Formalisation of Techniques: Chopping down the Methodology Jungle

A.H.M. ter Hofstede*
Th.P. van der Weide†

April 28, 1993

Published as: A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.

Abstract

In this article we discuss formalisation of techniques in the context of Information System Development methodologies. When such methodologies are developed, the primary goal is applicability. After the methodology has proven itself in practice, the methodology will be applied in more sophisticated situations, pushing the methodology to its limits. In those cases, informal definitions are known to be inappropriate. We will go into some typical problems. After that, we describe a procedure for proper formalisation. The Predicator Model ([3] and [24]) is presented as an extended example. Finally, we describe some experiences with this approach to formalisation.

1 Introduction

In the field of Information Systems Development it seems common practice to introduce one's own methodology, supported by a more or less adequate set of tools (workbenches) to support information analysts. In the last

*This work has been partially supported by SERC project SOCRATES. Software Engineering Research Centre (SERC), P.O. Box 424, 3500 AK Utrecht, The Netherlands, E-mail: hofstede@serc.nl

†This work has been partially supported by the ESPRIT project APPED (2499). Dept. of Information Systems, Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, E-mail: tvdw@cs.kun.nl

decades this resulted in a tremendous number of new methodologies to be used in the development process of information systems. In [1] this situation is referred to as the *methodology jungle*. A recent estimate [5] speaks about *hundreds if not thousands* of information system development methodologies.

After [1] we think of a methodology (informally) as

> *a collection of procedures, techniques, tools and documentation aids which help the system developers in their efforts to implement a new information system.*

A technique can be seen as ([15])

> *part of an information systems methodology which may employ a well-defined set of concepts and a way of handling them in a step of the work.*

A more structured view on methodologies is given in figure 1 taken from [25]. This framework is based on the original framework of [18] and captures the different aspects of methodologies. It makes a distinction between a way of thinking, a way of control, a way of modelling, a way of working and a way of support. The way of thinking is concerned with the philosophy behind the methodology and contains basic assumptions and viewpoints of this methodology. The way of control captures project management. The way of modelling describes the models and model components used in the methodology, while the way of working describes strategies and procedures of how to arrive at specific models. The way of support, finally, is concerned with tool support.

One of the reasons (extenuations) causing the methodology jungle is that this field of study is still in its infancy. This is particularly clear from the fact that most techniques lack a formal foundation, making it impossible for example to compare them; the underlying concepts are not properly introduced. In some cases the syntax for communicating a specification is defined, for example, it is reasonably clear what constitutes a well formed Dataflow Diagram, see [8]. Hardly ever attention is paid to semantics, in terms of an underlying operational model, for example, the meaning of a Dataflow Diagram in terms of a set of cooperating processes. Discussing numerous examples, mostly relying on pictures, is a very popular style for the "definition" of new concepts and their behaviour. This can be compared to "proof by example", or sometimes even "proof by intimidation". We will refer to techniques lacking a formal basis as informal techniques.

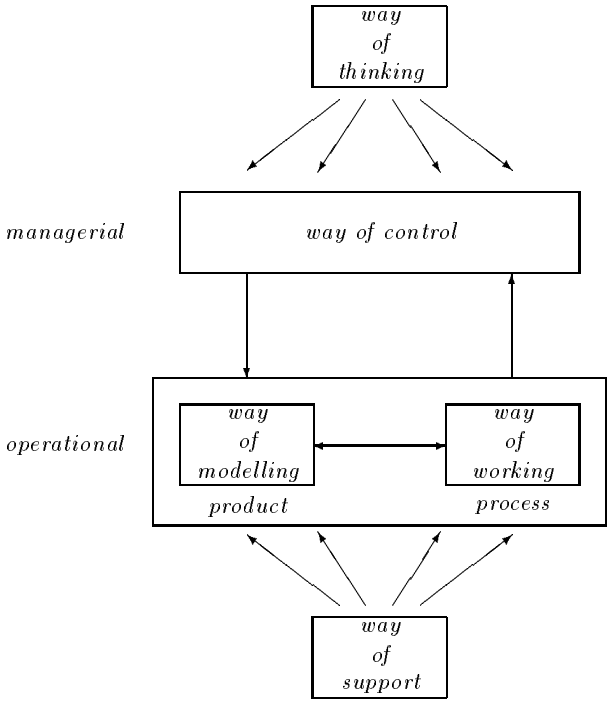


Figure 1: A framework for methodologies

The reason that informal techniques have become important is that they are usually visually oriented. In [10] the importance of *visual formalisms* is stressed:

> *Visual, because they are to be generated, comprehended and communicated by humans, and formal, because they are to be manipulated, maintained, and analyzed by computers.*

The power of a graphical formalism is also its weakness: both the application domain expert and the information analyst think they have a proper understanding of the concepts. At best, however, they only have a vague idea, what constitutes a *Well Formed Diagram* (shortly denoted as WFD), and an even more vague idea of its semantics.

What has to be done with existing informal techniques, to keep the advantages and to overcome the disadvantages? In our view, formalising (instead of discarding) informal techniques can be very worthwile, especially as many of these techniques are frequently used in practice and have in many cases proved their value there. The often overemphasised distinction between informal methods and formal methods (with the latter, usually formal specification methods are meant, as e.g. VDM [11] or Z [19]), suggesting a dichotomy between formal and informal, then becomes an artificial distinction: informal methods can become formal by formalisation. By formalising existing techniques no new technique has to be introduced. As a consequence, this results in conservation (no loss) of investment. Furthermore, similarities between different methods will become clear. This gives a clue for integrating methods by making minimal extensions (rather than the more usual top hat (hotchpotch) method for integration, see for example [15]).

Finally, a proper formalisation forms a solid base, and offers useful guidelines for the construction of teaching material. Resulting courses will have a broad scope, and will be of a more generic nature, which make them generally applicable.

In this paper we will discuss formalisation of techniques, how the indispensability arises, and what constitutes a proper formalisation. The organisation of the paper is as follows. First we will motivate formalisation. Then we describe the relevant ingredients of a formalisation. Finally, we discuss our experiences with such a formalisation of techniques.

2 The Need for Formalisation

In this section we emphasize the most important reasons why a proper formalisation is indispensable.

2.1 Ambiguity

An important motivation for the formalisation of an informal technique is that different interpretations will easily arise in cases, not covered by the examples by which the technique was introduced. *A forteriori*, situations that are not covered by any example are hardly ever recognised ([24]). The intention behind examples is to give hands-on experience, with the intention to convey the general idea. This may however be very difficult and sometimes even impossible. To illustrate this point, we will present a number of examples in the area of process modelling (section 2.1.1) and data modelling (section 2.1.2).

The consequences of different interpretations among persons can be disastrous, especially when this difference is between the application domain expert and the information analyst. This holds even more when the difference in interpretation is between man and machine ("the computer does not work", "the computer makes a mistake").

2.1.1 Example: Process Modelling

Activity graphs ([17]) can be used for process modelling. Activity graphs are bipartite directed graphs consisting of activities and states. Activities represent information processing tasks, while states represent information. The direction of an arrow between an activity and a state indicates whether that state is input or output of that activity.

In the Activity Graph of figure 2, activity A has two input states S_1 and S_2. The problem is that is not clear whether activity A needs an input from both input sets, or will be triggered also/only by a single input. Even worse, these different kinds of behaviour can not even be distinguished in terms of Activity Graphs. To state it more generally, in these kind of diagrams it is not possible to specify input-output relations precisely.

Dataflow Diagrams is another, often used, technique for process modelling. In figure 3, process P performs a check in the form of a query on the database that resides in datastore D. It is not clear what the direction of flow between P and D should be, in fact there is *no* information flow at all, since a query does not withdraw information from a database.

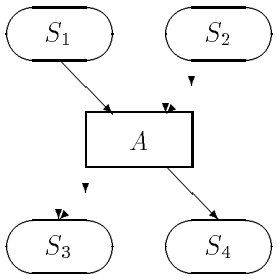


Figure 2: A sample Activity Graph

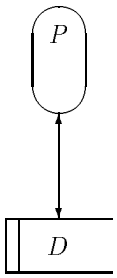


Figure 3: A sample Data Flow Diagram

2.1.2 Example: Data Modelling

NIAM ([14]) is a conceptual data modelling technique. In NIAM a distinction is made between *entity types* and *label types*. The difference is that label types can be represented directly on a communication medium, while entity types can not. Entity types and label types can play one or more *roles* in *fact types*, which is a generic term for relation types. Fact types may consist of one or more roles, and can participate in other fact types, leading to the notion of *objectified fact types*. In NIAM also the notion of subtyping exists. Entity types and label types can be subtypes of other entity types and label types. Each subtype hierarchy should have a unique top, which is referred to as its *pater familias* ([21]). In NIAM many types of graphical constraints can be expressed, examples are total role constraints, uniqueness constraints (keys) and set constraints. For NIAM a detailed procedure exists to support inexperienced information analysts to construct NIAM schemas. The appendix contains an overview of the graphical symbols of NIAM.

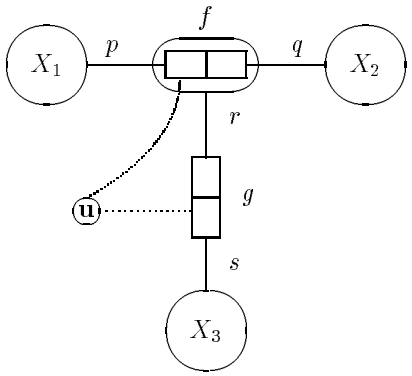


Figure 4: Uniqueness constraint over objectification

In figure 4 a simple NIAM schema, taken from [14], is shown. In this figure a uniqueness constraint is expressed over a so-called objectified fact type (f). Obviously the diagram of figure 4 is a *Well Formed Diagram* according to NIAM (see [14]). The meaning of the diagram requires knowledge of the meaning of the uniqueness constraint. The semantics of this constraint is explained in [14] by stating that it expresses a key on the ternary relation that is the result of flattening objectified fact type f. This key is shown in figure 5.

This example seems not difficult to understand, but does certainly not

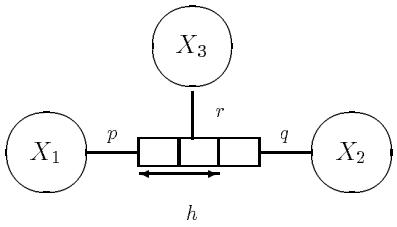


Figure 5: Semantics of uniqueness constraint

convey the general idea. First, the relation between fact types f and g on the one hand and h on the other hand is not clear. We will see examples, where the operation of flattening is not easily found by intuition.

Furthermore the reader might get the impression, that the schemata of figures 4 and 5 are equivalent (which of course is *not* the case!), giving the opportunity to switch freely between both alternatives, depending on the taste of the information analyst. Schema equivalence is not more than obliquely mentioned in [14], but the reader might have some intuition. But even then the situation is still not clear only from an example. What for example, is the meaning of the uniqueness constraints in figures 6 and 7? For yet another example, see figure 17.

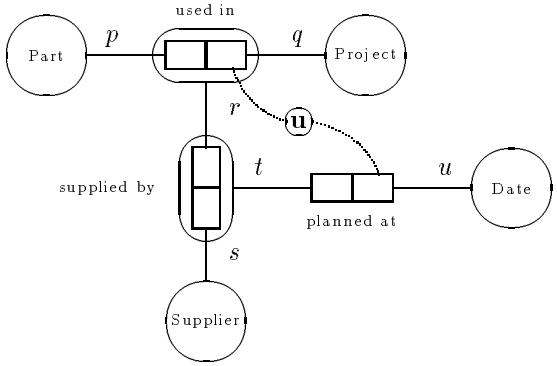


Figure 6: Complex uniqueness constraint

Note that ER has not enough expressive power for these examples (see [6], [20] and [27]). Due to its simplicity, complex cases where ambiguity may arise can not be specified. However, in figure 8 we see an example of an ER di-

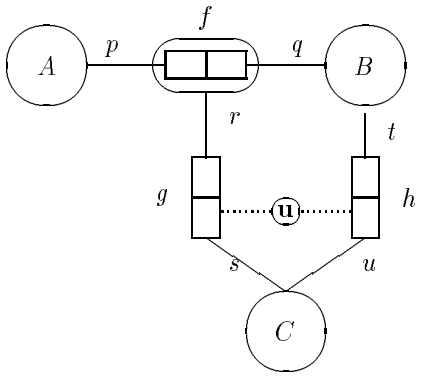


Figure 7: Another complex uniqueness constraint

agram with an unclear meaning, expressed in the ER dialect from [27]. In this dialect, an arrow from a relation type to an entity type indicates objectification of this relation type, thus enabling to use this relation type as an entity type. From [27] it is not clear whether these cyclic structures are allowed. They can be instantiated, but identification might be a problem. Identification is an important fundamental concept for information systems (see for example [20]).

2.2 Automated Support

A second reason for the need of formalisation is that a formal model gives a good clue for implementation. The better the formalism, the easier to implement. In fact, an implementation can be seen as yet another, but enormously more detailed formal description, usually in an imperative programming language.

Not only the implementation profits from a formal description. It also gives the opportunity for a much more sophisticated and extended support to the information analyst. The indispensability of automated support (with CASE tools) has been generally recognised. These tools are usually considered as a part of the Information System Development methodology (*way of support*, see figure 1).

Due to the lack of formality of the underlying concepts of the supported techniques, state of the art CASE-tools are not able to perform sophisticated verification checks, nor give warnings on suspect constructs.

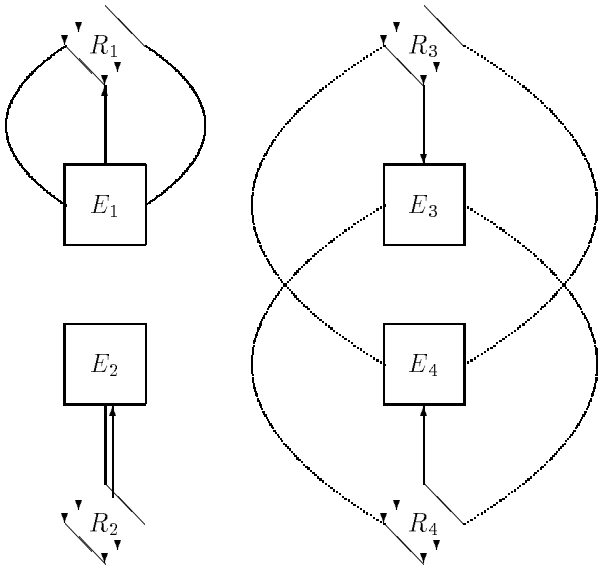


Figure 8: Some ER diagrams

2.2.1 Example: Supporting NIAM

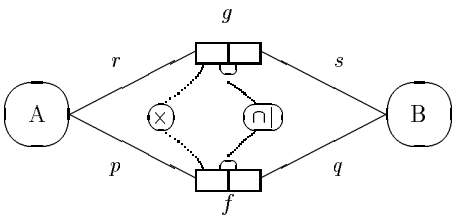


Figure 9: Inconsistent NIAM schema

Consider the NIAM schemata of figures 9 and 10. Current CASE-tools supporting NIAM (e.g. SDW, ProNIAM, RIDL*) do not detect the inconsistencies in these schemata, which are:

1. In figure 9, the subset constraint states that the set of instances of fact type f always should be a subset of the set of instances of fact type g. The exclusion constraint in figure 9 expresses that instances of type A occurring in fact type f should not occur as instance in fact type g

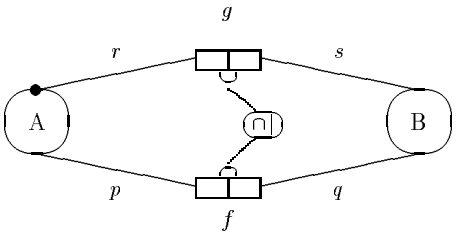


Figure 10: A suspect NIAM schema

and vice versa. Clearly, the only population of fact types f and g that satisfies both requirements is the empty population.

2. The subset constraint of figure 10 implies that every instance of type A occurring in fact type g should also occur in fact type f. The total role constraint states that every instance of A should occur in fact type g. If both constraints are correct, then a total role constraint stating that every instance of A should occur in fact type f is obviously missing. Note that this inconsistency is of a less serious nature as the inconsistency in the schema of figure 9, since in the schema of figure 10 the empty population is not the only valid population.

In both cases we have *Well Formed Diagrams*. If CASE-tools are to detect such inconsistencies, it is necessary that they are aware of the semantic definition of NIAM diagrams in general (and in this case, of constraints in particular).

2.2.2 Validation, Verification and Plausibility

Another important issue is *validation*, the question whether a *Well Formed Diagram* $\mathcal{D}$ meets its intended requirements in the Universe of Discourse. This cannot be checked by a computer, as the Universe of Discourse is only available in a highly informal format. So in order to validate $\mathcal{D}$ we would have to construct a formal description $\mathcal{F}$ of the Universe of Discourse, and then prove the equivalence of $\mathcal{D}$ and $\mathcal{F}$. That leaves us with the problem of validating $\mathcal{F}$, etc. This argument is known as *Church's Thesis* (see for example [12]).

In contrast to validation, *verification* can be performed automatically. Verification is checking whether a model, specified according to a certain technique, conforms to the rules of that technique. These rules can be checked automatically upon entry in the system.

A limited form of validity is testing on plausibility. Certain constructs are accepted during verification, yet are most probably incorrect. In some cases these implausibilities can be detected automatically. The implausibility of the diagram of figure 10 is an example. Another example is the occurrence in a Dataflow Diagram of a process that produces information out of nothing (i.e., a process without input flows), or a process that only consumes information (i.e., without output flows). Some other examples are:

Example 2.1 *Consider the following program fragment:*

i := 1; S; i := 2

If program fragment S *does not contain an application of variable* i*, directly or undirectly, then statement* i := 1 *is superfluous. It is not plausible that the programmer writes useless statements, so a warning should be given.*

Example 2.2 *Next we consider program fragment:*

i := 0;
while i > 0 **do** S

Suppose program fragment S *has no effect on variable* i*. Then we can conclude that the while-loop will not terminate if initially* i > 0*. Fortunately the while-loop is initialised by* i := 0*, effectively converting the while-loop into a skip-statement. Again we can not indicate an error in this program, but it is not very likely (plausible) that this program fragment properly relates to its intended meaning.*

For more information on static plausibilities see [23].

2.3 Properties

A third reason in favour of formalism is that a formal definition makes it possible to formulate and prove properties of techniques. One could for example look at the expressive power of a technique and compare it with other techniques. This is useful when looking for an optimal technique. A conclusion could be, that the one technique allows a much more precise specification, while the other technique provides insufficient possibilities to exclude invalid situations. This is for example the case with NIAM and ER (see section 2.1). This is an extra burden for the information analyst, or rather the programmer, to avoid bugs!

But also specific diagrams (within the same technique) can be compared and (dis)proved equivalent. This is important in order to abstract from the peculiarities of the actual information analyst. By introducing a set of transformation rules, transforming diagrams into equivalent diagrams, we get the opportunity to describe normal forms for *Well Formed Diagrams*. A normal form is a base for comparing different diagrams, probably originating from different information analysts trying to model the same Universe of Discourse.

Also properties of specific *Well Formed Diagrams* (models) according to some technique can be formulated and proved. An example of a schema property for a data model is structural identifiability, the question whether every entity type is identifiable (see [6] or [3]). An example of a property for a process model could be the fact whether deadlock or starvation can occur.

We see that properties can be proved for models in a technique and for a technique itself. In the latter case every *Well Formed Diagram* specified in the technique satisfies the property.

3 An Approach to Formalisation

From the previous section the need for formalisation will be apparent. In this section we outline a procedure for carrying out a formalisation. First we introduce the notion of *Well Formed Diagrams*, their construction and their meaning. Next we focus on the representation of *Well Formed Diagrams*, either graphically or textually. Then we discuss motivations and aesthetics regarding the choice of *Well Formed Diagrams*.

This approach is illustrated by two examples. The first example covers a formalisation of NIAM, the second example is concerned with process modelling.

3.1 Well Formed Diagrams

Formalising a technique consists of formalising its *way of modelling* (see figure 1). The important steps are:

1. Make a choice for the underlying system of concepts. Isolate these concepts, and describe their intended meaning (*way of thinking*).

   It is preferable to keep the number of concepts as small as possible. These concepts should be different and orthogonal. Advantage of this

proceeding is that proofs tend to become easier due to the fact that not so many cases have to be distinguished.

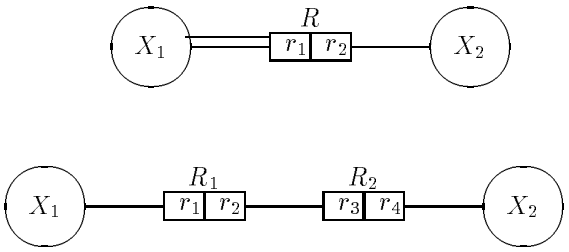


Figure 11: Syntactically incorrect NIAM schemata

2. Describe how a *Well Formed Diagram* is built from these concepts. The borderline between what is considered well formed and what is considered not operational may be subject to consideration. For example, the schema of figure 9 is considered a *Well Formed Diagram*, although its meaning is questionable. However, the schemata of figure 11 are not considered well formed. As another example, an Actvity Graph in which deadlock can occur, is still considered to be a *Well Formed Diagram* (see for example figure 12; note that this diagram would pass the plausibility check from section 2.2, as each process has both input and output states).

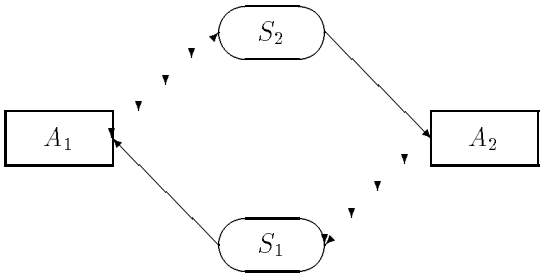


Figure 12: Activity Graph with potential deadlock

The reasons to allow certain non-operational *Well Formed Diagrams* are:

- it simplifies the process of formalisation, as at this stage these non-operational diagrams are very hard to characterise.
- it simplifies the process of modelling (*way of working*). The rationale behind this is that non-operational schemata obviously will not result from proper modelling.

3. Describe the meaning of a *Well Formed Diagram*. This requires an underlying operational model. The meaning of a *Well Formed Diagram* then is expressed in terms of this operational model. For example, a Dataflow Diagram ([8]) can be assigned a meaning by relating it to a PT-net ([4]). The semantics of a data model on the other hand, maps a diagram onto a state space, consisting of instantiations and transitions between these instantiations.

4. Describe a representation language for *Well Formed Diagrams*. Usually, more than one representation language will be introduced. A graphical language on the other hand is more suited when humans beings are involved, a textual language is more machine friendly.

3.2 Representation of Well Formed Diagrams

The representation language for *Well Formed Diagrams* should be such that precisely all *Well Formed Diagrams* are described. In order to simplify this, the *superset* technique again is applied (see item 2 above):

1. A simple language is introduced that specifies Diagrams, a superset of *Well Formed Diagrams*. This is usually called the syntax of the technique.

2. Extra rules restricting Diagrams to only *Well Formed Diagrams*. This is usually referred to as the semantics of the technique.

Note that the terms syntax and semantics in this respect are fairly misleading. We prefer syntax to cover the complete description of the representation language. This enables us to use the term semantics for the meaning of *Well Formed Diagrams*.

3.3 Aesthetics

Apart from the considerations of allowing non-operational diagrams as stated above, the following motivations and aesthetics for the choice of well formedness can be distinguished:

1. Diagrams that do not have a sensible meaning should not be considered well formed (in the latter case they would be assigned a void meaning). This is the case for the diagram of figure 11. This diagram is not well formed NIAM, as in the NIAM system of concepts every role should be connected to precisely one object type, as these roles form the domain of the mapping that constitutes the tuples of the fact type.

2. The (be it graphical or textual) notation should be flexible, providing information analysts the opportunity to use their own style of specification. Very restrictive languages usually are not convenient for everybody.

3. Conventions are needed when more constructs are possible to model equivalent cases. In case of deterministic modelling the conventions will result in a unique description for each *Well Formed Diagram*. Ideally, these restricting conventions are such that schemata are developer independent. This would mean, that every information analyst produces the same schema (if the schema is *valid*, i.e., an appropriate model of the Universe of Discourse).

4. Consequence of the *way of thinking* (see figure 1), the philosophy behind the technique. In NIAM for example uniqueness constraints over less than $n-1$ roles in an n-ary fact type do not make sense, as these fact types are considered to be non-elementary. Nevertheless they can be attached a sensible meaning (see [3]).

3.4 Example: The Predicator Model

The Predicator Model was introduced as a formalisation of NIAM (see [3]). We discuss the four important steps for formalisation, as discussed at the beginning of this section.

1. The Predicator Model has been built upon the following system of concepts:

   (a) a set $\mathcal{P}$ of predicators, with intended meaning a connection between an object type and a fact type. For example, in figure 5, p, q and r are predicators.

   (b) a set $\mathcal{O}$ of object types. An object type is intended to model a specific class of real world objects or relationships. In figure 5 we have the following object types: X_1, X_2, X_3 and h.

(c) a subset $\mathcal{F}$ of the objects $\mathcal{O}$, consisting of the fact types. The other object types ($\mathcal{A} = \mathcal{O} - \mathcal{F}$) are the atomic object types. In figure 5, h is the only fact type. X_1, X_2 and X_3 are the atomic object types.

(d) a subtype hierarchy for atomic object types. Figure 5 does not contain a subtype hierarchy.

(e) a set of constraints. The following types of constraints are distinguished: uniqueness, total role, exclusion, equality, subset, enumeration and occurrence frequency. Figure 5 contains a uniqueness constraint over predicators p and r.

The choice for this system of concepts was motivated by the objective of having a minimal number of concepts.

2. The Predicator Model allows a large class of *Well Formed Diagrams*. We summarise the most relevant requirements:

   (a) $\mathcal{F}$ is a partition of the set of predicators. In figure 5: $h = \{p, q, r\}$.

   (b) the constraints are properly formed, for example, each uniqueness constraint is a non-empty set of predicators that is *joinable via common descendants* (for more information on this, see [24]).

   Note that the diagram of figure 11 is not included as a *Well Formed Diagram*. This class of diagrams is larger than for example ER ([6]).

3. The meaning of a *Well Formed Diagram* is the set of instantiations (populations) that can be assigned to the diagram. Note that these populations are restricted by the (static) constraints.

4. A (machine friendly) representation language for the Predicator Model has been described in [3], in the style of Relational Algebra. User friendly representation languages are currently under development.

3.5 Semantics of Process Modelling Techniques

In this section, we outline the interpretation of *Well Formed Diagrams* in terms of an underlying operational model. Activity models, such as Dataflow Diagrams ([8]) and ISAC Activity Graphs ([13]), are generally used to model the processes within an information system and the information they produce and consume. This is typically done at a low ambition level, without

pretentions about the meaning of such a model. In [7] these modelling techniques are compared to each other by transformation of diagrams.

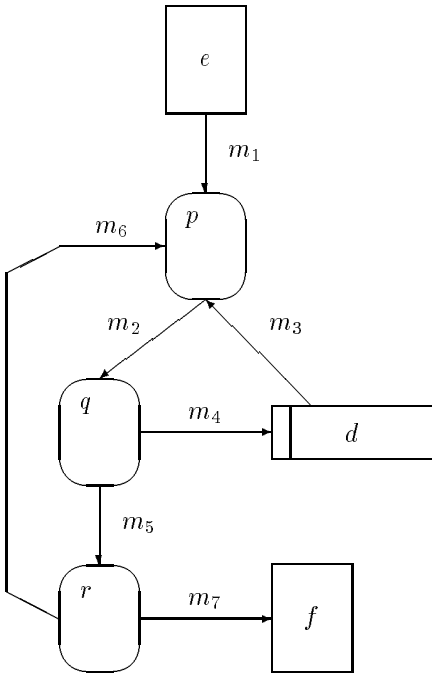


Figure 13: A sample Dataflow Diagram

Generally, a process description defines a class or set of objects related to each other by virtue of the fact that they are all activities which follow the dictated behaviour ([16]). An accepted method to assign meaning to activity models is to use some variant of Petri-nets as underlying operational model ([4], [22]). Well known variants are Predicate-Transition nets ([9] and [26]) and PT-nets ([2]).

The semantics of the dynamic behaviour of activity models then is described as a mapping from activity models into the underlying operational model. We illustrate this by an example taken from [4]. In this paper, PT-nets are used as underlying operational model. The Dataflow Diagram of figure 13 is interpreted as the PT-net of figure 14.

Another approach is followed in [26], where a schema interpeter for task structures is introduced as a Predicate-Transition net. As a consequence, the meaning of a task structure is defined as the interpretation by this schema

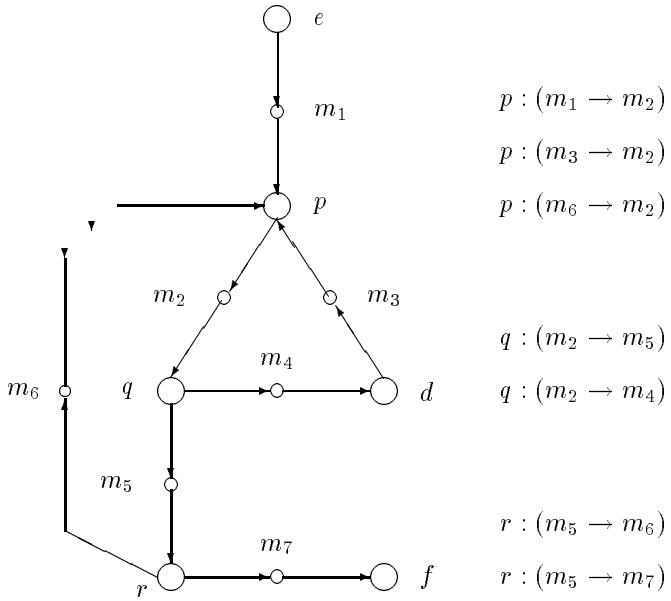


Figure 14: The Dataflow Diagram as a PT-net

interpreter. As such, this can be seen as an operational semantics.

4 Experiences with Formalisation

In this section we will discuss some of our experiences with the formalisation of NIAM in the Predicator model.

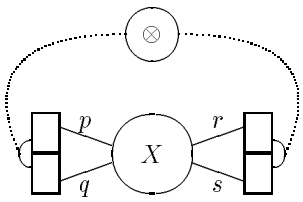


Figure 15: Ambiguous exclusion constraint

Formalising NIAM, which is completely graphically oriented, revealed that this graphical notation is not always univocal. To illustrate this we will

present two examples. First consider figure 15 in which we see an exclusion constraint between two binary homogeneous relationships. Note that it is not clear whether this exclusion constraint expresses the disjunction of the predicators $\langle p, q \rangle$ with $\langle r, s \rangle$ or of $\langle q, p \rangle$ with $\langle r, s \rangle$. The graphical notation should make it possible to indicate an ordering on the predicators involved in constraints. This would make it also possible for example to express quite elegantly the fact that a certain relation is symmetric, see figure 16 taken from [3].

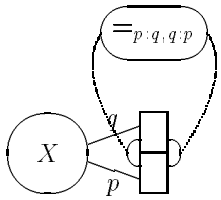


Figure 16: Symmetric relation

A second example of graphical ambiguity is shown in figure 17, again taken from [3]. In this figure a uniqueness constraint is shown which leaves room for alternative join conditions. Should the join condition state that $p = s$, $q = s$ or $p = s \wedge q = s$? This can not be derived from reference books on NIAM (e.g. [14]). Note that the graphical notation does not provide means for the formulation of alternative join conditions.

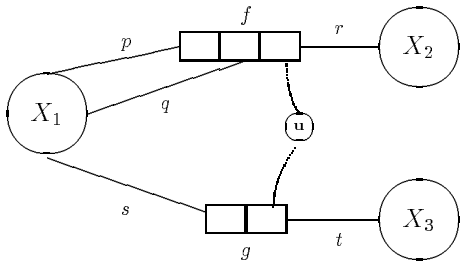


Figure 17: Ambiguous uniqueness constraint

As stated in section 2.3, formalisation of a method/technique enables the proof of properties of that method/technique. Properties that could be proven in the Predicator Model were, among others, that the NIAM constraints are not powerful enough to forbid the empty population and that

verification of NIAM schemas is an NP-complete problem. In fact it was shown that there are several levels of inconsistency possible in a NIAM schema, and therefore that there are several forms of verification. Two of these forms of verification were proven to be NP-complete. Note that this implies that the verification of NIAM schemas cannot be implemented efficiently in CASE-tools. It is important that good heuristics or incremental algorithms are found.

In our experience, formalisation provides for valuable insight in a technique. This makes it possible to construct teaching material based upon the formalisation. The Uniquest algorithm as presented in [3] and more elaborately in [24] can be considered for example as a very powerful way of deriving the (formal) semantics of a large class of uniqueness constraints, which can also be easily applied by novices. In [24] the Uniquest algorithm was applied on the uniqueness constraints of figures 6 and 7. Naturally this algorithm can also be implemented straightforwardly, which is a more positive result for CASE-tool developers.

5 Conclusions

In this paper the formalisation of techniques was discussed. The indispensability of such formalisation was stressed and illustrated by examples from a variety of well-known modelling techniques. The steps to carry out a formalisation were outlined, and finally, some of our experiences with a concrete formalisation were presented.

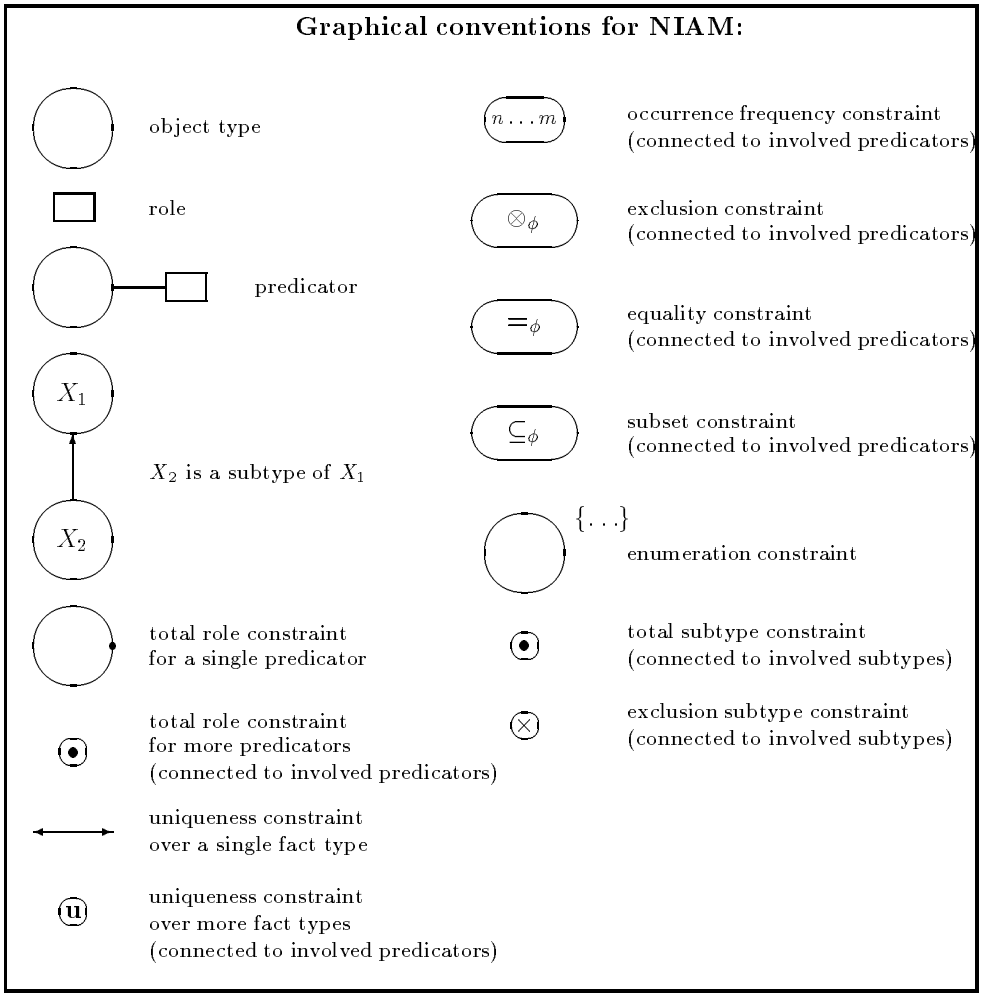
The field of study of Information Systems is not helped with the (informal) introduction of new methodologies or techniques incorporating yet other concepts. Formalisation of techniques is inevitable to produce order out of the chaos created by the tremendous number of existing methodologies and techniques (the Methodology Jungle). With respect to formalisation, it is important that emphasis shifts from defining syntax to defining formal semantics. It is essential to know the rules for Well Formed Diagrams, more important however is it to know their meaning.

Acknowledgement

We would like to thank Denis Verhoef for his contributive remarks.

Appendix: Legend of graphical symbols of NIAM

This appendix contains an overview of the symbols used in NIAM.



References

[1] D.E. Avison and G. Fitzgerald. *Information System Development: Methodologies, Techniques and Tools.* Blackwell Scientific Publications, 1988.

[2] J.A. Bergstra and G.P.A.J. Delen. Van dataflowdiagrammen via petrinetten naar systeemmatrixnotatie. Technical report, Mathematical Centre, Amsterdam, The Netherlands, 1982.

[3] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5), October 1991.

[4] P.D. Bruza and Th. P. van der Weide. The semantics of data flow diagrams. In *Proceedings of the International Conference on Management of Data*, pages 66–78, 1989. Hyderabad, India.

[5] J.A. Bubenko. Information system methodologies - a research view. In T.W. Olle, H.G. Sol, and A.A. Verrijn Stuart, editors, *Information System Design Methodologies: Improving the Practice*, pages 289–318. North-Holland, 1986.

[6] P.P. Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[7] E.D. Falkenberg, R. van der Pols, and Th. P. van der Weide. Understanding process structure diagrams. *Information Systems*, 16(4):417–428, Sept 1991.

[8] C. Gane and T. Sarson. *Structured System Analysis: Tools and techniques*. IST Databooks. MacDonald Douglas Corporation, St. Louis, 1986.

[9] H. Genrich. Predicate/transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and their Properties*, pages 207–247. Springer Verlag, 1987.

[10] D. Harel. On visual formalisms. *CACM*, 31(5):514–530, 1988.

[11] C.B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, 1986.

[12] H.R. Lewis and C.H. Papadimitriou. *Elements of the theory of computation*. Prentice Hall, Inc, Englewood Cliffs, New Jersey, 1981.

[13] M. Lundeberg, G. Goldkuhl, and A. Nilsson. *Information Systems Development - A Systematic Approach*. Prentice Hall, 1981.

[14] G.M. Nijssen and T.A. Halpin. *Conceptual schema and Relational Database Design: A fact oriented approach.* Prentice Hall of Australia Pty Ltd, 1989.

[15] T. W. Olle, J. Hagelstein, I. G. McDonald, C. Roland, H. G. Sol, F. J.M. Van Assche, and A. A. Verrijn-Stuart. *Information System Methodologies: A Framework for Understanding.* Addison-Wesley, 1988.

[16] L. Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering, Moterey, USA*, pages 2–13, 1987.

[17] G. Scheschonk. *Eine auf Petri-Netzen basierende Konstruktions, Analyse und (Teil)Verificationsmethode zur Modellierungsunterstützung bei der Entwicklung von Informationssystemen.* PhD thesis, Berlin University of Technology, 1984.

[18] P.S. Seligmann, 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*, 1989.

[19] J.M. Spivey. *Understanding Z: A Specification Language and its formal Semantics.* Cambridge University Press, 1988.

[20] T.J. Teory, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *Computing Surveys*, 18(2):197–222, 1986.

[21] O. de Troyer, 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, editors, *Proceedings of the IFIP WG 8.1 Working Conference on Computerized Assistence during the Information Systems Life Cycle*, pages 375 – 459, 1988.

[22] T.H. Tse and L. Pong. Toward a formal definition for DeMarco Data Flow Diagrams. *The Computer Journal*, 32(1):1–12, 1989.

[23] W.M. Waite and G.Goos. *Compiler Construction.* Springer Verlag, 1984.

[24] Th.P. van der Weide, A.H.M. ter Hofstede, and P. van Bommel. The uniquest algorithm: A formal semantics of complex uniqueness constraints. Technical Report 90-19, Department of Information Systems,

University of Nijmegen, The Netherlands, October 1990. To be published.

[25] G.M. Wijers and H. Heijes. Automated support of the modelling process: A view based on experiments with expert information engineers. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Advanced Information Systems Engineering*, pages 88–108, 1990.

[26] G.M. Wijers, A.H.M. ter Hofstede, and N.E. van Oosterom. Representation of Information Modelling Knowledge. In V.-P. Tahvanainen and K. Lyytinen, editors, *Proceedings of the Second Workshop on the Next Generation of CASE Tools*, pages 159 – 217, Trondheim, Norway, May 1991.

[27] E. Yourdon. *Modern Structured Analysis*. Prentice Hall, 1989.

Contents