Transforming Diversity into Uniformity – Experiments with Meta-structures for Database Recording

Torsten Madsen

University of Aarhus, Dept. of Archaeology

Moesgaard 8270 Hojbjerg, Denmark

e-mail: farktm@moes.hum.aau.dk

1. Introduction

In the sixties a high-spirited optimism spread throughout many parts of archaeology. The advent of the digital computer and the promise of storage and handling of information on a grand scale had an impact on archaeologists. As recalled by Scholz and Chenall the prospect was to create "A framework or model for recording any site, site feature, artefact, or archaeological situation, in the form of a highly structured, though expandable, code for the descriptive attributes of function, form, material, technique of manufacture, surface treatment, and design" (Scholz and Chenhall 1976:92).

However, the optimism did not last for very long. In the mid seventies Scholz and Chenall noted "that a generalised data bank is not very useful for research purposes". Moreover it seems evident that a preliminary description of data is necessary to formulate ideas concerning the choice of variables, which need to be observed and scaled for any specific purpose. The time to record data on the computer would appear to be only after this preliminary descriptive step - i.e., when actual procedures and variables have been defined for testing specific hypotheses. Data categories for observation, and conventions for recording data cannot be chosen independently of problem orientation" (Scholtz and Chenhall 1976:92).

The lesson learned by then was that there is no path leading to a unified description of archaeological materials. Descriptions are bound to particular research problems, and as these are ever changing, so will descriptions. Over the last 25 years the interest for databases in archaeological research has been marginal. Many researchers have created specialised databases for recording particular materials. As soon as the results of the research have been published all interest in the databases and their content is lost.

On the other hand, administrative bodies associated with archaeology have systematically tried to develop the use of databases over the last 30 years. Their answer to the Tower of Babel problem of database recordings has been - standardisation. If only we can agree to the same description system, there will be no problem at all, they argue. The administrators have increasingly been able to agree, and increasingly lost contact with the realities of research.

On previous occasions I have argued that standardisation of content is a non-issue in archaeological research (Madsen 1998, 1999a). In the years to come we may find an increase in large-scale standardised and centralised administrative databases, but at the same time we will find an explosion in small local databases with their own unique structure serving the research efforts of individuals, projects or small organisations. Wherever research is carried out there will be a need for individually designed recording systems that will contain the base material for this research. The content of these databases, however, are in grave danger of being lost. As soon as the tasks for which they are created have been accomplished, they merge into oblivion.

One possible remedy for this inherent anarchy in data recording and consequent loss of data is to design a database structure that will literally encompass all other structures. A database designed in such a way that it does not represent any particular part of reality, but rather represents the way we model reality for the purpose of database recording using, for example entity-relationship modelling. What we are looking for is a meta-structure for database recording. One benefit of such a database is that accessing data can always happen through the same application interface no matter what particular data structure is actually at hand, and it will thus be infinitely easier for users to access data. Furthermore, time could be invested in creating efficient and powerful ways of searching and presenting data, because the investment would not apply to just one database instance, but to all instances.

Over the last three years I have experimented with a system called GUARD (the name possibly an acronym for General Utility Archaeological Recording Database). Its background lies in the IDEA project from the mid-nineties (Andresen and Madsen 1992, 1996a, 1996b). This project, carried out together with Jens Andresen, was aimed at creating a flexible database solution to excavation recording. By the end of the project, which was partly a success, we used the experiences gained to outline the principles for a meta-structure for database recording. Although the design has been modified considerably from what was originally conceived, it is these principles that I have used now to implement a working system.

2. The meta-structure design

The meta-structure I have chosen is not very complicated, but as the level of abstraction is high it may not be easy to grasp how it works at the first glimpse. However, if you keep in mind that what is implemented is more or less the entity-relationship model, it is not all that difficult. I have used six basic building blocks named *Entity types*, *Classes*, *Entities*, *Attributes*, *Entity Attribute Values* and *Relationship Attribute Values*. These are the entity types of the meta-structure (figure 1).

The reason for including *Classes* as a separate entity type may not be self-evident and has in fact caused much debate between Jens Andresen and I. Why separate *Classes* from *Entity Types?* For instance, if you take artefacts, these will surely constitute a basic Entity Type in any specialist's database aimed at artefact recording. However, the artefacts will also be qualified according to elaborate class-hierarchies, and different classes are likely to have

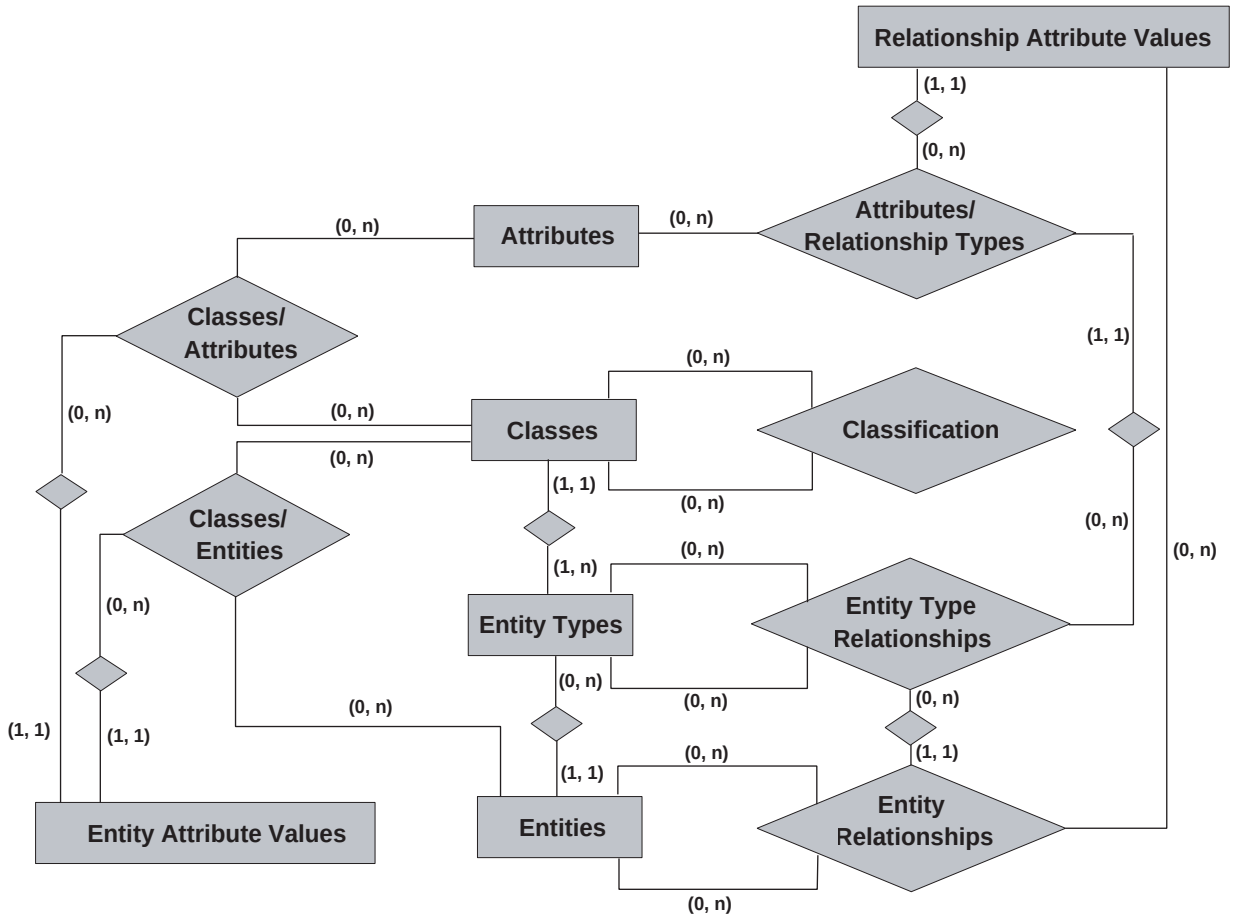

Figure 1: Entity-relationship model for GUARD.

different descriptive attributes. According to the standard database theory entities with different attributes should also be of different entity types. Ultimately, this would mean that each class could end up as its own entity type with a separate unique identification number, which of course would not agree with the way we normally use classifications.

To counter this problem *Classes* has been introduced as a kind of entity type that does not posses identification numbers – well it does, but the user is kept unaware. Otherwise it has all the qualities of entity types. For practical reasons it has become *Classes* and not *Entity Types* that have attributes associated with them. Thus any instance of *Entity Types* has at least one class called the root. If attributes are to be associated directly with an instance of *Entity Types* this will happen through its root class. In addition, any number of classes may be associated with an instance of *Entity Types* and each class may have its own set of attributes. Obviously, a relationship exists between *Entity Types* and *Classes* so that an instance of *Classes* can only exist if linked to an instance of *Entity Types*, and an instance of *Entity Types* must have at least one instance of *Classes* associated with it. Further, a relationship exists between *Entity Types* and *Entities* so that an instance of *Entities* can only exist if linked to an instance of *Entity types*.

Six named relationship types tie the six basic entity types of GUARD together. Three of these - *Entity Type Relationships*, *Classification* and *Entity Relationships* - are used to create an internal structure for *Entity Types*, *Classes* and *Entities* respectively, with a many to many cardinality between their entities.

The other three relationship types named *Classes/Entities*, *Classes/Attributes* and *Attributes/Relationship Types* are used to tie different entity types together with a many to many cardinality. Thus *Classes/Entities* makes it possible for an instance of *Entities* to be associated with many instances of *Classes*, and an instance of *Classes* to be associated with many instances of *Entities*, while *Classes/Attributes* makes it possible for an instance of *Attributes* to be associated with many instances of *Classes*, and an instance of *Classes* to be associated with many instances of *Attributes*. Together these two are then linked to the *Entity Attribute Values* entity type making it possible to assign a value to a unique combination of *Entities*, *Entity Types*, *Classes* and *Attributes*. The relationship type *Attributes/Relationship Types* makes it possible for an instance of *Attributes* to be associated with many instances of *Entity Type Relationships* and an instance of *Entity Type Relationships* to be associated with many instances of *Attributes*. *Attributes/Relationship Types* together with *Entity Relationships* are linked to the *Relationship Attribute Values* entity type making it possible to assign a value to a unique combination of *Attributes*, *Entity Type Relationships* and *Entity Relationships*.

The Entity-Relationship diagram of GUARD is thus fairly straightforward, and the same is true if we look at the table structure (figure 2). The only slight complication concerns the *Entity Attribute Values* and the *Relationship Attribute Values*. In both cases it is not just one table that is needed but as many tables as there are different data types involved. Each record in a table stores one value and that value is of one type only for that specific table. Thus, there are separate tables for text, double, integer, memo, etc.

Many questions probably spring to mind when confronted with a structure like this. Will it work at all? And if it works will it be

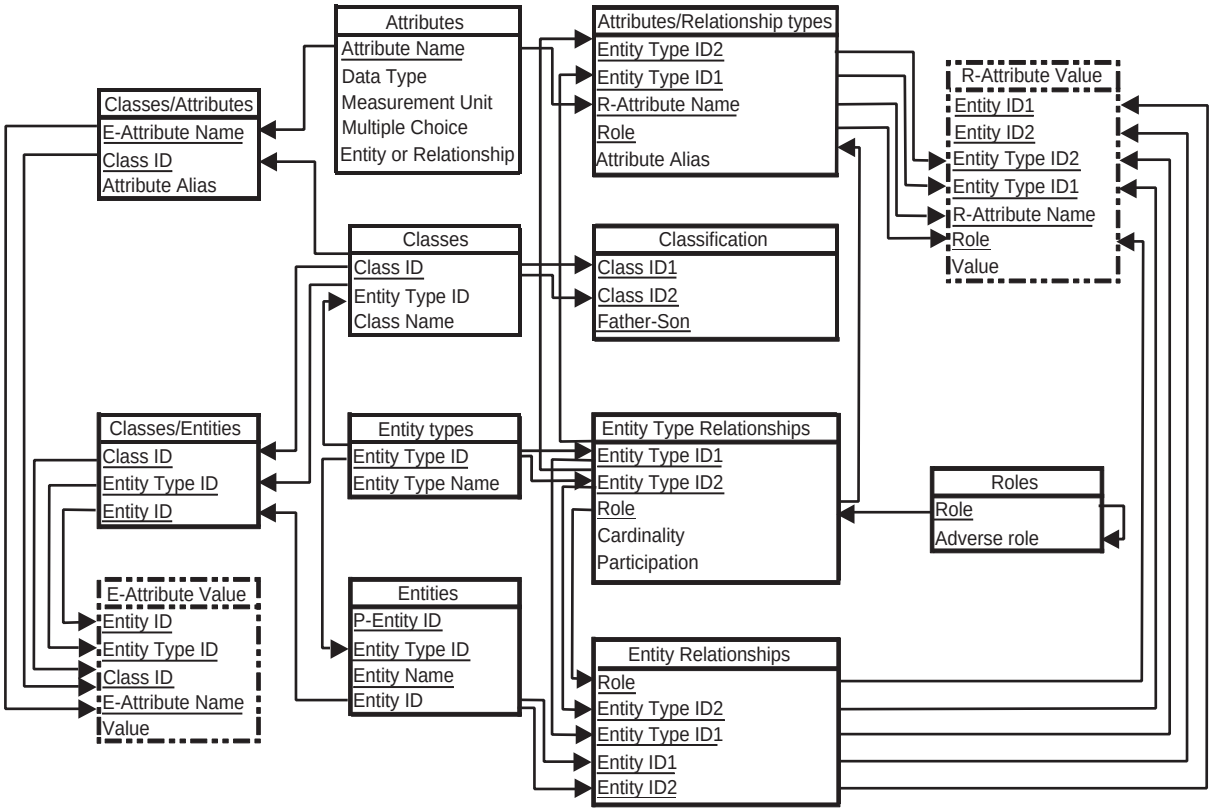

Figure 2: The central table structure of GUARD. Fields in primary keys are underscored. Tables drawn with dotted lines indicate a number of identical tables each holding values of specific data types.

painstakingly slow? Is it possible to create an intelligible user interface to this spaghetti of cross-referenced ID-numbers? Can you efficiently access and operate data, once it has been entered?

First of all it works. Surprisingly well, actually. It would be foolhardy to claim that I can depict any database design in GUARD, but I have tried it out on many different designs, and so far I have had no problems. In fact, database designs in archaeology are fairly simple, and I believe that the ones that will not fit will be few and far between.

Secondly, is it slow? In the beginning I feared this very much, but it does not seem to be the case. The potential problem stems from the way that every bit of information is atomised across many tables, and at the same time being overloaded with identification numbers. The latter surely has its price, as the size of any given database will be much larger in GUARD than in its original form. We are talking of a factor 3 or 4 in size. However, the numerous identity numbers, which are all indexed, help to maintain speed in searches. In fact the majority of searches associated with the user interface are done on individual tables and all searches are on indexed fields. The result is that the response time does not seem to grow significantly with the growing amounts of data. I have had the complete Danish SMR loaded on to GUARD taking up some 600 MB of space. In the *Entities* table there were more than one million entries, yet the result was merely a slight decrease in response time. Slowness, where it appears, is foremost related to the user interface itself and not to the table structure.

3. Design of the user interface

The user interface is the most complex part of GUARD. It is self evident that the abstraction level of the table structure does not allow forms to be drawn up in the usual straightforward way. It is necessary to write fairly extensive pieces of code to make it work. This is where the time investment in GUARD lies, but then of course the interface will work with any database design implemented.

Being a meta-structure GUARD has no inherent recording structure. A new database cannot record data before it has been structured. The structuring elements are themselves data, and they have to be entered before “real data” can be recorded. Hence the user interface has a number of forms through which it is possible to define entity types, relationship types, class structures, variables, lookup lists as well as setting links between variables and classes. To define a recording database in this way is very challenging and instructive. You have to be explicit in all your choices.

The main entry form presents you with the basic elements of GUARD in a straightforward way (figure 3). In the upper left-hand control the entity types are found. You only see those entity type relationships that have an existence dependency between them - shown as hierarchies in the control. The example in figure 3 is the Danish SMR with a county (“Amt”), district (Herred), parish (“Sogn”) and site number (“SB Nummer”) breakdown, and various entity types related to the site number. At present you cannot see if a relationship type exists between say “event” and “objects”, but if it has been defined you can open a form through which you can link instances of “event” with instances of “objects” using the relationship type.

In the lower left-hand control the entities are found. When we have a dependency structure as the one just described, we will also have dependency among the entities, where a number of entities will be the offspring of one entity of the independent entity type. Selecting an entity type thus sets a selection path only. To

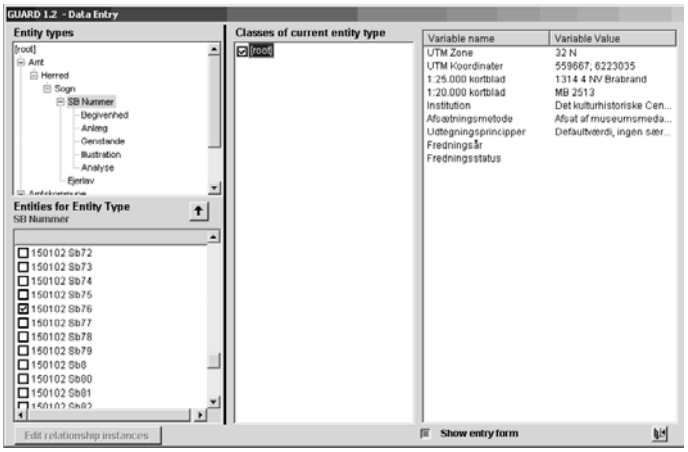

Figure 3: The main form of GUARD to access the data structure for data entry and editing.

reach a particular entity of the selected entity type we also have to traverse the hierarchical structure of the entities. Ideally this can be done in a tree view control as the one used for the entity types, but since the number of entities typically will run into thousands, the time it will take to build up the content of this control would be devastating. Instead a list view control has been chosen, where you start at the bottom of the entity type selection path, and from where you can select your way through the entity hierarchy by double clicking an entity, which will give you access to the next level, etc.

In the centre of the form the class structure of the current entity type is shown. In the example only the root class is present, but there might have been an extended hierarchy class present. In the rightmost control the attributes (here called variables) of the current class is shown, and if values have been recorded for the attributes these are shown as well.

You can enter and edit data in a form based on the attribute settings for a particular class (figure 4). The input form is highly standardised, and may appear rather primitive in its appearance. The problem is that there is no way of knowing in advance, what variables, and thus what data entry fields are needed, not only from database application to database application, but indeed from entity type to entity type, and from class to class. The solution is to use a form containing a large number of data entry fields related to the different data types available, organised into pages by way of a tab control. Whenever a class is selected the proper number and types of entry fields are activated, and their record sources are set to the proper slots in the database, while all unused fields and unused pages of the control are hidden. This all-enclosing data entry form works quite well despite its appearance.

From the main form you may call up a form in which to set entity relationships between the current entities and other entities (figure 5). In the actual example we are about to set this is represented by the institution responsible for recording the position of the current site. We have chosen the entity type “Institution”, we have chosen the action or role “has been marked by” and we can now choose the institution and link it. If there are attributes defined for the entity type these will appear in the lower right-hand control of the form, where they can be edited. The entity relationships that may be created between entity types can be of one to one, one to many or many to many cardinality. Naturally, relationship types within individual entity types may also be established.

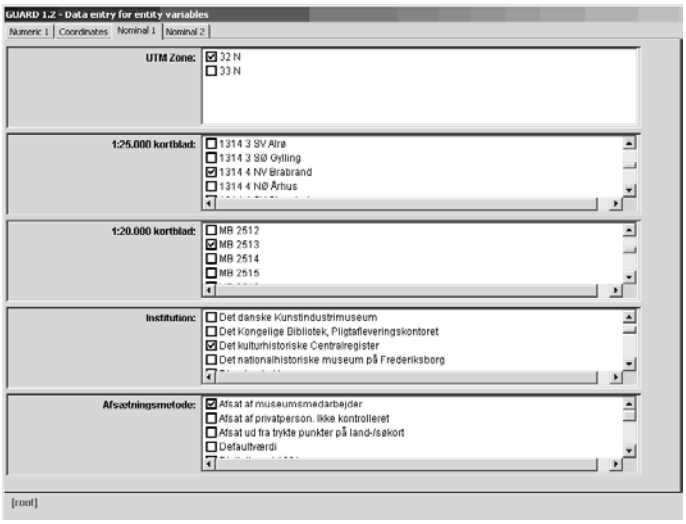

Figure 4: Pop-up form for data entry and editing values for entity variables.

Thus an intelligible and usable interface can be created. The same applies to search procedures whether for tabular output or for reports. Various experiments have been carried out and a standardised user interface for data search is currently being developed. It is too early, however, to say exactly how this interface will be organised.

4. Prospects of GUARD

GUARD may be used directly as a “production” database, or it may be used as a repository for already existing databases. My experience so far shows that it is fairly easy to migrate data from an existing database into GUARD. To do so you must be familiar, however, with the table structure in GUARD, and most of the migration process has to be run from VBA code modules. It will be possible to write some standard procedures and functions, but in each specific case it will be necessary to stitch them together according to the specific structure of the database we wish to import. Most standard research databases can be migrated with less than a day’s work, which is not very much if you consider the benefits gained from the common table structure. In the future it will be easier for the curators of these databases, and it will also be easier for future users to access the data, when all data-sets are accessed in exactly the same manner irrespective of their structure.

GUARD may thus be well suited for archival purposes, but the reasoning behind its development never was archival purposes. Basically, it was to counter the tendencies to dictate data standards in archaeology. I have previously argued that we may seek standards of form, which is exactly what GUARD does, but never standards of content (Madsen 1998, 1999a). We need our recording systems to vary according to the issues and problems. Research carried out according to predetermined common standards of what should be recorded and how it should be recorded will soon cease to be research. It will be mere reproduction.

Thus GUARD is aimed at making archaeological recordings flexible and versatile. A system that will allow users to create powerful database solutions with little effort and allow them to redesign at least part of the recording structure on the fly, as recordings progress, has great research potentials. At the same time, as the

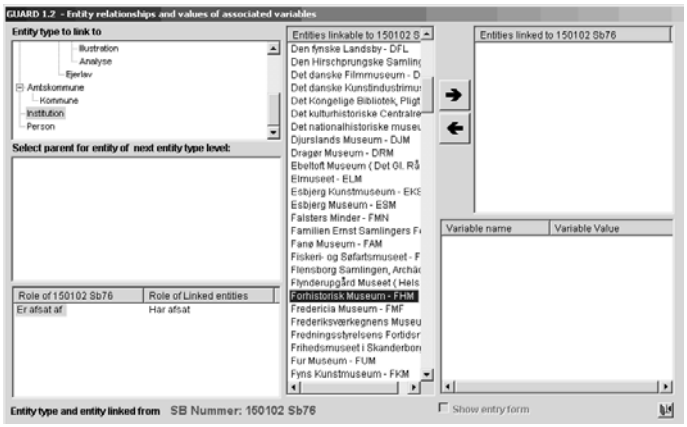

Figure 5: Pop-up form for setting entity relationship links, and to enter and edit associated relationship type variables.

structure of the database remains the same regardless of the structure and content of the recordings, we will hopefully be able to shut the mouths of those who claim that the only way to achieve compatibility is through conformity.

In developing GUARD for research purposes, special attention was paid to the potentials of using proper classification systems in connection with entity types. In most current applications, classification is a matter of choosing one class from a list of alternatives. Thus an entity can have a class assigned to it from a set of classes that are all at the same level. This is obviously not satisfactory. In GUARD a classification can consist of a hierarchical tree-structure with one root and as many branches and levels as needed. Classifications are thus of a classic monothetic divisive nature. At the moment I stick to this because, whereas the table structure allows polythetic classifications, I have so far found no satisfactory way to implement a polythetic structure in the user interface.

Within any classification tree you may choose to assign only one class to an entity, or as many classes as you wish. Further, you may operate with as many parallel classification trees as you wish. When, in the design phase, you assign an attribute to a class, all classes in the branches of the tree structure above it will automatically inherit this attribute. In this way it is ensured that all more specialised classes will hold the basic attributes of their parent class apart from what ever more specialised attributes they may posses themselves. If you do not want a class to share attributes with its parent class you may of course remove the attributes from it.

Classifications are operational. That is, when searching for data you may use the position in the classification tree as a parameter in you searches. You can specify that you want all of this class only, or all of this class and all classes lying in branches above it in the classification tree.

Another area that I have given some consideration to is the possibility of using the *Entity Type Relationship* and *Entity Relationships* to depict complex relationships. It is mostly relationships between entities of the same entity type that are of interest. Archaeology is full of such relationships. One example could be contexts with contexts in an archaeological excavation leading to statigraphy and the use of Harris Matrices to solve the complexity of this. Another example could be the fitting together of artefacts like pottery and flints and the use of graphs to elucidate the refitting patterns. A third example could be the association of decorative elements with each other leading to composition patterns. Recently I have used GUARD in my work with the latter problem, and I found a clear potential for systematic studies within an area that traditionally has been considered very difficult to handle (Madsen 1999b).

GUARD is a general utility system. Its primary aim is to enhance the quality of descriptive practice in archaeology. It offers a tool that gives the researcher a better chance of creating a description and recording system in accordance with the complexity of the problems at hand. The database solution attained, however, is one of a far greater scope than just providing flexible recording. The use of a meta-structure design points towards a standardisation of form in recording systems, detached from content, and hence it points towards the area of archival standards. I do not intend to suggest that GUARD is *the* solution, but I think it is an important step in the direction we need to take if we do not wish for everything to end up in either rigid conformity dictated by administrators on the one hand, or chaotic *ad hoc* use of databases on a primitive level and with a severe loss of information on the other.

References

ANDRESEN, J. and MADSEN, T., 1992. Data Structures for Excavation Recording. A Case of complex Information Management. In Larsen, C.U. (ed.), *Sites and Monuments. National Archaeological Records*. The National Museum of Denmark: 49-67

ANDRESEN, J. and MADSEN, T., 1996a. IDEA – the Integrated Database for Excavation Analysis. In Kamermans, H. and Fennema, K. (eds.), *Interfacing the Past. Computer Applications and Quantitative Methods in Archaeology CAA95*. Analecta Praehistorica Leidensia 28, Leiden: 3-14.

ANDRESEN, J. and MADSEN, T., 1996b. Dynamic classification and description in the IDEA. III International Symposium on Computing and Archaeology. *Archeologia e Calcolatori* 7: 591-602.

MADSEN, T., 1998. Digitaliseret registrering af udgravninger – er der brug for datastandarder og fælles strategier? Hansen, H.J. and Ødegaard, V. (ed.), *De Nordiske Museer og Informationsteknologien – rapport fra en konference 1.-3- december 1996*. TemaNord 1998: 513, Nordisk Ministerråd, København: 167-177.

MADSEN, T., 1999a. Digital recording of excavations: Do we need data standards and common strategies? Hansen, H.J. and Quine, G. (eds.), *Our Fragile Heritage. Documenting the Past for the Future.* København: 131-138.

MADSEN, T., 1999b. Coping with Complexity. Towards a formalised methodology of contextual archaeology. *Archeologia e Calcolatori* 10, 1999: 125-144

SCHOLTZ, S., CHENHALL, R.G., 1976. Archaeological Data Banks in Theory and Practice. *American Antiquity*, Vol 41, no 1: 89-96.