The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/28319

Please be advised that this information was generated on 2019-12-21 and may be subject to change.
Meta-CASE: Is the game worth the candle?

A.H.M. ter Hofstede and T.F. Verhoef

Abstract. The current generation of CASE (Computer Aided Software Engineering) tools is too inflexible to provide adequate modelling support. One of the proposed solutions to this problem is the development of so-called CASE shells. A CASE shell is a method independent CASE tool, which may be instantiated with a specific method to become a CASE tool supporting that method. As such, a CASE shell provides complete flexibility. This paper does not address the benefits of CASE shells, as they are completely clear, but focuses on the feasibility of this concept from a theoretical as well as a practical point of view.

Keywords: CASE shell, computer-aided software engineering (CASE), early phases, meta-modelling.
development are referred to as upper CASE tools. Upper CASE tools tend to be strongly product oriented, as confirmed by the study of Wijers & Dort (1990) and more recently Kusters & Wijers (1992). These studies among Dutch users of CASE tools concluded that these tools are mainly used for documentation and verification purposes. After a model had been constructed, it was specified by the use of an automated tool, and then verified. It is suspected that this limited usage is largely due to the design of the tools not paying appropriate attention to the information engineer's modelling needs.

Undoubtedly automated tools support consistency in the development process, but Bubenko and Floyd identify the danger of computerizing unsuitable methods (Bubenko, 1986; Floyd, 1986). Guidelines on why and how to perform various tasks (i.e. how to perform the modelling process), and how to determine the specification's quality, are not part of automated tools. Naturally, the range of the required facilities should be thoroughly understood before using automated tools, see also Benyon & Skidmore (1987). In this reference an environment (automated or not) is proposed which should support the practising information engineer in the choice of suitable techniques, depending on characteristics of the modelling problem. CASE tools, however, have the problem that the view of the information systems development life cycle to be supported has been hard-coded in these tools, and therefore cannot be changed or customized to also include knowledge that is based upon information engineers' practical experience. By consequence, information engineers are left with the problem of finding a way of applying these rigid and inflexible tools in their information engineering practice.

In this paper, the term 'flexibility' refers to the extent to which practising information engineers are able to adapt a tool to their working style. In Vessey et al. (1992), three philosophies for method support are distinguished: restrictive, guided, and flexible. The meaning of these philosophies is introduced in an informal way, by using analogous terms such as 'enforcement', 'encouragement', and 'complete freedom'. As such, the approach of Vessey et al. (1992) corresponds to a strict categorization of levels of flexibility. As flexibility corresponds to a continuous spectrum of levels of adaptability, this paper will not focus on these three particular values of flexibility.

From the mid-1980s onwards, research has focused on the problem of developing flexible CASE tools. It is claimed that automated tools are preferably built according to a CASE shell architecture. Such an architecture allows for the modification and extension of the tool's behaviour as the tool includes explicit and adaptable method knowledge. As a consequence, information engineers are able to adapt support tools to their working styles instead of the other way around. Crucial for the development of a CASE shell is the availability of a suitable and formally defined technique for the representation of method knowledge. Such a technique is referred to as a meta-modelling technique. Method knowledge represented in a CASE shell according to such a technique is called a meta-model.

The concept of a CASE shell is not new. Commercial products such as Toolbuilder of IPSYS Software (IPSYS Software plc, Macclesfield, England), Methods Workbench of IDSE Metasoft Ltd (IDSE Metasoft Ltd, Camberley, England) and MetaDesign of Meta Software Corporation (Meta Software Corporation, Cambridge, Massachusetts, USA) or academic products such as RAMATIC (Bergsten et al., 1989), Metaview (Sorenson et al., 1988) and MetaPlex (Chen & Nunamaker Jr, 1989), claim to generate CASE tools tailored to specific methods and organi-
zations. Even a tool that supports the modification of meta-models exists — MetaEdit (Meta-
Case Consulting, Jyväskylä, Finland (Smolander et al., 1991)). However, all these shells focus
on the support of modelling techniques and pay hardly any attention to the modelling process —
the importance of which is stressed in Knuth et al. (1986); Lockemann & Mayr (1986); Potts
(1989) and Wijers & Heijes (1990) among others. Furthermore, the degree of support of
modelling techniques which they offer is limited, due to the low expressive power of the meta-
modelling techniques used.

The focus of this paper is on the feasibility of flexible support of information modelling in the
early phases and as such on the feasibility of CASE shells. Flexible support is of course con-
sidered to be feasible if the benefits outweigh the realization effort. As the benefits are clear, this
feasibility study focuses on the effort needed to realize adequate flexible support. To acquire
insight into the effort needed, it is necessary to know the complexity of information engineers'
modelling knowledge used in the early phases. Hence, three fundamental research questions
need to be addressed: (1) which dimensions do exist within modelling knowledge? (2) how
complex are these modelling dimensions? and (3) what is the diversity needed in these mod-
elling dimensions to support individual information engineers?

In section 2, focus is on the various aspects of information modelling and their relations,
essential for flexible support. This section addresses the first research question.

With respect to the second research question, it can be remarked that the early phases of
systems development are still poorly understood (Guindon & Curtis, 1988). Activities in these
stages are characterized by incompleteness and vagueness (Belady, 1985). Terminology is
often fuzzy and not standardized. Therefore, a prerequisite for dealing with the first research
question is a language in which information modelling concepts can be adequately expressed,
i.e. an adequate meta-modelling technique. State-of-the-art meta-modelling techniques, as
described in Brinkkemper (1990); Smolander et al. (1991); Araujo & Carapuça (1992) and Heym
& Österle (1992) are not fully suited for this purpose. They do not have sufficient expressive
power to capture information modelling concepts and relations between these concepts, and
tend to neglect the modelling process. In addition to that, they rarely have a formal semantics
and, therefore, only tend to add to the current confusion with respect to information modelling
(Hofstede & Welde, 1992). In section 3, techniques are described that are capable of formally
describing the aspects described in section 2. Focus in this section is not on the techniques as
such, but rather on the inherent complexity of an adequate meta-modelling technique.

With respect to the third research question, it can be remarked that relatively little is known
about the diversity of information modelling processes in practice and the corresponding degree
of flexibility needed for adequate support. Empirical studies reported in the literature (Ballay,
1987; Guindon, 1990a; Wijers, 1991; Bansler & Bødker, 1993) show that information modelling
knowledge as applied by experienced information engineers turns out to deviate from modelling
knowledge described in textbooks, regarding both modelling concepts and the way models
using these concepts are constructed. These studies served as a starting point for the
experiments described in section 4, which focused more closely on the precise behaviour of
information modelling experts and the degree of flexibility needed for adequate support.

In section 5 the findings of the previous sections are summarized and the feasibility of
adequate flexible support of information modelling processes in the early phases is addressed. The question arises whether the concept of a CASE shell is a realistic goal, given the inherent complexity of an adequate meta-modelling technique and the desired degree of flexibility. In other words: is the game worth the candle?

A VIEW ON INFORMATION MODELLING

Information modelling processes can be looked upon from many different perspectives, depending on the underlying goal. From a management point of view, resources, deadlines, and quality requirements are important. From a collaboration perspective, focus will be on communication between individuals in groups. Given our goal, the investigation of the feasibility of flexible support, knowledge about the information modelling processes to be supported is important. This knowledge has to be reflected in flexible information modelling support environments. Therefore, this section addresses our view on information modelling by exploring the structure of the repository of a CASE shell.

Essentially, three orthogonal dimensions are recognized within the repository. In its most elementary form, the structure of the repository of a CASE shell can thus be represented as a $2 \times 2 \times 2$ cube (see Fig. 1). These dimensions are subsequently discussed.

The first dichotomy is that of method level versus application level, also referred to as types versus instances. The method level is concerned with knowledge which may be used by information engineers. The method level controls the ways how information modelling processes may be performed, and defines which products may result from those information modelling processes. The application level is concerned with information which results from projects for specific organizations and applications by a specific group of information engineers. The application level is an instantiation of the method level. For example, (the use of) the data flow diagramming technique as such is part of the method level, the development of a specific data flow diagram in a specific project setting is part of the application level.

The second dimension is that of process versus product: in order to provide information modelling support, it is necessary to have knowledge about the (intermediary) products and

![Figure 1. Information modelling dimensions.](image)

their relationships on the one hand, and about the underlying modelling process on the other. In
other words, both questions 'what should be produced?' and 'how should it be produced?'
should be answered. In Wijers (1991), the process side is referred to as the way of working,
while the product side is referred to as the way of modelling.

Knowledge about information modelling processes is structured by several key concepts. It is
necessary to know which tasks may be performed as part of an information modelling process.
Tasks can be large tasks: 'Perform the Business Area Analysis' within the Information Engi­
neering method, and can be minor tasks: 'Add a total role constraint to an Information Structure
Diagram' within the NIAM (Natural language based Information Analysis Method) method
(Nijssen & Halpin, 1989; Halpin & Oriowska, 1992). These examples show that decomposition is
a key concept too: tasks may be decomposed into subtasks. Knowledge about information
modelling processes also concerns the flow of control: which tasks may be performed next?

Knowledge about information modelling products shows the structure of, and the relation­
ships between, information modelling products. Examples of information modelling products are
a 'list of requirements', a 'Create Read Use Delete (CRUD) matrix', a 'cardinality constraint', and
an 'organization hierarchy'. Examples of structure and relationships: 'attribute types belong to
entity types' and 'organization hierarchies consist of organization units'.

It should be clear that this dichotomy of product versus process is not a dichotomy of strict
separation. On the contrary, knowledge about information modelling processes and knowledge
about information modelling products are strongly interwoven. For example, the product 'Entity
Relationship Diagram' has to satisfy more and more constraints as the information modelling
process proceeds. In early stages, a draft version of an ERD (Entity Relationship Diagram)
suffices, containing only Entity types and Relationship types. In later stages, however, strong
syntactic rules apply. For example, all Entity types should be related to one another, and
Attribute types should be added. To illustrate the relationship between product oriented
knowledge and process oriented knowledge even more: each task which is part of the modelling
process should lead to a change in some modelling product.

This completes the discussion of the second dimension. It should be clear that the two
dimensions discussed so far are orthogonal: both knowledge about information modelling
processes and knowledge about information modelling products exist at method level and at
application level. To clarify this, it may be specified, at method level, that the following tasks are
to be performed: (i) 'Select manager for interview session', (ii) 'Interview manager', and as a
result (iii) 'Refine organization model'. These three tasks may be succeeded by the decision (iv)
'Is the organization model at the desired level of detail?', which triggers task (i) if the outcome is
negative, and which leads to continuation if the outcome is positive. Correspondingly, execution
of these tasks in a specific project, at application level, may lead to dozens of specific interviews
and specific model refinements. Analogously, a notion such as 'entity type' on the method level,
may lead to many instances on the application level, e.g. 'Customer' and 'Article'.

The third dichotomy concerns the difference between conceptual and graphical knowledge.
Evidently, models must be represented in one way or another: diagrams, matrices, tables, lists,
and programme specifications are examples. A clear distinction should be made between the
modelling concepts and their external notation. In Sutcliffe et al. (1989) It is argued that some
methods allow alternative equivalent notations for one and the same modelling concept, but that on the other hand similar graphical and textual topologies can represent different types of modelling concepts.

A similar argument is valid for the process oriented view on information modelling. If one looks at some of the commercial available CASE tools, one observes different ways of model manipulation, for example, how entities can be created in entity relationship diagrams. In ADW (Sterling Software, Dallas, Texas, USA) one action within the ERD window suffices to create an entity. In Excelerator (INTERSOLV, Rockville, Maryland, USA) a menu selection has to be performed first, after which one can point at the location preferred.

This third distinction is particularly important for CASE shells. In some way or another, it has to be specified how models appear on the screen and how actions can be performed on these represented models. Furthermore, the specification of graphical knowledge allows information engineers to change the user interface of tools to their own preferences.

Again, it should be clear that this third axis is orthogonal in relation to the two previous ones. Both knowledge about information modelling processes and information modelling products have graphical counterparts. Modelling concepts such as data flows and organization units are related to graphical notions such as arrows and boxes. Conceptual tasks such as additions of model components lead to graphical interaction patterns such as menu selections, object clicking and dragging, and so on.

COMPLEXITY OF META-MODELLING

A meta-modelling technique is a technique in which modelling knowledge can be expressed. As such, a meta-modelling technique should at least be capable of capturing the various perspectives on information modelling as described in the previous section. This implies that a meta-modelling technique should have sufficient expressive power. There are, however, other requirements that meta-modelling techniques have to fulfil.

As a meta-model should not be ambiguous, a meta-modelling technique should be formally defined — both syntax and semantics (see also Hofstede & Weide, 1992). It has to abstract from implementation details. Meta-models often need to be validated with modelling experts whose modelling knowledge is going to be captured by those meta-models, therefore a meta-modelling technique should support the construction of comprehensible meta-models (e.g. offer graphical representations, decomposition mechanisms etc.). Finally, as a CASE shell needs to be instantiated with a particular meta-model to become a concrete CASE tool, meta-models should be executable.

In this section, (partial) meta-modelling techniques for the various perspectives on information modelling are outlined. As stated before, the goal of this section is to stress the inherent complexity of meta-modelling rather than to provide an in-depth treatment of the various techniques. This section reflects the view on information modelling presented in the previous section. Section 3.1 concerns the representation of product oriented knowledge, section 3.2 concerns the representation of process oriented knowledge. Both these sections are restricted to conceptual knowledge. Section 3.3 deals with graphical knowledge.
Representing a way of modelling

Modelling techniques in general contain concepts with complex structures and their models usually have to satisfy quite complex rules. To capture these structures and rules, a powerful data modelling technique is required, together with a powerful constraint modelling technique.

In this section the data modelling technique Predictor Set Model (PSM) and the constraint modelling language LISA-D (Language for Information Structure and Access Descriptions) are highlighted. The PSM has been specifically defined with the representation of complex structures, often needed for meta-modelling, in mind. PSM is defined in Hofstede & Weide (1993), and LISA-D in Hofstede et al. (1993).

Firstly, the elementary notion of object type, relationship, and role are addressed. Subsequently, the necessity of complex objects and object inheritance is illustrated by several meta-modelling problems. Finally, attention is paid to constraints to represent complex rules in product knowledge.

Object types, relationship types and roles

One of the key concepts in data modelling is the concept of relationship type. In Entity Relationship modelling (ER) and NIAM a relationship type is considered to be an association between object types. In Fig. 2 the graphical representation of a binary relationship type ‘Identification’ between object types ‘Entity Type’ and ‘Attribute’ is shown in the NIAM style, while in Fig. 3 the corresponding ER diagram is depicted. A relationship type consists of a number of roles, which denote the way object types participate in that relationship type. In the example, ‘Identification’ has role names ‘identifies’ and ‘is identified by’.

In PSM a relationship type is considered to be a set of roles. A relationship type may be treated as an object type (objectification), and can therefore play a role in other relationship types.

Object composition

Knowledge about information modelling products can be characterized as structured in a complex way. For example, the information modelling product ‘entity-relationship diagram’
consists of a large variety of information model components. To describe these composition relationships between modelling concepts, PSM offers three representation mechanisms for object composition: set types, sequence types, and schema types.

An instance of a set type is a set of instances of its element type. As a simple example of the use of set types in the context of meta-modelling, consider the total role constraint in NIAM. An example of such a constraint is depicted in Fig. 4.

Figure 4. A sample total role constraint in NIAM.

In this figure the total role constraint, represented by the circle with the black dot, requires every instance of entity type A to participate in at least one of the roles p, q and r. Syntactically, a total role constraint is nothing more (or less) than a set of roles. Total role constraints have no other identification than their constituting roles. In a meta-model of NIAM, the total role constraint should therefore be modelled as a set type having an object type 'Role' as its element type (see Fig. 5).

Figure 5. An example of a set type in the context of a meta-model of NIAM.

Sequence types can be compared to set types. The differences are that, in the case of sequence types, the ordering of elements is important and elements may occur more than once. An instance of a sequence type is a sequence (tuple) of instances of its element type.

As an example of a sequence type in the context of meta-modelling consider the Jackson Structured Design (JSD) entity structure diagrams (see Fig. 6). These diagrams allow for the representation of action iterations (graphically represented by an asterisk), choices between actions (graphically represented by a tiny circle), and sequences of actions. To represent this JSD product knowledge in a meta-model, the concept of sequence type is necessary. Fig. 7 represents part of a meta-model of JSD entity structure diagrams. This meta-model captures the fact that an action can be decomposed into a sequence of other actions, that it can be a repetition of another action and that it can be a choice between a number of actions.

The third and most complex representation mechanism within PSM is schema objectification.
Schema objectification allows to define part of a schema as an object type (referred to as schema type). Schema objectification can thus be seen as a decomposition mechanism. An instance of a schema type is an instantiation of the associated schema part. As an example of a schema type, consider the meta-model of Activity graphs as shown in Fig. 8.

Activity graphs are bipartite directed graphs consisting of activities and states. States can be input and output for activities and can be compared to flows in Data Flow Diagrams (DFDs). Both states and activities can be decomposed into other activity graphs. Figure 8 shows the use of the concept of Schema type to represent the meta-model of Activity graphs. ‘Activity graph’ is a schema type, the decomposition relation is reflected by the binary relationships to ‘Activity’ and ‘State’.

Scheme types are particularly important for meta-modelling as they allow for a natural representation of decomposition constructs in modelling techniques. The Importance of schema types has also been stressed by Welke (1988) (although the term window type is used).

**Object inheritance**

PSM offers two representation mechanisms for the representation of inheritance of properties between modelling concepts: specialization and generalization.
Specialization, also referred to as subtyping, is a mechanism for representing one or more (possibly overlapping) subtypes of an object type. Intuitively a specialization relation between a subtype and a supertype implies that the instances of the subtype are also instances of the supertype.

Specialization relations are organized in so-called specialization 'hierarchies'. The top of a specialization hierarchy is referred to as the *pater familias*. Identification of subtypes is derived from their supertypes, as object types inherit all properties from their ancestors in the specialization hierarchy.

Figure 9 shows a specialization hierarchy. Each specialization relation is represented as an arrow. As a consequence, the *pater familias* of e.g. object type 'product change' is 'objective'.

Specialization is useful in the context of meta-modelling as it allows the definition of specific subsets of instances of certain object types for which only specific relations are important. In

---

**Figure 8.** An example of a schema type in the context of a meta-model of activity graphs.

**Figure 9.** Example of a specialization hierarchy.

terms of the previous example, it is possible that for 'financial control objectives' specific relationships are relevant, which are irrelevant for other types of 'objectives'.

Generalization is a mechanism that allows for the creation of new object types by uniting existing object types. For generalization it is typically required that the generalized object type is covered by its constituent object types (or specifiers). Furthermore, properties are inherited 'upward' in a generalization hierarchy instead of 'downward', which is the case for specialization. This also implies that the identification of a generalized object type depends on the identification of its specifiers.

In Fig. 10 an example of generalization is shown. The dashed arrows indicate that the object type 'Formula' is a generalization of the object type 'Variable' and the relationship type 'f'. Therefore, a formula may be either a single variable, or constructed by some function (say f) from simpler formulas.

![Figure 10. Example of generalization.](image)

Generalization is essential for meta-modelling as it provides the only way to construct recursive types (such as 'Formula' in the previous example). Recursive types frequently occur when modelling documents (such as reports in the context of meta-modelling).

**Constraints**

PSM offers a number of graphical constraint types for the representation of rules which hold for modelling products. Examples can be found in the meta-model of Yourdon DFDs (data flow diagrams) presented in Fig. 11. This meta-model also demonstrates the need for the many type construction mechanisms in PSM. First the DFD concepts which appear in this meta-model are clarified.

According to Yourdon (1989), a DFD pictures a system as a network of functional processes. The main components of a DFD are processes, flows, data stores and terminators. A process transforms input into output. Processes have a process specification or are decomposed into a DFD. Each process has a number. Control processes are a special kind of process. A control process does not process data, but coordinates other processes. The operation of a control process is modelled by means of a state transition diagram. Terminators represent external processes communicating with the system under consideration. Data stores model collections...
of data 'at rest'. Data stores may be external, which means that they are used for communication with the outside world.

Flows represent data 'in motion'. Several types of flows exist. A simple flow has a source and a destination. Processes, data stores and terminators can be source or destination of simple flows. A complex flow consists of a set of flows converging to one other flow or a flow diverging into a set of other flows. Control flows represent triggers, i.e. signals or interrupts.

Some of the graphical constraints in this meta-model deserve some further explanation. Only some of the graphical constraints in Fig. 11 will be explained. The total role constraint on the role named 'has' attached to the object type 'DFD-Element' and represented by a black dot, expresses that each instance of this object type has to have a 'name'. The two exclusion constraints attached to binary relationship types and represented by encircled crosses express that the source and the destination of a 'Data-Flow' are different and that the source and the
destination of a 'Control-Flow' are different. The two uniqueness constraints each over two relationship types and represented by an encircled 'u' express that no two 'Data-Flows' with the same 'Name' have the same 'DFD-Object' as destination; and that no two 'Data-Flows' with the same 'Name' have the same 'DFD-Object' as source. The occurrence frequency constraint on the role with role name 'relates-to-lower-level' and represented by the encircled text '1..2' expresses that a 'Data-Flow' is related to at most two other 'Data-Flows' on a lower decomposition level. The exclusion constraints attached to the set type 'Flow-Group' states that a 'Data-Flow' does not occur in more than one 'Flow-Group'.

Of course, there are many other constraints that have to be fulfilled. These constraints, however, are too complex to be expressed using the graphical constraint types offered by PSM. It should even be noticed that aiming at expressing the most complex constraints graphically might decrease the comprehensibility of the meta-model under consideration. Figure 11 provides a good example of a meta-model which cannot be grasped at once, even in spite of the fact that only a minority of the constraints applicable are represented graphically.

The language LISA-D has been introduced for the representation of constraints that cannot be graphically expressed in PSM. The fact that a 'Data-Flow' should not have a 'Terminator' as a source and as a destination can be formulated in LISA-D as follows:

\[
\text{NO Terminator produces DATA-Flow is-input-for Terminator}
\]

LISA-D expressions exploit the natural language basis of PSM to improve comprehensibility. They use names defined in the associated PSM schema as well as a number of predefined keywords for the description of meaningful connections. This example shows the use of the predefined keyword **NO** in connection with some object names and role names.

In meta-models, complex constraints often occur. In the DFD meta-model for example, recursive decomposition of processes is not allowed. LISA-D offers powerful constructions for the expression of complex constraints. The recursive decomposition requirement can be formulated as follows:

\[
\text{NO Object-type}
\]
\[
\text{ANY REPETITION OF (is-supertype-of)}
\]
\[
\text{THAT Object-type}
\]

**Representing a way of working**

This section addresses several constructs for the representation of a way of working. As stated in section 2, a way of modelling and a way of working are closely related. Therefore, attention is also paid to the representation of relationships between a way of modelling and a way of working.

To represent knowledge about information modelling processes adequately, constructs are needed that allow for the description of moments of choice, sequence, parallelism, synchronisation, and iteration. Task structures, formally defined in Hofstede & Nieuwland (1993),
contain constructs for expressing these task dependencies. In Fig. 12, the main concepts of task structures are graphically represented. They are discussed subsequently.

The central notion in task structures is the notion of a task. A task is defined as something that has to be performed in order to achieve a certain goal: the realization of (part of) some information modelling product. A task can be defined in terms of other tasks, referred to as its subtasks. This decomposition may be performed repeatedly until a desired level of detail has been reached. Tasks with the same name have the same decomposition, e.g. the tasks named B in Fig. 12. Performing a task may involve choices between subtasks, decisions represent these moments of choice. Decisions coordinate the execution of tasks. Two kinds of decisions are distinguished; terminating and non-terminating decisions. A decision that is terminating, may lead to termination of the execution path of that decision. If this execution path is the only active execution path of the supertask, the supertask terminates as well.

Triggers, graphically represented as arrows, model sequential order. In Fig. 12 the task with name G can start after termination of the top task named B. Initial items are those tasks or decisions, that have to be performed first as part of the execution of a task that has a decomposition. Due to iterative structures, it may not always be clear which task objects are initial. Therefore, this has to be indicated explicitly. Finally, synchronisers deal with explicit

Figure 12. Graphical representation of task structure concepts.
synchronisation. In Fig. 12 the task named H can only start when the tasks with names C and G have terminated.

As a concrete example of a task structure, consider Fig. 13. This task structure models the overall way of working of the Yourdon method (Yourdon, 1989). This structure is self-explanatory. The decomposition of the task 'Perform database conversion if necessary' is shown in Fig. 14. From this decomposition, it follows that a database conversion only has to be performed if a current database exists.

As remarked in section 2, knowledge about information modelling processes and knowledge about information modelling products are highly interwoven. Tasks, for example, may change products and pass intermediate results. Decisions, on the other hand, may be influenced by these intermediate results and product changes.

As techniques for a way of modelling can be considered to be data modelling techniques and techniques for a way of working to be process modelling techniques, their integration poses identical problems as the integration of data and process modelling technique does.

In the context of meta-modelling, at least constructs for elementary model changes, for

![Diagram](image-url)

Figure 13. Example of a task structure.

passing of intermediate results, for specification of pre- and postconditions, and for decision rules are needed. An example of an elementary model (or product) change would be the addition of a process to a DFD (which would amount to the creation of an instance of the entity type 'Process' in the DFD meta-model of Fig. 11).

In Fig. 15 an example of information passing in the context of meta-modelling is shown. In this figure, the task 'Select control process to be decomposed' selects a Control process that has to be passed on to the task 'Decompose control process'. The buffer with name 'Control process to be decomposed' contains the control processes in the order in which they are produced by the selection task. A buffer contains an ordered sequence of values and is necessary in this meta-model if both tasks depicted could be performed concurrently. A local variable, which can only

contain one value at a time known to each task in its associated decomposition, can be used if these tasks can only be performed sequentially.

Constraints specified in LISA-D can be used as pre- or postcondition or decision rule. As remarked in section 2, information modelling products have to satisfy different requirements in different stages of the modelling process. This can be formally specified by the use of pre- and postconditions. Decision rules can be applied to support decisions that are of a formal nature, e.g. 'Have all control processes been decomposed?'. The decision 'Does a current database exist?', shown in Fig. 14, obviously cannot be formally supported.

**Representing graphical knowledge**

So far, focus has been solely on the conceptual aspects of meta-modelling. Which modelling notions are important, how are they interrelated and under which circumstances (and how) may they be changed? To support information modelling processes adequately, this knowledge does not suffice, see, again, the view on information modelling in section 2. It is also necessary to be able to capture representations of modelling concepts. Consider for example the meta-model of DFDs as presented in Fig. 11. This meta-model does not capture the fact that a process is represented as a circle and a terminator as a rectangle. Therefore, a mapping of conceptual notions to corresponding graphical representations is required (see Fig. 16).

The mapping between conceptual notions and graphical representations can be partial, not

<table>
<thead>
<tr>
<th>Name</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>decimal string</td>
</tr>
<tr>
<td>Terminator</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>Data store</td>
<td></td>
</tr>
<tr>
<td>Control flow</td>
<td></td>
</tr>
<tr>
<td>Data flow</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 16.** Mapping of conceptual notions to corresponding graphical representations.
every conceptual notion has to have a graphical counterpart. Furthermore, instances of the same conceptual notion can have a different graphical representation depending on specific conditions. A ‘Data-Flow’ for example has to be represented as a line if it is an outgoing flow (formally, if it plays the role ‘is-output-of’) and has to be represented as an arrow if it is an incoming flow (formally, if it plays the role ‘is-input-for’), see also Fig. 17. At the same time, one graphical representation may be used for the representation of several conceptual object types. Within the systems development method SADT, the arrow is an illustrative example, as it may represent input flows, output flows, control flows, and so-called resource flows.

Figure 17. Representation of diverging and converging flows.

In addition to the aforementioned mapping, it is also necessary to be able to impose additional constraints on graphical representations. In DFDs for example, the number of a process always appears in the circle representing that process. In Hofstede et al. (1992a) and Hofstede et al. (1992b), a technique is described in which these kinds of constraints can be formally expressed. The technique uses LISA-D information descriptors to relate instances on which specific requirements have to be imposed. A number of constraint types can subsequently be applied to these information descriptors. The aforementioned requirement for example can be formulated as:

\[ \text{inside(Process WITH Number)} \]

Definition of graphical constraints is a complex matter. The above example specifies a constraint which should hold for all specific cases, therefore this example is, in terms of the $2 \times 3 \times 2$ cube of Fig. 1, on the method level. However, to enforce this constraint, knowledge is needed about the precise positions of all graphical object instances which represent the instances of the conceptual object types ‘Process’ and ‘Number’. This example shows that it is necessary to have knowledge about a large number of graphical properties of graphical object instances, such as their size, their position, and their angle.

To dive once more in this river of complexity, definition of graphical constraints sometimes requires the use of so-called handles which identify certain specific parts of a graphical object. The SADT example may again serve as an illustration. In SADT, the meaning of an arrow head depends on whether it is connected to the top or the bottom of a box. In the former case, the arrow head represents a control flow, in the latter case it represents a so-called mechanism. Therefore, to connect a control flow to a box, only the top of the box may be used. The top of a box is therefore considered to be a different handle than the bottom of the box.

In the context of DFDs, consider the representation of diverging and converging flows (see Fig. 17). To be able to connect the various parts of these flows properly, handle names are necessary. In Fig. 18 some handle names of flows are depicted.

The following constraint enforces that incoming flows of a diverging flow all start at the end of the outgoing flow (see also Fig. 19).
An area which, to our knowledge, has never been addressed in the context of meta-modelling, is the dynamic side of the representation of graphical knowledge. The representation of graphical actions seems to be completely ignored. This issue is quite complex as it requires an adequate integration with static representation aspects of modelling concepts, and with the underlying conceptual process oriented view.

For example, consider the following sequences of graphical actions:

1. selection of menu option ‘Delete’
2. selection of process ‘Collect payments’.

These actions lead to the removal of the graphical representation of this specific process, they might lead to the conceptual removal of the process itself, and they might even lead to the removal of the connected data flows (and their graphical representations), depending on the constraints specified. It should be remarked that not every graphical action has consequences for the conceptual view. Consider for example an action in which process representations can be enlarged or reduced, an action which changes the position of a process representation, or an action which selects a menu option.

At the same time, the term ‘flavour’ has been deliberately used. To offer one coherent toolkit for an adequate representation of modelling knowledge, even more representation mechanisms are required, see Hofstede (1993). This reference also offers the complete formal definitions for the representation mechanisms within the conceptual part of the cube. The formal definitions for the representation mechanisms within the graphical product oriented part of the cube are given in Hofstede et al. (1992a) and Hofstede et al. (1992b).
This section focuses on the second research question which is dealt with in this paper. Since this paper deals with flexible support of information modelling processes, it is, of course, necessary to know how much flexibility is needed, in other words, to know in which manner and to which extent information modelling processes deviate from each other. As stated before, little attention has been paid in the literature to differences and similarities between information modelling processes. This section discusses the approach towards getting insight in information modelling processes in practice, and discusses the insight gained. For a detailed discussion of both the approach and the results, see Verhoef (1993).

Approach

To achieve insight in individual information modelling processes, the behaviour of several information engineers in practice has been observed. To structure this observation process, an approach has been developed for the acquisition of information modelling knowledge which is based upon several starting points. These starting points arise from the observation that many (situational) factors may influence the course of information modelling processes. The three main factors are: the target domain (including its nature, its complexity, and the users participating in the information modelling processes), the information engineers themselves (including their educational background, their cognitive style, and their level of expertise), and the methods and the techniques used by the information engineers. Therefore, the approach has been determined by the need to find a balance between two conflicting requirements: (1) to control these situational factors as much as possible, so that the insight gained has a generic nature, and (2) to observe information modelling processes which are performed in a natural rather than in a laboratory environment, so that the insight gained has a realistic nature. The approach has, furthermore, been determined by the need to have a point of reference. Henceforth, information modelling knowledge has not only been represented as-it-is applied in practice, but also as-it-should-be applied, according to the underlying IS development method used. A third decision in determining the approach has been to observe experienced information engineers. This decision has been influenced by the fact that experienced information engineers may deviate from the modelling knowledge prescribed by the underlying IS development method to a larger extent than novice information engineers. Therefore, involvement of experienced information engineers might provide more clues to the nature of deviations than involvement of novice information engineers. This well-known fact stems from the field of expert systems. Given these starting points, we discuss the choice of an IS development method, and the setting of the observatory experiments in practice below.

For the choice of an IS development method, three criteria apply:

(i) the method should support the early stages of the development life cycle
The literature on the method should be available (to represent prescribed modelling knowledge according to a reference book).

i) the method should be applied widely (to represent applied modelling knowledge).

These three criteria supported the choice of the Yourdon method (Yourdon, 1989).

The **modelling knowledge acquisition approach** aims at acquiring a detailed understanding, in particular of applied modelling knowledge. This approach consists of four tasks: preparation, elicitation, conceptualisation, and interpretation, see Fig. 20. The first task concerns technical and organisational preparations, and a preparation of the experienced practitioners involved to a modelling task to perform. The elicitation of modelling knowledge takes place when the expert information engineer is performing a modelling task in the context of a real-life case. During the elicitation session, the expert is encouraged to think aloud. Communication processes between the expert and the users are restricted to personal computers for verbal communication and video for exchanging diagrams, see Fig. 21. The elicitation task results in a protocol transcript, which contains all verbal data and diagrams. Specific textual fragments in the protocol transcript are marked during the interpretation task, e.g. as 'decision', 'modelling concept', or 'modelling task'. Finally, the resulting text-based model is transformed into a metamodel during the conceptualisation task. This approach has been developed for a more general

---

![Diagram](image-url)  
*Fig. 20. Approach to acquisition of modelling knowledge.*

This observation study has involved three expert information engineers. Selection of these experts has taken place according to a number of requirements. The experts should have more than five years of professional experience in the field of information modelling, and be proficient at the Yourdon approach. Their managers should consider them to be experienced and competent, and they should also be experienced in a variety of application domains.

Each of them have performed two modelling tasks with a duration of 4 days, involving two different real-life cases. These two cases have been selected using the following criteria. The case should be representative in the sense that specifications are informal and therefore ambiguous and incomplete. It should be a realistic case, i.e. the organization involved or some departments in the organization involved should have a non-trivial problem that requires solving. Finally, the problem owners should be able to be available whilst the information modelling task is being dealt with. Particulars on the real-life cases can be found in Verhoef (1993).

These six modelling knowledge sessions have taken place in the experimental setting of Fig. 21. Thus, the modelling knowledge acquisition approach has been applied six times, resulting in...
six protocol transcripts. The interpretation task has led to six text-based models, which, finally, have been transformed into six meta-models. Additionally, the modelling knowledge as it is prescribed in the Yourdon method handbook of Yourdon (1989) has been represented, in order to compare the individual experts to their stick of reference. These results are presented in detail in Verhoef (1991, 1993), for the Yourdon meta-model as-it-should-be-applied, and for the six meta-models as-they-have-been-applied, respectively. The next section summarizes the main observations.

Results

This section discusses the insights gained in prescribed and applied modelling knowledge, based upon these seven meta-models. Given our focus on the early stages, emphasis has been on data-flow diagrams (DFDs) and entity-relationship diagrams (ERDs) while using a product oriented view on modelling knowledge in Yourdon, and on constructing the essential model while viewing Yourdon’s modelling knowledge from a process oriented perspective.

A product oriented perspective

Focusing on a way of modelling in Yourdon, the main modelling concepts are similar over the model type variants. For example, every ERD consists at least of entity types and relationships. DFDs always consist of processes and data stores, with flows between them. Although the main modelling concepts are similar it was observed that at the same time each model type variant has its own modelling concepts. Comparing the prescribed model type variants to the applied ones, it was observed that some prescribed modelling concepts, such as complex data flow and associative object type, are not applied at all. At the same time, the experienced information engineers used more refined modelling concepts. Examples are customer and supplier rather than external party, and planning, control, preparation, transformation, and termination processes, rather than just processes. Finally, the applied model type variants contain more concepts which serve communication purposes (e.g. sample value) or which provide quantitative information (e.g. frequency and volume). In addition to ERDs and DFDs, several other modelling concepts were used by the experts as well, in particular to create a (sometimes only mental) model of organizational aspects during the problem analysis stage. These non-diagramming concepts are found only in the applied ways of modelling. Some typical examples are: problem cause, organization unit, information need, and requirements. It was observed that several different graphical notations are used to denote one modelling concept. Three external notations for the modelling concept relationship within ERDs were seen. One of the information engineers even used two different graphical notations during one knowledge acquisition session. Clearly, the choice of a fixed set of graphical notations is not considered to be a matter of relevance during the problem analysis stage.
A process oriented perspective

Consecutive modelling tasks gradually lead to more structured models, both in the prescribed way of working and in the applied ways of working. In the course of modelling processes, more, and more refined modelling concepts are used, and the intermediate models have to satisfy a growing number of verification rules. The nature of modelling tasks changes from free to structured.

The applied ways of working differ from the reference book to a large extent with regard to the order in which modelling tasks are performed. The prescribed way of working is characterized by an almost strictly linear order of modelling tasks. The actual application shows an opportunistic order, which is determined by characteristics of the problem domain and of the problem at hand, as well as by the expert’s preferences. The information engineers reformulated their approach several times during the course of the knowledge acquisition sessions. In some cases, they even scheduled a number of tasks to be performed in advance. In most cases, however, they only stated that they preferred to pay attention to a specific part of the problem domain, usually to fill clear lacunae in their insights in the problem domain. Their momentary needs strongly influenced the order in which the several modelling techniques were used. Modelling techniques were used as a means to increase insight or to communicate insights, be it in the problem domain or in a specific solution scenario.

The experts showed individual ways of working. This is clearly demonstrated by the relative dominance of data modelling and process modelling. One of the applied ways of working can be characterized as data driven, one as process driven, whereas the third shows an equilibrium between the two.

Various process modelling strategies have been applied: input driven process modelling, output driven process modelling, and data driven process modelling. From an input driven point of view, processes handle events, and lead to other processes. From an output driven point of view, processes result in fulfilling information needs, and other processes are necessary to deliver the input for these processes. From a data driven point of view, processes manipulate data, i.e., create, read, use, and delete instances of entity types, relationships, and attributes.

Various data modelling strategies have been applied too: noun driven data modelling, object driven data modelling, and process driven data modelling. In the noun driven strategy, each noun in the description is considered to be a candidate entity. In the object driven strategy, objects in the real world are related to each other. Each object is questioned for the necessity of storing information on it. The process driven strategy investigates each operating process for entity types, and integrates the resulting partial data models.

As a final observation, the experts incorporated user participation as an essential ingredient in their ways of working. They often validated their results with respect to correctness and completeness. They focused on comprehensibility of intermediate information models, by adding sample values or quantitative data.

SUMMARY

As stressed in the introduction of this paper, feasibility of flexible information modelling support is dependent on (1) the number of modelling dimensions, (2) the complexity of information modelling in the early phases and (3) the extent to which flexibility is needed in these phases. The more modelling dimensions exist, the more complex information modelling is, and the more diverse information modelling processes are in practice, the more effort is needed to realize flexible information modelling support and the less feasible this goal is.

Section 2 dealt with the first research question. Three orthogonal modelling dimensions have been identified, capturing information engineers' modelling knowledge in the early stages: method versus application, product versus process, and graphical versus conceptual.

Section 3 dealt with the second research question and demonstrated the inherent complexity of a meta-modelling technique capable of describing all the relevant aspects (as defined in section 2) of information modelling methods. Information modelling products are in general quite complex due to decomposition mechanisms, complex structures and complex rules. In addition to the rules that information modelling products (syntax) have to satisfy, their formal meaning (semantics) must be also described. In the case of a data model this means that the meta-model should capture which instantiations satisfy the constraints specified and in the case of a process model, this means that all possible process executions have to be defined on the meta-level. Information modelling processes may be quite complex if modelling tasks may be performed in parallel. Furthermore, a formal and complete description of the precise effect information modelling processes may have on the various products (and vice versa) turns out to be difficult. Finally, both information modelling processes and products not only have to be approached from a conceptual point of view, but also from a representational point of view. Information modelling products may have complex associated representations and information modelling processes may have complex associated graphical interactions. This relation between the conceptual part of a meta-model and its representational part is essential for flexible support, but has hardly been investigated.

Section 4 dealt with the third research question and demonstrated the inherent diversity of information modelling in the early phases. The ways of modelling and the ways of working applied by the observed experienced information engineers differ to a large extent. Each information engineer uses its own rules, heuristics, graphical representations and so on. This means that for adequate flexible support, meta-models have to be constructed for each individual information engineer. Clearly, this is not feasible, especially since capturing the method followed by an experienced information engineer turns out to be a very time-consuming and difficult task. The six elicitation sessions led to voluminous text protocols, each including about 50 pages of text and about 30 diagrams, some of which went through several stages. Due to the bulky text protocols, the interpretation task and the conceptualisation task have been time-intensive for the knowledge engineer. This observation is even reinforced by the fact that the information engineers' way of working has been represented at a low level of granularity only. For the sake of diminishing his specification effort, the knowledge engineer has decided not to represent the level of manipulating individual objects. The representations did capture the level of diagrams going through several stages.
These specification effort problems may be partially solved if one is less ambitious. To achieve flexible support, it is necessary to find an adequate way to decrease the level of ambition whilst approaching this area. To be more precise, it is necessary to diminish the specification effort effectively. The easiest (and least satisfactory) approach is to neglect aspects of information modelling knowledge, in other words, to use the modelling dimensions as trade off parameters. For example, by not paying attention to the modelling process or by not paying attention to representational aspects. This approach has been used in the development of all 'state-of-the-art' meta-modelling techniques mentioned in section 1. None of these techniques address the modelling process. In Verhoef et al. (1991), the modelling process has been addressed. This reference, however, neglects the representational aspects of information modelling. Examples of using the modelling dimensions as trade off parameters are:

1. be less ambitious with regard to the level of consistency of modelling products
2. allow for a small number of graphical representations only
3. do not describe the modelling process into many levels of decomposition.

A more promising approach would be to exchange complete freedom for 'controlled flexibility'. Those specifying the knowledge base are then provided with a (pre-specified) generic meta-model, which may be adapted to one's needs by the application of a number of pre-defined meta-model transformations. Another promising approach to reduce the large specification effort is triggered by our observation that a detailed way of working of information engineers is difficult to acquire and requires a lot of effort to describe. Perhaps, it is best not to try to support this level in all details. Alternative support could then be achieved by offering a number of predefined operations. It is considered that such a building block approach would be an interesting issue for future research.

Summarizing, it is clear that unrestricted, adequate, flexible support of information modelling is, practically speaking, impossible to achieve. Sometimes, however, restrictions (e.g. when complete graphical support is not needed) may be perfectly acceptable. To balance benefits and efforts of flexible information modelling support, a research agenda has been presented, centred around the level of ambition to be realized. Whether the game is worth the candle depends on the choice of this ambition level.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous referees whose suggestions have improved the paper.

References


Vessey, I., Jarvenpaa, S.L. & Tractinsky, N. (1992) Eva-
Verhoef, T.F. (1991 ) Structuring yourdon’s modern structured
Smolander, K., Lyytinen, K., Tahvanainen, V.-P. & Marttinn,
Potts, C. A generic model for representing design methods.
Martin, J. (1990)
design: techniques and software support. In: Information
Processing 86. Kugler, H.J. (ed). North-Holland,
Amsterdam.
Martin, J. (1990) Introduction to information Engineering,
volume 1 of Information Engineering. Savanl Research Studies, Lancashire.
and Relational Database Design: a fact oriented
approach. Prentice-Hall, Sydney, Australia.
Potts, C. A generic model for representing design methods. 
In: Proceedings of the 11th International Conference on 
Smolander, K., Lyytinen, K., Tahvanainen, V-P. & Marttinn, 
methodology modelling. In: Proceedings of the Third 
The metaview system for many specification environ­
System specification semantics: defining the knowledge
captured by structured system development methods in
conceptual models. In: Information Systems Concepts: 
an In-depth Analysis. Falkenberg, E.D. and Lindgreen, P. 
(eds), pp. 53–77. North-Holland, Amsterdam, the Nether­
lands.
Verhoef, T.F. (1991) Structuring Yourdon’s modern structured
analysis. In: Proceedings of the Second Workshop on the 
Verhoef, T.F. (1993) Effective Information Modelling Sup­
uation of vendor products: CASE tools as methodology 
Welke, R.J. (1988) The CASE repository: more than
(eds). Georgia State University, Atlanta, Georgia.
Wijers, G.M. & van Dort, H.E. (1990) Experiences with the
modelling process: a view based on experiments with
Yourdon, E. (1989) Modern Structured Analysis. Prentice-
Hall, Englewood Cliffs, New Jersey.

Biographies
Arthur ter Hofstede received his master’s degree (1989) and his PhD degree (1993) from the University of Nijmegen in the Netherlands. He has worked for 4 years at the Software Engineering Research Centre (SERC) in Utrecht, where his research focused on meta-CASE technology and the foundations of conceptual modelling. Currently he is assistant professor at the Department of Information Systems of the University of Nijmegen. His research interests include conceptual data modelling, query languages, OO analysis, and meta-modelling.

Denis Verhoef received his master’s degree (1989) and his PhD degree (1993) from Delft University of Technology in the Netherlands. He has worked for 4 years at the Software Engineering Research Centre (SERC) in Utrecht. His research covered flexibility of CASE tools in theory and practice. Currently, he is research consultant at ID Research. His research projects include development approaches and tooling, workflow, meta-data management and Euromethod.