EVALUATING METHOD ENGINEER PERFORMANCE: AN ERROR CLASSIFICATION AND PRELIMINARY EMPIRICAL STUDY

Steven Kelly, Matti Rossi
Department of Computer Science and Information Systems
University of Jyväskylä
P.O. Box 35
FIN–40351 Jyväskylä
Finland
email: {stevek, mor}@hyeena.jyu.fi
tel: +358 41 603036
fax: +358 41 603011

ABSTRACT

We describe an approach to empirically test the use of metaCASE environments to model methods. Both diagrams and matrices have been proposed as a means for presenting the methods. These different paradigms may have their own effects on how easily and well users can model methods. We extend Batra's classification of errors in data modelling to cover metamodelling, and use it to measure the performance of a group of metamodellers using either diagrams or matrices. The tentative results from this pilot study confirm the usefulness of the classification, and show some interesting differences between the paradigms.

KEYWORDS

method engineering, metamodelling

INTRODUCTION

In recent years we have seen a tremendous growth in the number of available software development methods. As has been stated, it is a practical impossibility to build a new CASE environment for every method, and there is no suggestion that everyone will begin to use the same method, or even one of a small set of methods (Bubenko 1988). The solution to this conundrum can thus lie only in a CASE environment which can be customised to support any method. A proposed solution, called CASE shells (Bubenko 1988) or metaCASE environments (Ald91), has produced promising research prototypes, and recently also commercial products. We believe that we are now at a stage where these environments can be evaluated by users to shed light on their usability and expressive power.

We describe an approach that can be used to empirically test the use of a metaCASE environment in modelling systems development methods. Modelling a method into a metaCASE environment is in itself a form of system development, along with its own paradigms, tools and methods: we could call these 'meta-methods'. In evaluating such meta-methods, as with any method, we should try to eliminate tool-centred factors, as these are not essentially part of the method. Similarly, we should not concern ourselves with the purely symbolic parts of the method's representations, as these may easily be changed without affecting the essence of the method. Rather, we should concentrate on the conceptual aspects of the meta-method, i.e. the metamodelling language. It is however possible to use completely different representational paradigms with the same conceptual meta-method: both diagrams (Ros95) and matrices (Kel95, Kin94) have been proposed as a means for representing metamodels. These different paradigms may have their own effects on how easily and well users can model methods. Most evaluations of metamodelling methods have so far taken place on a purely conceptual level, ignoring the effects of the way that different representational paradigms may affect method definition.

In this paper we aim to develop a research instrument that can be used to evaluate the effects of different representational paradigms by means of a controlled experiment of how accurately users model a method in a metaCASE environment. In this pilot study we use one metamodelling language, but two different representational paradigms, matrices and diagrams. Batra (Bat93) summarised the results of four experiments which studied errors made when performing data modelling, a similar activity to modelling methods. He classified the errors according to the *facet* (type of element, e.g. entity, attribute or relation) that was modelled, and concluded: "The analysis shows that the modeling of different facets requires different cognitive thinking, and leads to different types of errors". We adapt his classification and follow his approach, looking at each facet to see whether there is any difference in the performance between matrix and diagram representations. We believe that the developed classification can be used to study metamodelling errors in other settings as well and it should be easy to adapt to other environments.

We first describe the background of the study and key terminology of method engineering. We take Batra's classification (Bat90) and extend it to create a set of facets of metamodelling. Then we describe the metamodelling

task the subjects were asked to perform and use the classification to analyse our preliminary results. Finally we conclude and provide some future research issues.

Background and terminology

In its simplest form we can say that a *metamodel* is a conceptual model of a development method (Bri90). Consequently, metamodelling can be defined as a modeling process, which takes place one level of abstraction and logic higher than the standard modeling process (Gig91). At its simplest, a metamodel captures information about the concepts of a method and rules about how they may be used and connected together. For example, in Data Flow Diagrams the concepts used to model systems are processes, data stores, external entities and data flow relationships between them. A metamodel may also include information about the presentation forms (or signs, cf. Lep94) and uses of a method.

MetaCASE environments are configurable CASE environments that use a set of primitives (their meta-metamodel or metamodelling language) which allow them to describe a given method, and mechanisms to implement a CASE tool to support the defined method. Design of methodologies and of applications systems are very comparable exercises in human endeavour: in both cases one has to decide what data is needed and what processes are to be supported.

Brinkkemper (Bri95) has defined method engineering as "a discipline to design, construct and adapt methods, techniques and tools for the development of information systems". If the method engineering process is supported by specific computer aided tools we call the engineering discipline Computer Aided Method Engineering (CAME), and the supporting tools CAME tools. A person responsible for developing and implementing method specifications is called a method engineer (Kum92).

In this study we are interested in the utility of the CAME tools. The power of CAME comes from the definitive power of the metaCASE environment's metamodelling language and the ease of use of the CAME tools for a method engineer. Most CAME tools to date have used textual languages to represent methods, as in the very earliest tools (e.g. see ISD85), but recently there has been a move towards a more visual approach. The development of visual metamodelling languages has been heavily focused on diagrammatic paradigms, motivated by the fact that a large amount of methods themselves use diagrammatic representations. Hofstede and Weide (Hof93) emphasise the importance of diagrammatic formalisation in instantiating a method, and develop a general approach for such formalisation. Smolander et al. (Smo91), Protsko et al. (Pro89), and several others have proposed languages to represent graphical notations of a method, their connections, and/or graphical constraints. In addition, Kelly (Kel94) and Kinnunen et al. (Kin94) have investigated the utilisation of a matrix format as a visual representation form for metamodelling. In Appendix 1 we give a brief description of the MetaEdit+ tool and GOPRR metamodelling language used in this experiment.

One (implicit) assumption for the shift from a textual metamodelling language to a visual one lies in the belief of the improved user friendliness and accuracy. For instance, a study of data modelling with graphical E-R diagrams and a textual language (Bat90) found "overwhelming evidence in favor of" the graphical model for the greater correctness of the models produced. However, no studies in the method engineering field have been performed to support this belief, beyond a short comment in (Gol93). This necessitates empirical research, for example laboratory experiments, not only to evaluate this belief, but also to investigate user preferences for different visual representational paradigms. The implementation of visual paradigms should also be improved to address the wide spectrum of functionality which the metamodelling languages could support. This includes, for instance, the investigation of the role of visual languages for the querying and retrieval of reusable method fragments.

Classification of facets in metamodelling

In this chapter we develop a classification of user errors in metamodelling into various facets. As a starting point for our classification of facets we took that which Batra defined and used over a series of experiments (Bat90, Bat93). Batra's studies were of errors made while performing data modelling, i.e. modelling the structure of data for a database, an activity very similar to modelling methods; to our knowledge there has been no previous classification of facets specific to metamodelling. Batra's classification fits our needs particularly well because it has been successfully applied to models made using the Entity-Relationship method, from which our GOPRR method is descended. Also, Batra's analysis concentrated mainly on relationships, the subject of our hypothesis.

Batra claims that the errors arise from the Hutchins, Hollan and Norman's 'gulf', or directness distance between a user's goals and the human-computer interface (Hut85). We can argue that the metamodelling task is in essence a

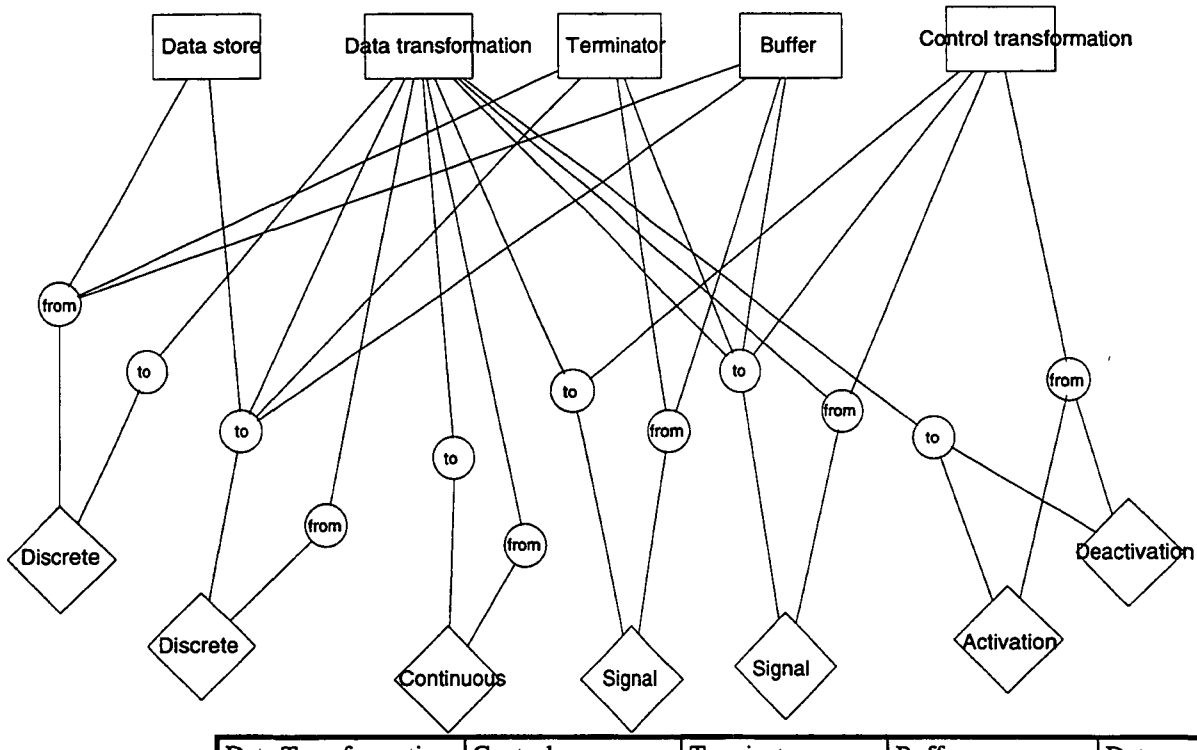
mapping task from a part of 'reality', which happens here to be the description of a software development method, into a conceptual representation of the method in a given metamodelling language and its representation forms.
We have modified Batra's classification to make it more appropriate to metamodelling (see table below). As properties in GOPRR are more complicated that attributes in ER, we supplemented the 'Descriptor' attribute facet with a new 'Property data type' facet. Similarly, properties in GOPRR can contain objects, something not found in ER: this caused the addition of a 'Complex property type' facet. Batra's 'Category' facet representing the use of inheritance was omitted from this analysis, because use of inheritance in GOPRR is more a matter of taste than correctness: all inheritance hierarchies produced by subjects were flattened before analysis. In place of the 'Category' facet we examined whether the right metatype was chosen for each element, e.g. that an object was not modelled as a relationship. Other minor changes could be made: for the relationship facets, 'binding' would be a better name within GOPRR, and 'n-ary' more accurate than 'ternary'; we maintain Batra's naming conventions for simplicity.
The facets can usefully be grouped into those dealing with objects (O), properties (P), and relationships (R). The last group is subdivided for the purposes of our hypothesis into simple relationships (unary and binary, R(S)), and complex relationships R(C).

	Facet	Description	Facet in Batra?
O	1. Entity	Object, Role or Relationship type identified	yes
O	2. Metatype choice (object, relationship etc.)	E.g. Relationship modelled as Object	changed from Category
O	3. Complex Property type	Is the right complex meta property type used (list, combo box, reference etc.)	added
P	4. Identifier	Does a property uniquely identify its owner	yes
P	5. Descriptor	Are there the right set of properties for a type	yes
P	6. Property data type	Is the data type right for each property (String, number etc.)	added
R (S)	7. Unary relationship	Relationships with two roles, in both of which is the same object	yes
R (S)	8. Binary relationship	Relationships with two roles, each with a different object	yes
R (C)	9. Ternary relationship, Role cardinalities	Relationships with more than two roles, either explicitly or by >1 cardinality for a role	yes

Research hypothesis and scoring

In this research we wanted to shed light onto some key aspects of the usability of advanced method engineering environments. There has been a clear move in the field from textual or programmatic method definition formats into graphical, matrix or form-based paradigms. We wanted to compare the performance of these new paradigms with each other: which paradigm was better for which facets of metamodelling. To do this, we needed to remove other factors such as the underlying metamodelling language, symbols and tools. We thus decided to analyse the difference between metamodelling with the Matrix and Diagram Editors of MetaEdit+: both work with the same underlying conceptual metamodelling language, GOPRR, and the tools are mutually consistent in their user interface: similar menu structures, use of popup menus, same toolbar etc. The matrix representation shows object types on the axes of the matrix and the relationship or role types in cells: the relationships shown in the matrix are *from* the object on the left axis *to* the object on the top axis. The diagram representation shows objects as rectangles, connected by lines via roles (circles) and relationships (diamonds).
As independent variable we took the use of the matrix or diagram editor, and as dependent variables the metamodelling correctness in each facet or area of metamodelling. The working hypothesis was:
Matrix is better with simple relationship modelling, Diagram better otherwise
This hypothesis is based on observations about the readability of the models and the ways that the diagrams and matrixes are used (Kel94). We believed that simple relationships (i.e. unary and binary) would be easier to read in the matrix representation, because the relationship information can be presented more compactly in a matrix without

the clutter and line crossings common in diagram representations. For example, Figure 1 shows the legal relationships for the objects of the Real Time Structured Analysis metamodel, represented as both a diagram and a matrix. To find what relationships are allowed from a Data Transformation to a Control Transformation, we can follow the grey arrows in the matrix and read across the Data Transformation row to the Control Transformation column, and find that only a signal relationship is allowed. Trying to find the same information from the diagram representation is somewhat more difficult.



	Data Transformation	Control Transformation	Terminator	Buffer	Data Store
Data Transformation	signal, discrete, continuous	signal	signal, discrete	signal, discrete	discrete
Control Transformation	signal, activation, deactivation	signal	signal	signal	
Terminator	signal, discrete	signal			
Buffer	signal, discrete	signal			
Data Store	discrete				

Figure 1. Real Time Structured Analysis OPRR Metamodel, as a graph (properties hidden) and a matrix (showing relationship names)

In the case of the more complex relationships (i.e. ternary or higher order), on the other hand, we expected the diagram representation to be easier to use, because whilst the matrix representation is still more compact, the ternary relationship components are scattered throughout the matrix. Outside of relationship handling, the differences between matrix and diagram formats seemed likely to be smaller, but we envisaged diagrams performing somewhat better because of their familiarity and more graphical nature. To test our hypothesis we built an experiment, which is described in the next section.

A scoring scheme was developed for giving consistent scores for different types of errors. To reduce subjectivity in analysing the metamodels we used an automatic report to provide the conceptual metamodel from each assignment in textual form, thus removing any effects the representational paradigm used might have on our ability to analyse the results, and reducing the possibility of bias towards our hypothesis in our analysis. The resulting report was in each case compared to that of a reference 'correct' metamodel, which had been made by the authors and checked by an

independent method engineer. As the use of inheritance in GOPRR was allowed, but would give distorted results (there is no 'correct' inheritance hierarchy) to the analysis, all inheritance hierarchies were first flattened and purely abstract types removed before the analysis.
There may be several different acceptable ways to model a method, depending on our approach, goals, and even personal style. Thus in our approach to classifying 'errors' in modelling methods we mainly classify as 'errors' cases where there is a clear omission, addition, internal inconsistency or undeniably incorrect model.
Whilst we found it relatively easy to spot correct components and incorrect components, additions or omissions, and to decide which facet the component belonged to, it was harder to determine how to score these occurrences within each class. The main problem was the lack of any definite number of correct components: although we had a reference metamodel, we recognised that our having e.g. 4 object types did not mean that was the only possible solution. Thus we decided to count the correctly modelled, incorrectly modelled, unnecessarily added, and necessary but omitted components separately for each facet in a subject's metamodel, and to form a percentage correctness within each facet from the number of correct components divided by the total number of components modelled by that subject for that facet (correct+incorrect+added+omitted). Thus we obtained percentage correctness scores for each subject for each facet.

Experiment set-up and tools used

The task given to the subjects was to build a metamodel for the Class Diagram of the Booch/Rumbaugh Unified Method (Boo95) using the Diagram or Matrix Editor in MetaEdit+. We had originally envisaged that the task would also include a second Diagram type, but time constraints forced us to restrict the task to the Class Diagram. The Unified Method, published only one month earlier, was chosen to ensure that previous experience with the given method would not interfere with the results. There were 11 study subjects, who were students from a method engineering course given at the University of Jyväskylä in fall 1995. Each subject had 20 hours of lectures about method engineering before the assignment, and they had been given two hours of training with the MetaEdit+ tools and eight hours of lessons about the GOPRR metamodelling language it uses. The students were divided randomly into two groups to model the Unified Method with either the matrix or the diagram editor of MetaEdit+.
Each subject was given instructions to make the metamodel of the Class Diagrams in a three hour modelling session. The material given them was part of the documentation of the Unified Method from Rational (Boo95). The students were instructed to only use the matrix or diagram editor for the task. The task performance time was controlled and the students had no possibility to talk to others, but they could obtain help from the researchers in possible technical problems with the tools.
Throughout the experiment experienced users of MetaEdit+ and GOPRR metamodellers were on hand to assist the modellers. They were instructed to answer questions on MetaEdit+ and GOPRR, but not on issues regarding decisions about how to model the given method. They were also instructed to actively look for models that showed a clear misapprehension of some part of MetaEdit+ or GOPRR, but similarly not to intervene in the case of errors in the model of the method. The result of the modelling session was either a matrix or diagram model of the Unified method. The results were analysed according to criteria given in Section 0.

Preliminary analysis of user metamodels

The results were analysed by the authors, who first analysed one assignment together to try to ensure that their use of the facet classification would be similar. The remaining assignment method models were divided between the researchers, half each, to be analysed. One subject was excluded from the results, as he only managed to model a tiny fraction of the method. The analysis of the remaining subjects' work has been done, but the interpretation of the findings is still underway. Here we present preliminary results.
Figure 2 presents the mean percentage score of all diagram users, and all matrix users, in each facet. Marked differences are few outside of the relationship facets: the clearest is in the Identifier facet, but this is probably to be explained by the small number of subjects and the overall lack of understanding in the subject group of how and when to use identifiers. The modelling of identifiers was not immediately visible in the matrix or diagram formats, and was performed in exactly the same way in both tools, thus we can safely disregard this as a statistical anomaly. Descriptor had the next largest difference of the non-relationship facets, and here we see the diagram format performing better: descriptors are immediately visible in a diagram, but in a matrix the user must open a separate dialog to see them.

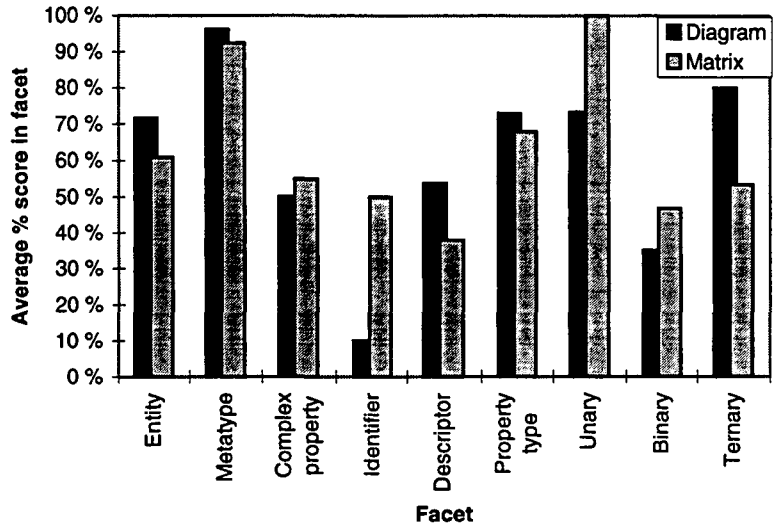


Figure 2. Mean percentage correctness score in each facet

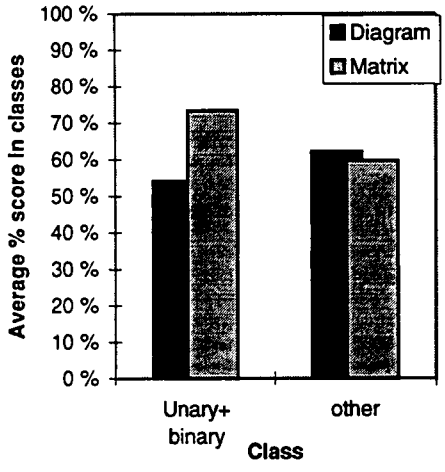


Figure 3. Mean score in unary and binary relationships, and elsewhere.

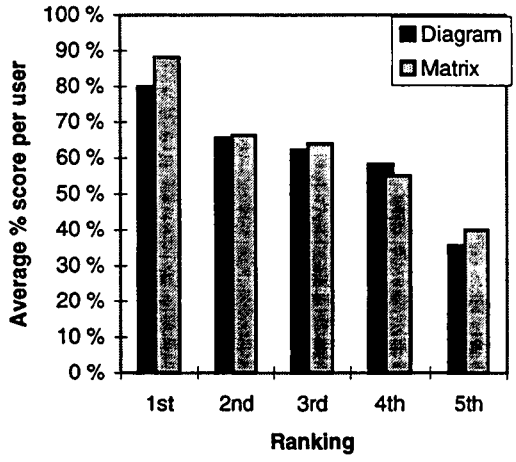


Figure 4. Each user's mean score over all facets (best Matrix user paired with best Diagram user etc.)

To test our hypothesis, we combined each subject's scores in the unary and binary facets, and again in all other facets, and compared the mean results of diagram and matrix users. Figure 3 shows the results, which would tend to support at least the first part of our hypothesis: that matrix users would be better on unary and binary relationships (73% matrix, 54% diagram). The second part of the hypothesis, that elsewhere diagrams would be better, is not as clear (62% diagram, 60% matrix). It would be better supported if we omitted the anomalous Identifier results (giving 71% diagram, 61% matrix); alternatively, it may be that the matrix format did indeed perform better overall than we had envisaged, further strengthening its claims as a metamodelling representation paradigm.

Overall, taking all facets into account, users of the matrix and diagram formats performed remarkably similarly: individual differences of subjects within a format were much greater than those between formats, as seen in Figure 4, which shows the overall mean score for each of the ten subjects. The mean was 60% for diagrams and 63% for matrices, with standard deviation of 16 percentage points in both cases. By a happy coincidence the best matrix user obtained almost the same score as the best diagram user, and so on down to the worst. Whilst this of course has in itself no statistical significance, it allows us to give a good subjective impression of the overall similarity of performance of the diagram and matrix users, showing each user of each format in order of mean score in a clear head-to-head comparison.

In addition to these preliminary summary results, we have also begun analysing the test scores in more detail using SPSS. Using different subjects for the diagram users and matrix users gives us two independent samples, enabling better analysis than if the same subject had performed the modelling task with each tool. The scores in each facet are continuous interval variables (there is no clear zero point), although it may be safer to regard them as discrete, given that the scores are the quotient of two discrete values. The independent variable, use of matrix or diagram, is strictly a nominal variable: however, taking it as a binary variable with values of 0 and 1 is not unusual and allows the use of more sophisticated statistical tests. Looking at Pearson product-moment correlation coefficients, we see the largest correlation between scores on objects (facets 1–3) and on properties (4–6): 0.6785 ($p=0.016$), followed by that between simple relationships (7–8) and complex relationships (9): 0.4721 ($p=0.084$). Given our hypothesis concerned only simple relationships, we can examine the partial correlation between use of matrix and simple relationship score, controlling for the complex relationship score: this gives 0.59 ($p=0.047$). The similar test for matrix and object, controlling for property scores, gives –0.30 ($p=0.44$). In other words, there is a statistically significant direct relationship ($p<0.05$) between the use of a matrix and the score obtained for simple relationships, controlling for the score for other relationships; there is no such correlation between use of matrix and scores for objects or properties.

Multiple regression results seem to show that use of matrix and ternary score form a good predictor of unary scores ($R^2=0.61$, significance of $F = 0.038$): the ternary has partial regression coefficient $B=0.53$ (significance of $T = 0.029$) and matrix has $B=0.41$ (significance of $T = 0.032$). If the use of matrix is excluded from the model, the significance of T for the ternary term is no longer meaningful (0.18).

These SPSS results remain however preliminary, subject to more detailed examination of the data and assumptions of the statistical tests used.

CONCLUSIONS

In this paper we have performed a pilot study for an empirical experiment which would analyse the usability of different method engineering representational paradigms. The study served as a preliminary test for our metamodelling error classification. The classification was developed from Batra's error facet classification for database design (Bat90).. We found our extension of Batra's classification to be a useful tool, and believe that it can be a valuable aid in empirical evaluation of metamodelling languages and tools. Comparing our results with Batra's, we find some overall similarities, but a striking difference in the property group of facets. Batra's subjects were "usually successful" in modelling attributes, whereas that was the worst area for our subjects: a mean score of 49% compared to 65% for relationships and 71% for objects. This is probably due to the greater variety and complexity of properties used in modelling methods than in data modelling, or maybe the fact that the attributes do not have a separate representation in GOPRR.

On the basis of this pilot study we can make some tentative conclusions about the performance of matrix and diagram formats for metamodelling, and use these to generate hypotheses for further research to confirm the findings. We believe that this kind of approach can give us important feedback about the performance and usability of method engineering tools. This kind of research is needed, because there is a lack of empirical evidence for the claimed usefulness and ease of use of metamodelling and method engineering environments.

Our initial findings tend to support our hypothesis that the matrix is a better format for working with simple relationship types in metamodels, whilst in other areas the diagram format seems to perform slightly better. Due to the small sample size ($n = 10$), however, these results cannot be regarded as conclusive. Should further research confirm these findings, we can draw at least two practical conclusions:

- CAME environments should support more than one representation paradigm for metamodelling, as different parts of the metamodelling task are better performed by some paradigms than others.
- When metamodelling with matrix and diagram tools, it would be useful to perform most work in the diagram tool, but addition of simple (unary and binary) relationships could best be performed while viewing the metamodel as a matrix.

The latter result is doubly important for binary relationships, as (1) these form the bulk of all relationships in general in methods, and (2) they had the second worst overall percentage score for both diagram and matrix users (after identifier and descriptor respectively).

To confirm our tentative findings from this experiment we would need to extend the setting in three ways: a larger number of participants, larger metamodel, and greater method engineer experience. Another extension would be to include more metamodelling paradigms, such as textual and form-based: this would give further guidance to metamodellers as to which paradigm to use when, and to CAME environment designers as to which tools to include.

When using multiple paradigms for metamodelling there would be important effects from how well integrated the paradigms were in the CAME environment. Studies should thus also be performed on CAME environment use with multiple paradigms: does switching between paradigms form a problem for the method engineer, possibly even outweighing the benefits of the different paradigms for different tasks.
Another approach to analysing how well method engineers perform with different paradigms could be a comprehension task, such as the ones described in (Par96). Because a comprehension task uses a predefined model from which the subjects find answers to predefined questions, assessing the results would be easier. It would shed light on the problems of interpreting the metamodels, whereas here we analysed the problems of developing the metamodels.

REFERENCES

Alderson, Albert, "Meta-CASE Technology," pp. 81–91 in **Software Development Environments and CASE Technology**, A. Endres, H.Weber (Ed.), Springer-Verlag, (1991).

Batra, D., J. Hoffer and P. Bostrom, "Comparing Representations with Relational and EER Models," **CACM** 33(2) (1990) pp.126–139.

Batra, D., "A Framework for studying human error behavior in conceptual database modeling," **Information & Management** 25 (1993) pp.121–131.

Booch, G., J. Rumbaugh, "**Unified Method for Object-Oriented Develoment**," Rational Software Corporation, Santa Clara, US (1995).

Brinkkemper, Sjaak, "**Formalisation of Information Systems Modelling**," Ph.D. Thesis, Univ. of Nijmegen, Thesis Publishers, Amsterdam (1990).

Brinkkemper, Sjaak, "Method engineering: engineering of information systems development methods and tools," **Information & Software Technology** 37(11) (1995) p.1–6.

Bubenko, J. A., "A Method Engineering Approach to Information Systems Development," the proceedings of the **IFIP WG8.1 Working Conference on Information Systems Development Process** (1988) pp.167–186.

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

Goldkuhl, Göran, Stefan Cronholm, "Customizable CASE Environments: A Framework for Design and Evaluation," Accepted to **COPE IT '93. LiTH-IDA-R-93–42**, Linköping University, Sweden (1993).

Hofstede, A. H. M. ter, Th. P. van der Weide, "Formalisation of techniques: chopping down the methodology jungle," **Information & Software Technology** 34(1) (1993) p.57–65.

Hutchins, E. L., J. D. Hollan and D. A. Norman, "Direct Manipulations Interfaces," **Human Computer Interaction** 1 (1985) pp.311–338.

ISDOS, "**System Encyclopedia Manager, Language Definition Manager: User Manual (SEM/LDM)**," Version 1.4 (June 1985).

Kelly, Steven, "A Matrix Editor for a MetaCASE Environment," **Information and Software Technology** 36(6) (1994) pp.361–371.

Kelly, Steven, "What's in a Relationship: on distinguishing property holding and object binding," in **Proceedings of 3rd International Conference on Information Systems Concepts, ISCO 3**, W. Hesse and E. Falkenberg (Ed.), University of Marburg, Lahn, Germany (1995).

Kelly, S., K. Lyytinen and M. Rossi, "MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment," pp.1–21 in **Proceedings of the 8th International Conference on Advanced Information Systems Engineering, CAISE'96**, Springer-Verlag (1996).

Kinnunen, Kimmo, Mauri Leppänen, "O/A Matrix and a Technique for Methodology Engineering," in **Proceedings of the Fourth International Conference on Information Systems Development**, J. Zupansis and S. Wrycza (Ed.), Moderna Organizacija, Kranj, Slovenia (1994).

Kumar, Kuldeep, Richard J. Welke, "Methodology Engineering: A Proposal for Situation Specific Methodology Construction," pp. 257–269 in **Challenges and Strategies for Research in Systems Development**, Kottermann W. W. and Senn J. A. (Ed.), John Wiley & Sons, Washington (1992).

Leppänen, Mauri, "Metamodelling: Concept, Benefits and Pitfalls," pp. 126–137 in **Proceedings of the Fourth International Conference on Information Systems Development**, J. Zupansis and S. Wrycza (Ed.), Moderna Organizacija, Kranj, Slovenia (1994).

Parsons, J., "On Experimental Evaluation of Classification in Information Modelling: Local versus Global Schemas," **Proceedings of the EMMSAD'96 Workshop**, University of British Columbia, Vancouver (1996).

Protsko, L. B., P. G. Sorenson and J. P. Tremblay, "Mondrian: system for automatic generation of dataflow diagrams," **Information and Software Technology** 31(9) (1989) pp.456–471.

Rossi, M., "The MetaEdit CAME environment," **Proceedings of the MetaCase 95**, University of Sunderland press, Sunderland (1995).

Smolander, Kari, Kalle Lyytinen, Veli-Pekka Tahvanainen and Pentti Marttiin, "MetaEdit — A Flexible Graphical Environment for Methodology Modelling," pp. 168–193 in **Advanced Information Systems Engineering, Proceedings of the Third International Conference CAiSE'91, Trondheim, Norway**, May 1991, R. Andersen, J. A. Bubenko jr. and A. Solvberg (Ed.), Springer-Verlag, Berlin (1991).

Welke, R. J., "The CASE Repository: More than another database application," in **Challenges and Strategies for Research in Systems Development**, William W. Cotterman and James A. Senn (Eds.), Wiley, Chichester UK (1992).

APPENDIX 1

A short description of MetaEdit+ and its GOPRR metamodelling language is given below, for a longer discussion see (Kel96). MetaEdit+ allows the user to specify the ISD method to be used, and to use several different tools to make and manipulate models of the IS according to that method. These different tools represent different representational paradigms: the same underlying conceptual data can be shown in various formats, including graphical diagram, matrix and table formats. The same information can be handled in a way appropriate to the current task: for some tasks it is easier to work with the data in diagram form, for others in matrix form. Both forms rely on the underlying conceptual information, which contains no information about what the representation is like. MetaEdit+ models the underlying conceptual information of both models and metamodels with GOPRR (Kel96), which forms an evolutionary extension of the OPRR model, used successfully in specifying methods for MetaEdit (Wel92, Smo91).

The fundamental GOPRR modeling constructs are:

- **G**raphs, which are aggregates of a set of objects and their connections. Examples of graphs are an Entity Relationship Diagram or a Data Flow Diagram.
- **O**bjects, which are considered as independent and identifiable design objects. Examples of objects are an Entity in an Entity Relationship Diagram or a Process in a Data Flow Diagram.
- **P**roperties are attributes of graphs, objects, relationships and roles. An example of a property is the number of a Process in a Data Flow Diagram.
- **R**elationships are associations between objects. An example of a relationship is a Data Flow in a Data Flow Diagram.
- **R**oles define the ways in which objects participate in specific relationships. An example of a role is the 'to' end of a data flow relationship, which is indicated by an arrowhead.

The GOPRR model is equally applicable to both the metamodel (type) and model (instance) levels of CASE data, and thus is particularly well suited to a metaCASE environment, as the same concepts can be used both for making methods and for making models with those methods. A particular benefit is the ability to make a model of a method on the *instance* level, and then transform that by a simple mapping into a *meta*model for the method on the type level.

In addition to these fundamental concepts or metatypes GOPRR also uses some auxiliary constructs, the most important of which is *binding* (Kel95). A binding specifies a relationship, a collection of roles, and for each role a collection of objects taking part in that relationship in that role. In other words, the objects, relationships and roles themselves store no information about each other; rather, a graph includes a set of bindings that specify how its various elements are connected together. In a metamodel bindings work as rules, specifying the legal ways in which objects of the different types can be connected via relationships of various types.