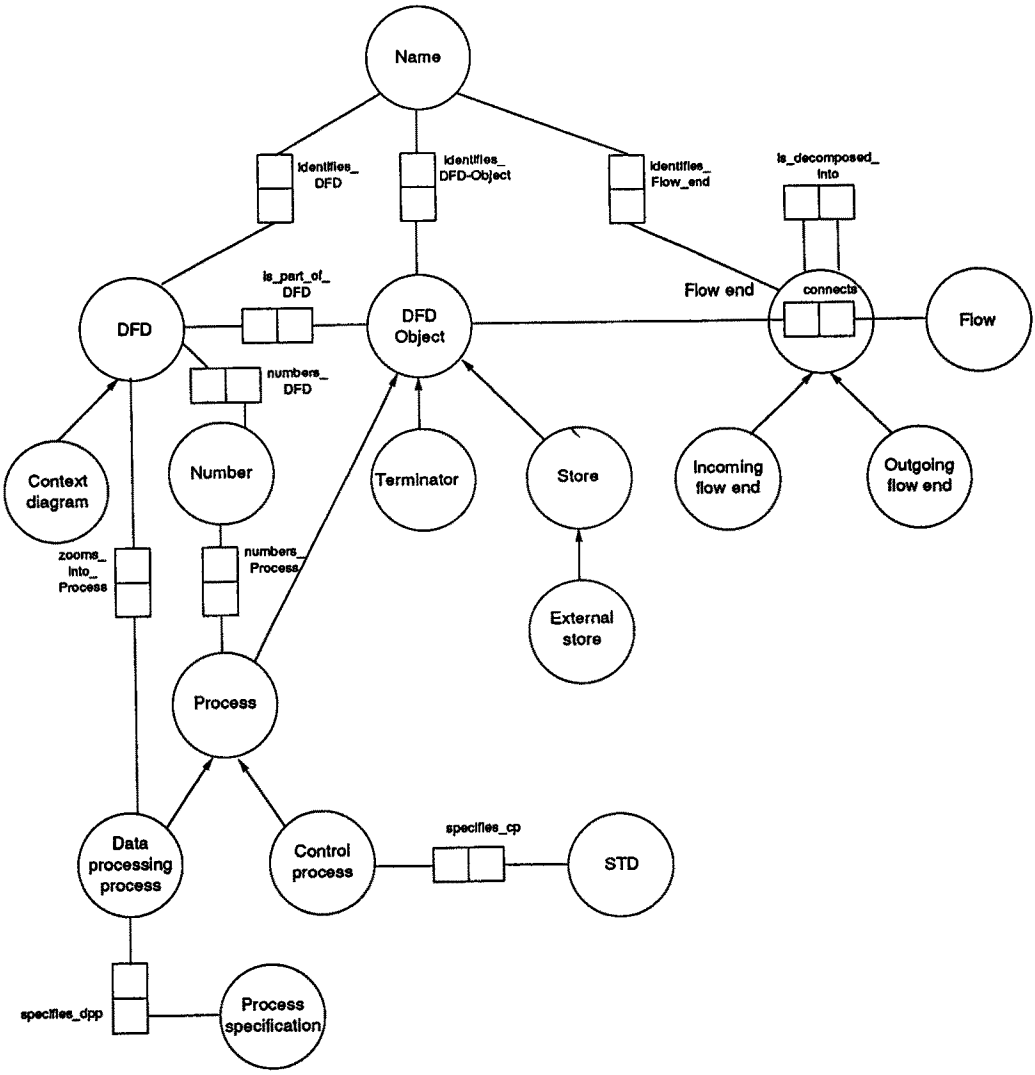

Figure 3.2 Concept structure of the Data Flow Diagramming technique

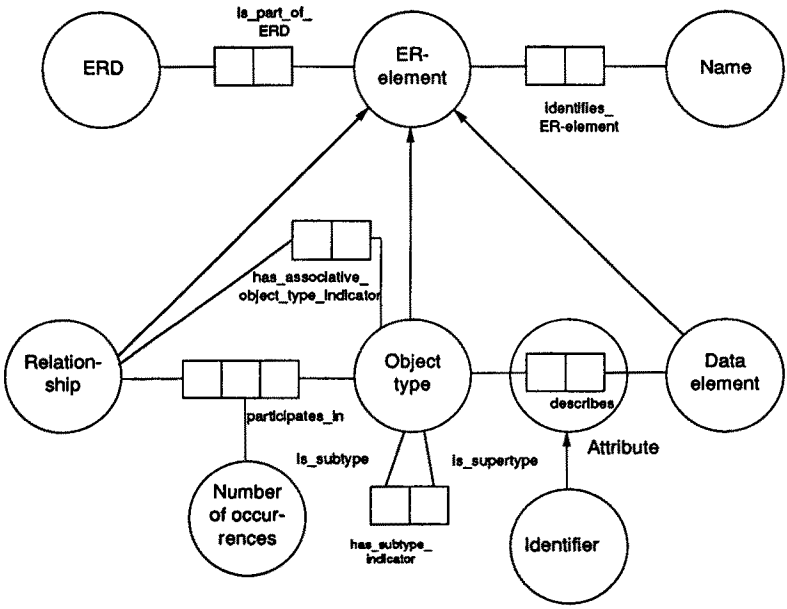

Figure 3.3 Concept structure of the Entity Relationship Diagramming technique

3.2 The way of working in Modern Structured Analysis

The application of the defined meta-modelling technique on the way of working in Modern Structured Analysis has been restricted to the construction of the essential model. The task structure of this overall task is presented in figure 3.4. The construction of the essential model can be decomposed into two tasks, of which the task 'Construct the behavioral model' has to be performed after the task 'Construct the environmental model'. In the sequel of this paper, we only pay attention to the construction of the environmental model. A complete meta-model of essential modelling can be found in [Verhoef, 1991].

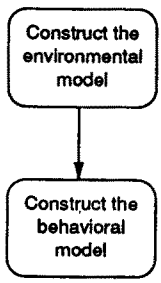

Figure 3.4 Task structure of Yourdon's Modern Structured Analysis

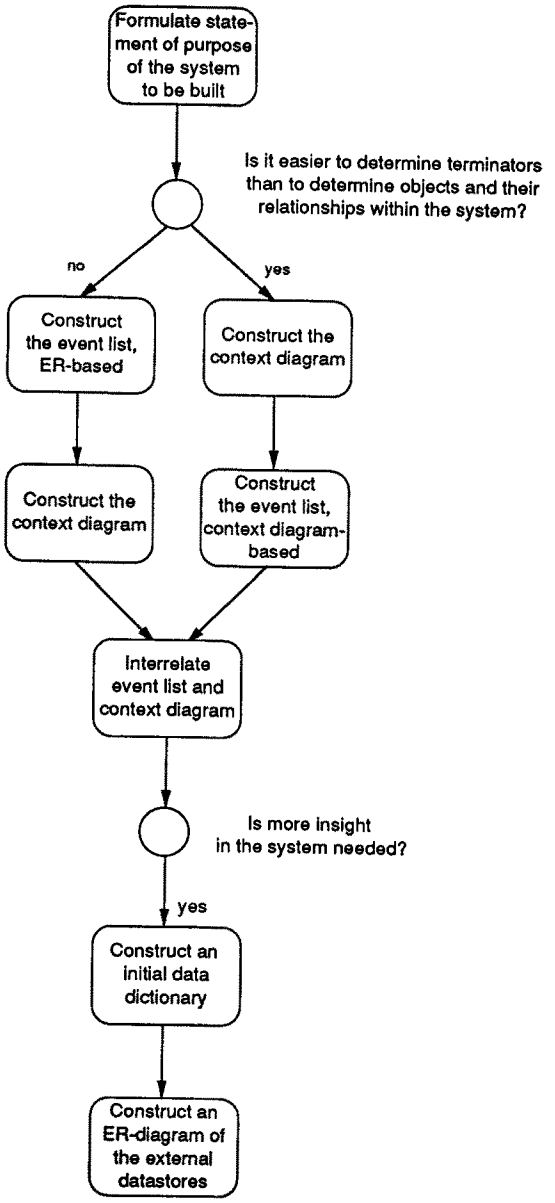

Figure 3.5 Task structure of the task 'Construct the environmental model'

The construction of the environmental model should deliver the formulation of the statement of purpose of the system to be built, an event list and a context diagram. When these products have been delivered and interrelated, it has to be decided whether more insight in the system is needed. If so, an initial data dictionary and an ER diagram of the

external stores have to be constructed. If the outcome of the decision is negative, the task 'Construct the environmental model' is considered to be completed.

With respect to the order in which the context diagram and the event list have to be constructed, an additional guideline is offered in the form of a decision. This guideline states that in a situation in which it is easy to determine the terminators, the strategy starting with the task 'Construct the context diagram' is to be preferred. Independent of the strategy chosen, a proper correspondence between the context diagram and the event list has to be established.
This decomposition of the task 'Construct the environmental model' is shown in figure 3.5. All these tasks can be decomposed into more detail.

3.3 Integrating the way of modelling and the way of working in Modern Structured Analysis

This subsection deals with the integration of the way of modelling and the way of working. This integration will be demonstrated by an example. Again, more details can be found in [Verhoef, 1991]. A close examination of the task 'Construct the event list, ER-based' in figure 3.5 shows that this task can be decomposed one more level of detail than in [Yourdon, 1989], as presented in figure 3.6. In this subsection, the subtask 'Create an initial ERD' is taken as an example. Because this subtask is not decomposed any further, this task will be referred to as a *leaf task*. Leaf tasks are to be decomposed in the integration of the way of modelling and the way of working to the level of basic tasks.

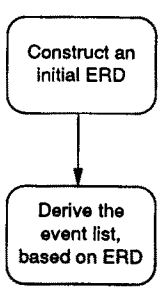

Figure 3.6 Task structure of the task 'Construct the event list, ER-based'

3.3.1 Determining task views

The task structure constructed so far can be seen as a decomposition hierarchy. A first activity in integrating the way of modelling and the way of working is the determination of the task views, i.e. those parts of the concept structure that are manipulated by the respective tasks in that decomposition hierarchy.

For the leaf task chosen, 'Create an initial ERD', this task view amounts to a subset of the concepts use to denote the ERD technique. In this stage of data modelling, it is not

necessary to pay attention to specific data elements or to associative object type indicators and subtype indicators. The task view resulting from this observation is given in figure 3.7.

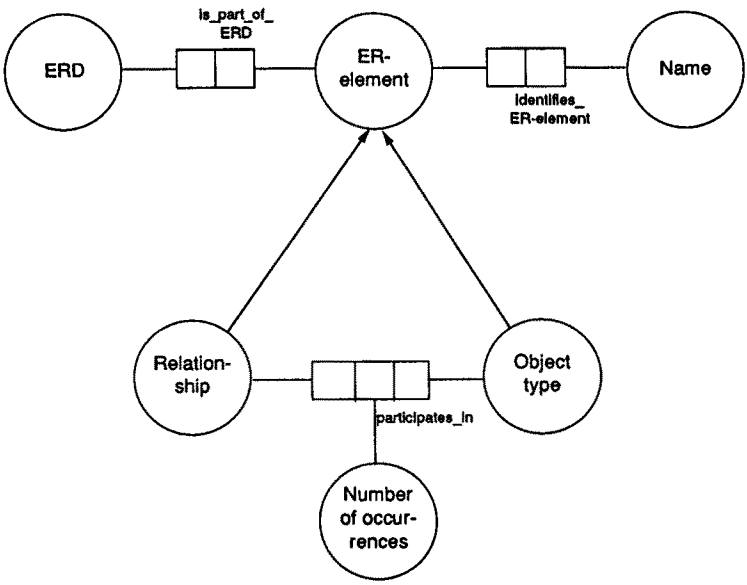

Figure 3.7 Task view of the task 'Construct an initial ERD'

3.3.2 Determining verification rules

For the leaf task 'Create an initial ERD' a set of verification rules has to be defined. Verification rules are classified as either a priori or as a posteriori, as stated in section 2. An example of an a priori rule valid for this task is:

- All ER-elements are part of exactly one ERD.

  $\forall e$ [ER-element(e) $\Rightarrow \exists!d$ [ERD(d) $\wedge$ is_part_of_ERD(e,d)]]

Two examples of a posteriori rules valid for this task are:

- Every ERD contains an Object type.

  $\forall d$ [ERD(d) $\Rightarrow \exists o$ [Object type(o) $\wedge$ is_part_of_ERD(o,d)]]

- All Object types participate in a Relationship.

  $\forall o$ [Object type(o) $\Rightarrow \exists r \exists n$ [Relationship(r) $\wedge$ Number of occurrences(n) $\wedge$ participates_in(o,n,r)]]

By determining this set of verification rules, it is illustrated that the idea to link verification rules to tasks instead of to concepts increases flexibility as regards the use of information modelling knowledge specifications. For instance, the verification rule 'Every Relationship should have a name' is not valid in this preliminary stage of data modelling.

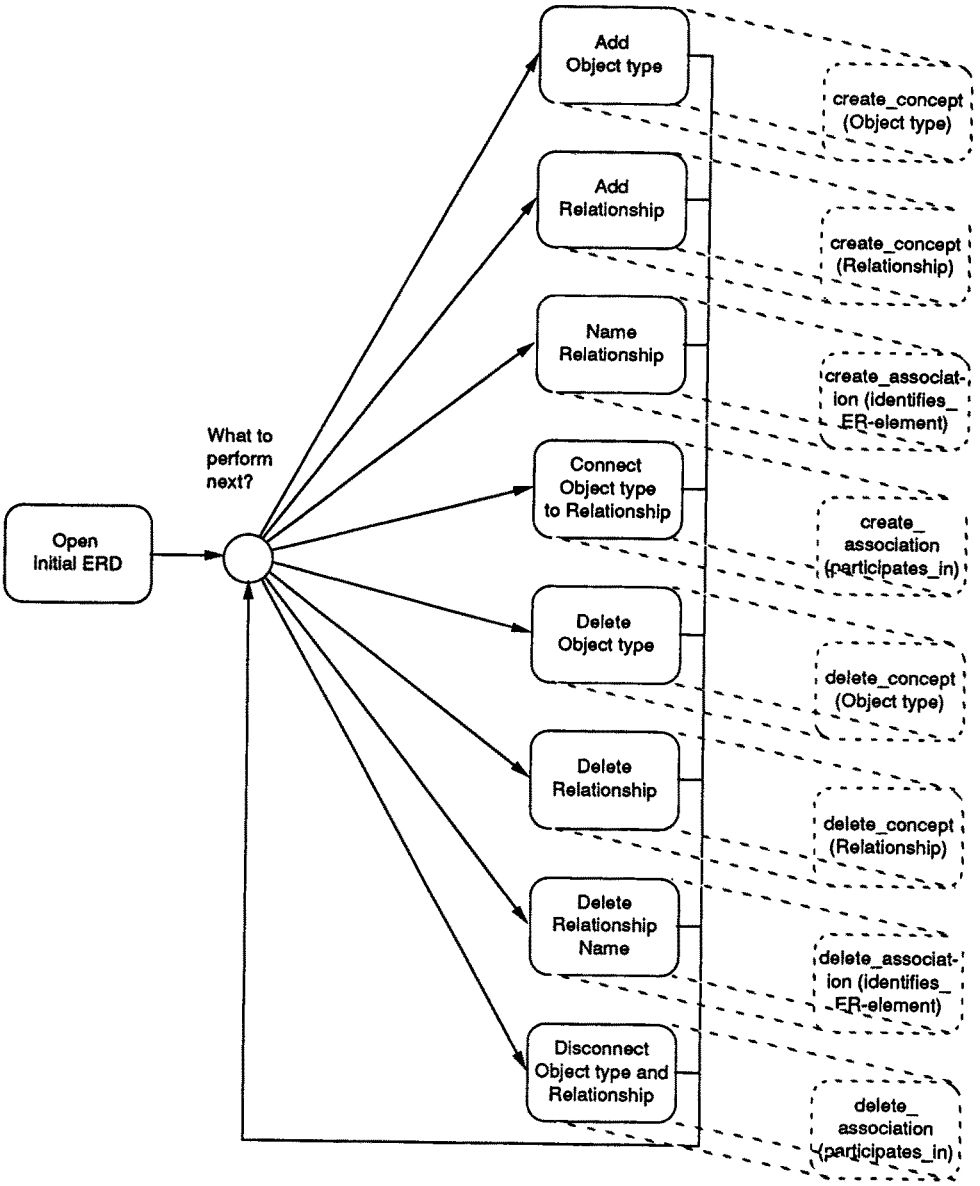

Figure 3.8 Decomposition of the leaf task 'Construct an initial ERD' into basic tasks

3.3.3 Decomposing leaf tasks to basic tasks

The next activity in the integration is establishing the decomposition of the leaf tasks to the level of basic tasks, i.e to the level of tasks that manipulate instances of concepts and associations in the task view. The leaf task 'Construct an initial ERD' can now be seen as an iteration of basic tasks of the type create_association, create_concept, delete_association and delete_concept, see figure 3.8. In figure 3.8 we have highlighted that each basic task with a *logical* name is mapped onto a formal basic procedure.

It is not necessary to add basic tasks for all concepts and associations in the task view determined. The creation of some of these concepts and associations is enforced by the a priori rules that hold for this leaf task. For instance, the rule 'All ER-elements are part of exactly one ERD' implies that the creation of instances of ER-elements enforces the creation of the association *is_part_of_ERD*. The same holds for the deletion of some of these concepts and associations.

It is important to notice that the task structure shown in figure 3.8 could not be derived from [Yourdon, 1989] but had to be designed in order to arrive at the level of basic tasks. We have strived for a flexible task structure in which all kinds of manipulations on the concepts of the task view could be performed.

3.3.4 Determining information places

Within the task 'Construct an initial ERD' each object type created has to be part of the specific initial ERD. This has to be enforced by the addition of an information place that is filled by the task 'Open initial ERD' and that is referred to by the task 'Add object type' as shown in figure 3.9. By this example we have illustrated the use of information places.

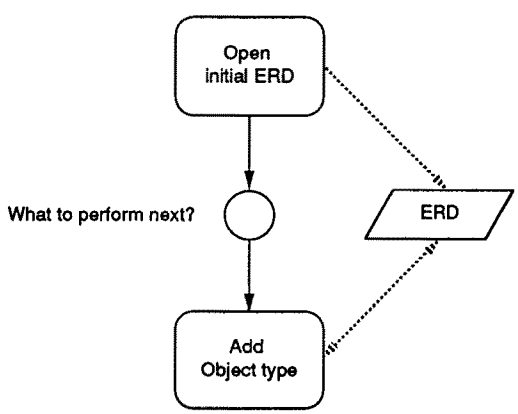

Figure 3.9 Information places in the task 'Construct an initial ERD'

4 Conclusions

In this paper, it has been stressed that for the improvement of automated support of information modelling processes it is necessary to focus both on modelling tasks and on the resulting models in the course of an information modelling process. The aim of the SOCRATES project is to realise this automated support by a CASE shell, which is method independent. By incorporating knowledge of a specific information modelling process, the shell is transformed into a workbench for that process. To enable this incorporation, a uniform representation technique is necessary. Such a technique has been referred to as a meta-modelling technique.

A meta-modelling technique is presented, which can be used to represent a way of modelling as well as a way of working observed in a method. The representation of a way of modelling has been called a concept structure. The analogon for a way of working has been called a task structure. Links between the way of modelling and the way of working have been established. The formal base of this technique is a suitable starting point for the implementation of a CASE shell.

The meta-modelling technique has been applied on Yourdon's Modern Structured Analysis. In the approach followed it became clear, that Yourdon's way of modelling had been described in far more detail than its way of working. In this sense, the assumption that more attention is paid to the way of modelling than to the way of working has been confirmed. A further decomposition of the way of working was necessary in order to be able to perform an integration between the way of modelling and the way of working.

By establishing the integration of the way of modelling and the way of working, Modern Structured Analysis has been structured. From the frequency in which delete procedures occur in the task structure it is apparent that a linear way of working, in which complete and correct models are delivered in one go, is more or less prescribed. Another observation is that much more attention has been paid to the construction of data flow diagrams than to the construction of entity-relationship diagrams and state-transition diagrams.

Future research in the SOCRATES project focuses on the representation of information modelling knowledge of experienced practitioners in the meta-modelling technique presented in this paper. Furthermore the SOCRATES CASE shell based upon the meta-modelling technique presented in this paper is currently being developed, see also [Schaapherder, 1990] and [Wijers et al., 1990b].

References

[Bergsten et al., 1989] Bergsten, P., J. Bubenko jr., R. Dahl, M.R. Gustafsson and L-Å. Johansson, *RAMATIC - a CASE shell for implementation of specific CASE tools*, SISU, Stockholm, 1989. First draft of a contribution to section 4.4 of the TEMPORA T6.1 report.

[Bommel et al., 1990] Bommel, P. van, A.H.M. ter Hofstede and Th.P. van der Weide, *Semantics and Verification of Object-Role Models*, Technical Report 90-13, Department of Information Systems, University of Nijmegen, June 1990 (submitted to Information Systems).

[Bots, 1989] Bots, P.W.G., *An Environment to Support Problem Solving*, Ph.D. Thesis, Delft University of Technology, Delft, the Netherlands, 1989.

[Chen et al., 1989] Chen, M. and J.F. Nunamaker jr., *MetaPlex: An integrated environment for organization and information systems development*, in *Proceedings of the 10th International Conference on Information Systems*, Boston, Massachusetts, 1989, pp. 141-151.

[Genrich, 1987] Genrich, H., *Predicate/Transition Nets*, in W. Brauer, W. Reisig and G. Rozenberg (Eds.), *Petri Nets: Central models and their properties*, L.N.C.S. nr. 1 254, Springer Verlag 1987, pp. 207-247.

[Hofstede et al., 1989] Hofstede, A.H.M. ter, T.F. Verhoef, S. Brinkkemper and G.M. Wijers, *Expert-based support of Information Modelling: A Survey*, Report RP/soc-89/7, SERC, Utrecht, the Netherlands, 1989.

[Hofstede et al., 1990] Hofstede, A.H.M. ter, T.F. Verhoef, G.M. Wijers and S. Brinkkemper, *The SOCRATES project*, in S. Brinkkemper and G.M. Wijers (Eds.), *Proceedings of the Workshop on the Next Generation of CASE-tools*, Noordwijkerhout, the Netherlands, April 1990.

[Jackson, 1983] Jackson, M.A., *System Development*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[Knuth et al., 1986] Knuth, E., J. Demetrovics and A. Hernadi, *Information System Design: On conceptual foundations*, in H.J. Kugler (Ed.), *Information Processing 86*, North-Holland, Amsterdam, the Netherlands, 1986, pp. 635-640.

[Lockemann et al., 1986] Lockemann, P.C. and H.C. Mayr, *Information system design: Techniques and software support*, in H.J. Kugler (Ed.), *Information Processing 86*, North-Holland, Amsterdam, the Netherlands, 1986, pp. 617-634.

[Martin, 1986a] Martin, J., *Information Engineering Volume 1: Introduction to Information Engineering*, Savant Research Studies, England, 1986.

[Martin, 1986b] Martin, J., *Information Engineering Volume 2: Methodologies for Strategy and Analysis*, Savant Research Studies, England, 1986.

[Martin, 1988] Martin, J., *Information Engineering Volume 4: Design and Implementation*, Savant Research Studies, England, 1988.

[Nijssen et al., 1989] Nijssen, G.M. and T.A. Halpin, *Conceptual Schema and Relational Database Design: A fact oriented approach*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[Potts, 1989] Potts, C., *A generic model for representing design methods*, in *Proceedings of the 11th International Conference on Software Engineering*, Pittsburgh, Pennsylvania, 1989, pp. 199-210.

[Schaapherder, 1990] Schaapherder, F.E.S., *Selecting a Software Development Environment for the SOCRATES Workbench Shell*, Internal Report, SERC, Utrecht, the Netherlands, October 1990.

[Seligmann et al., 1989] Seligmann, P.S., G.M. Wijers and H.G. Sol, *Analyzing the structure of I.S. methodologies, an alternative approach*, in *Proceedings of the First Dutch Conference on Information Systems*, Amersfoort, the Netherlands, 1989.

[Smolander et al., 1990] Smolander, K., P. Marttiin, K. Lyytinen and V. Tahvanainen, *MetaEdit - A Flexible Graphical Environment for Methodology Modelling*, in S. Brinkkemper and G.M. Wijers (Eds.), *Proceedings of the Workshop on the Next Generation of CASE-tools*, Noordwijkerhout, the Netherlands, April 1990.

[Troyer et al., 1988] Troyer, O. de, R. Meersman and P. Verlinden, *RIDL* on the CRIS case: A workbench for NIAM*, in T.W. Olle, A.A. Verrijn-Stuart and L. Bhabuta (Eds.), *Computerized Assistance during the Information Systems Life Cycle*, North Holland, Amsterdam, the Netherlands, 1988, pp. 375-459.

[Turner et al., 1987] Turner, W.S., R.P. Langerhorst, G.E. Hice, H.B. Eilers and A.A. Uijtenbroek, *System Development Methodology (SDM II)*, North-Holland and Pandata, 1987.

[Verhoef, 1991] Verhoef, T.F., *Structuring Yourdon's Modern Structured Analysis*, Technical Report, SERC, Utrecht, the Netherlands, March 1991 (to appear).

[Welke, 1988] Welke, R.J., *The CASE Repository: More than another database application*, in *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.

[Wijers et al., 1990a] Wijers, G.M. and H. Heijes, *Automated Support of the Modelling Process: A view based on experiments with expert information engineers*, in B. Steinholtz, A. Sølvberg and L. Bergman (Eds.), *Advanced Information Systems Engineering*, L.N.C.S. 436, Springer Verlag, Berlin, Germany, 1990, pp. 88-108.

[Wijers et al., 1990b] Wijers, G.M., A.H.M. ter Hofstede and S. Brinkkemper, *Flexible Guidance of the Design Process*, Technical Report 90-07, SERC, Utrecht, the Netherlands, November 1990.

[Wijers et al., 1990c] Wijers, G.M., A.H.M. ter Hofstede and N.E. van Oosterom, *Representation of Information Modelling Knowledge*, Report 90/09, SERC, Utrecht, the Netherlands, November 1990.

[Wintraecken, 1985] Wintraecken, J.J.V.R., *Informatie-analyse volgens NIAM*, Academic Service, Den Haag, the Netherlands, 1985 (in Dutch).

[Yourdon, 1989] Yourdon, E., *Modern Structured Analysis*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.