ArchaeoInfo

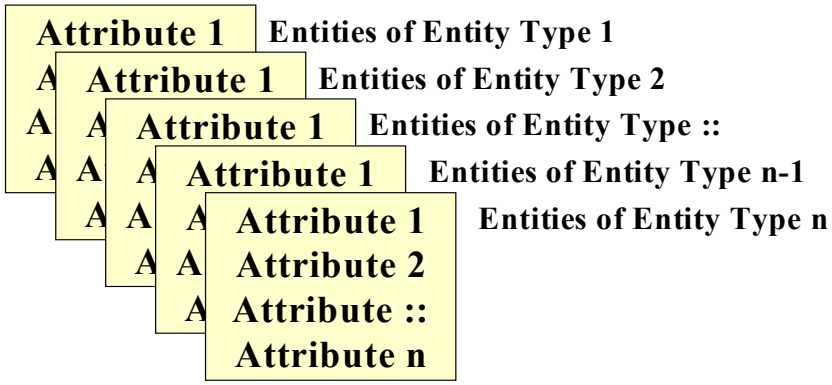
An object-oriented information system for archaeological excavations

Torsten Madsen

When I started in Archaeology, recording of excavations was simple. We drew plans and sections, numbered the drawings and their content, and wrote down a description of all numbered objects in a notebook. In a separate notebook, we kept a list of finds with coordinates and/or cross-references to objects on drawings. In a third notebook, we wrote all our remarks, and that was that.

With the addition of a numbered list of features, this is still how most excavations are run in Denmark. Paper is grateful, if you do not become too formalistic about it, and what ever complexity lies in your recordings, it is easily absorbed in the prose style.

The relational world is extensive

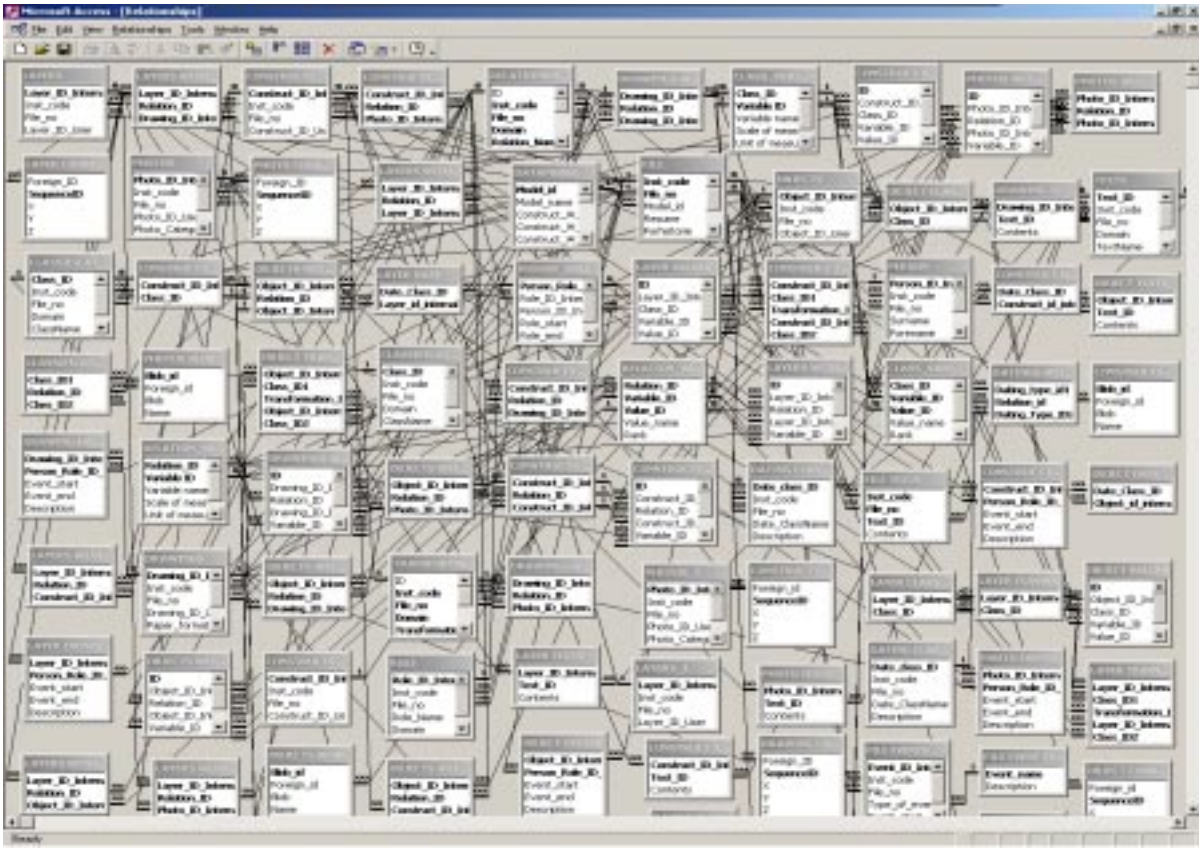


- The world consist of Entities
- Entities are grouped into Entity types
- Entities of an Entity type must have mutual attributes
- Attributes must be non-composit in nature
- Normalisation is obtained by splitting Entity types into new ones
- The more detail, the more Entity types are created, each of which should occupy a separate table

When we use databases for digital recording, we have to be formalistic, and if we use relational database theory, we are quickly drawn into a world of complexity. We have to dissolve the universe of observation into logically coherent units, the content of which must share mutual attributes.

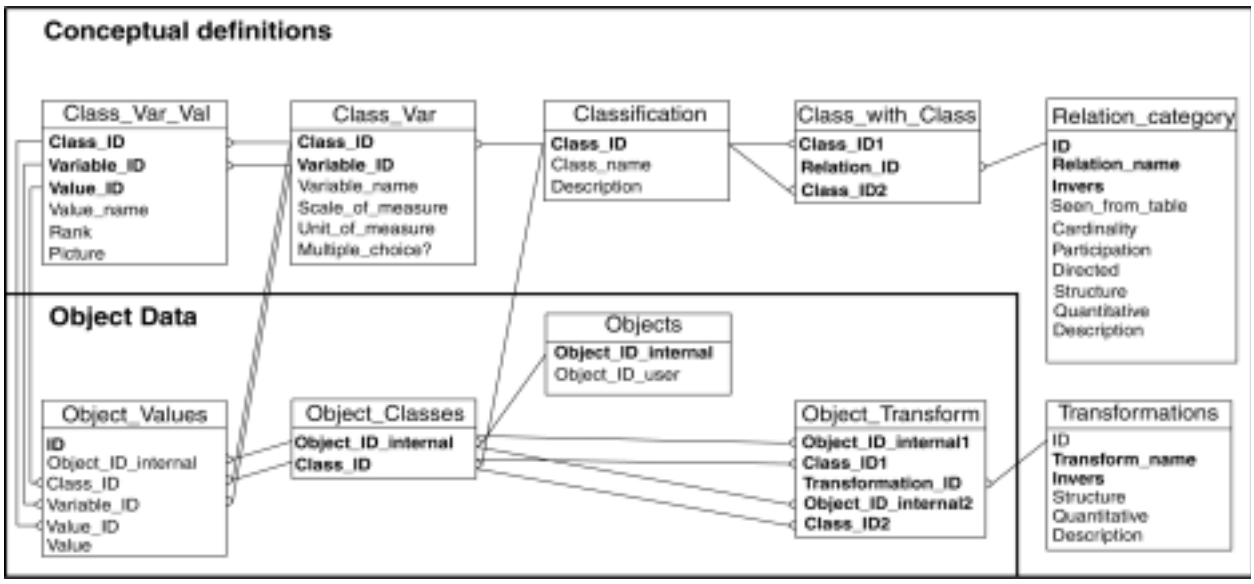
Starting at the core of excavations we find different types of observational units like contexts, features, finds, artefacts, drawings, photos, etc. Further, the observations within the units can be of different types. Contexts can be deposits, cuts and structures for instance, and finds can be bulk finds, single finds, samples, etc. Each of these types are characterised by different attribute sets, and are thus different entity types. The number of context and

finds types may be limited, but when it comes to features, not to speak of artefacts the number of types become endless, and each type, possessing its own unique set of attributes, should have its own table. This does not work.



In the mid nineties, I worked together with Jens Andresen on an excavation recording system called IDEA based on relational database theory. Quickly it became apparent to us that the numerous types posed a problem that could not be handled in any direct way using standard relational database theory. We came up with a solution that did solve the problem. It was an object-oriented solution, even though we spoke of meta-structures.

Object-oriented aspects of IDEA



Arceologia e Calcolatori 7, 1996, p. 595

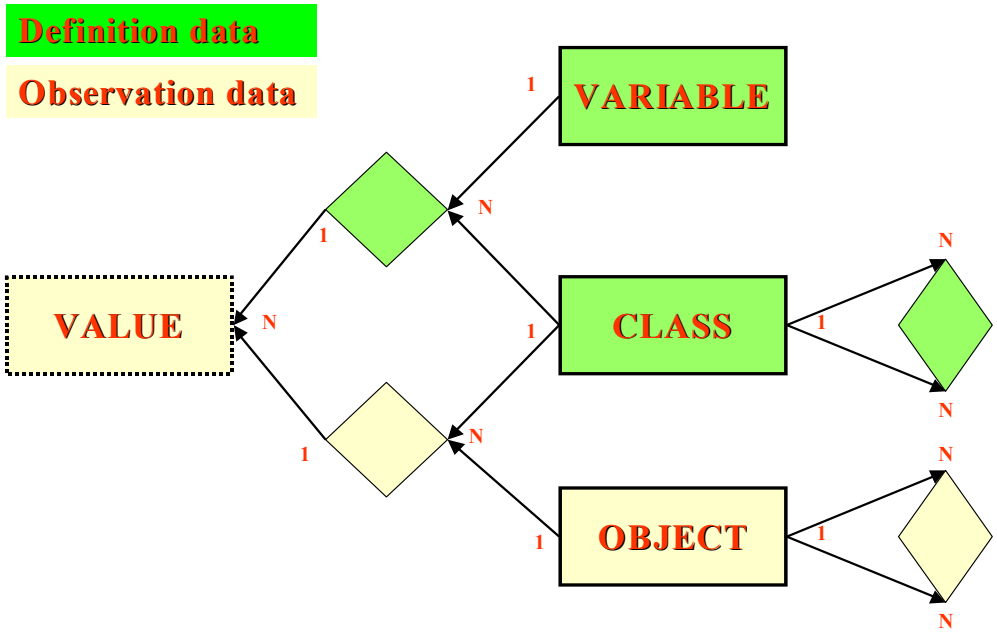
The core of IDEA was based on relational theory, however, and it was simply too complicated to be maintained and developed further due to the many cross-related tables. It never went beyond its version 1.0, and in an assessment of it, we concluded that it would be necessary to start from scratch at a higher level of abstraction, and that Object Orientation would provide the answer.

The object oriented world is simple

The world consist of uniquely identifiable objects	OBJECT
Each object belongs to a class that is part of a class tree	CLASS
Each class can be described by variables	VARIABLE
Each variable can attain values	VALUE

Object Orientation starts with the assumption that the world consists of objects uniquely identified in their own right, and not through membership of some group like Entity types. Classes formalised in tree structures with or without multiple descent are assigned to objects, and each class is characterised by variables. There are some important properties associated with Object Orientation such as encapsulation of methods with objects, the use of abstract data types, and single or multiple inheritance of methods and variables through class trees. Dedicated databases that support all the rules of object orientation exist, but it is to a certain degree possible to implement object-oriented databases in a relational DBMS.

OO-structure in a relational DBMS



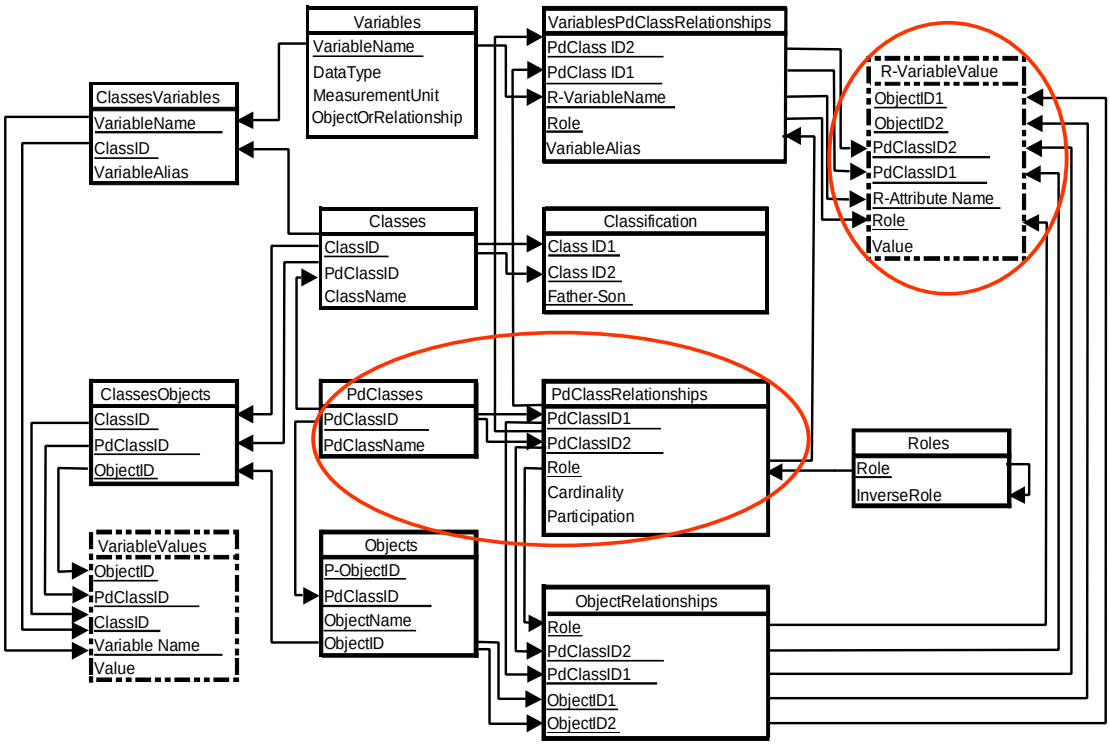
The database design for object orientation is very simple, which is why, I believe, the approach has proved fast and flexible. Presenting the design through an Entity Relationship diagram, we note that there are only four entity types: Objects, Classes, Variables and Values. Classes and Variables are definition data, setting up the class hierarchies and their characteristics, while Objects and Values are observation data. The relationship type structure is simple. Classes are linked to classes to provide the class trees. Objects are linked to objects to provide possibilities for cross-references between all objects. Classes are linked to Variables to provide the classes with descriptive characteristics. Objects are linked to classes, and finally values can only be entered when linked to a combination of objects, classes and variables.

You should note that Values cannot be represented through a single table. It is necessary to have one table per data type in the DBMS used. It is also possible at this point to introduce a sort of abstract data types.

There is no immediate way to encapsulate methods with objects, and integrate inheritance in class hierarchies with the data structure. To add these facilities to the system, we have to do it through the user interface. Indeed the user interface is the major problem and perhaps the major weakness associated with an object-oriented approach within a relational DBMS. The coding necessary to create a useable interface is extensive. It takes time to create and maintain the code, and even if there is full data integrity in the database itself, there is always the possibilities of a snag hidden somewhere in the code making what you get in say a cross tabulation not exactly what you should have got.

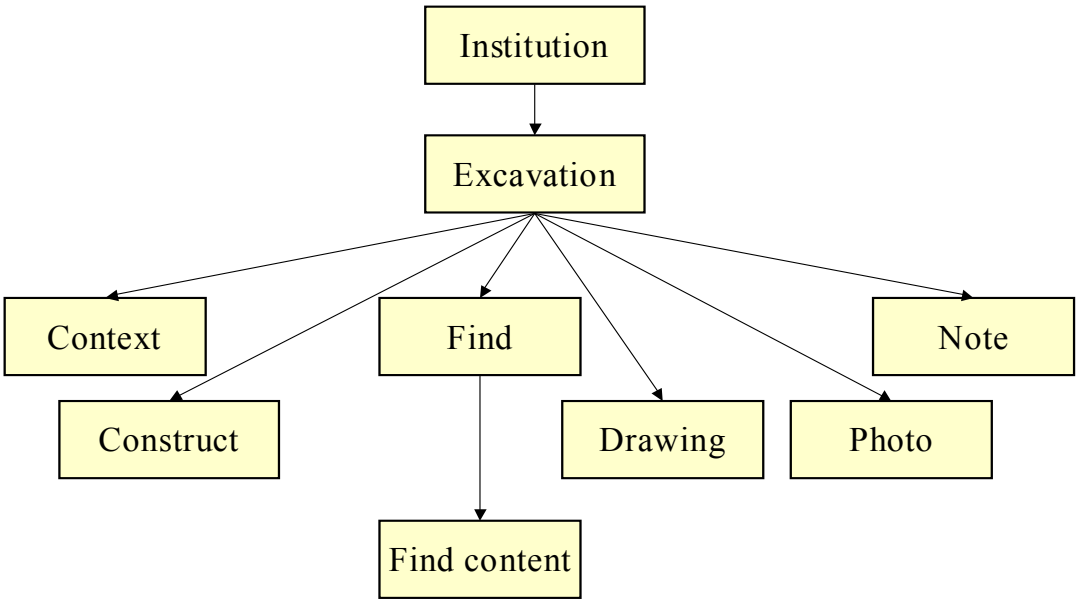
I began to work seriously with object-oriented solutions within relational DBMS in the late nineties, first exploring them in general, and then over the last 2½ year specifically aimed at the creation of an information system for archaeological excavations. This system has been baptised ArchaeoInfo, and the second half of this paper will deal specifically with this system.

OO-table structure in ArchaeoInfo



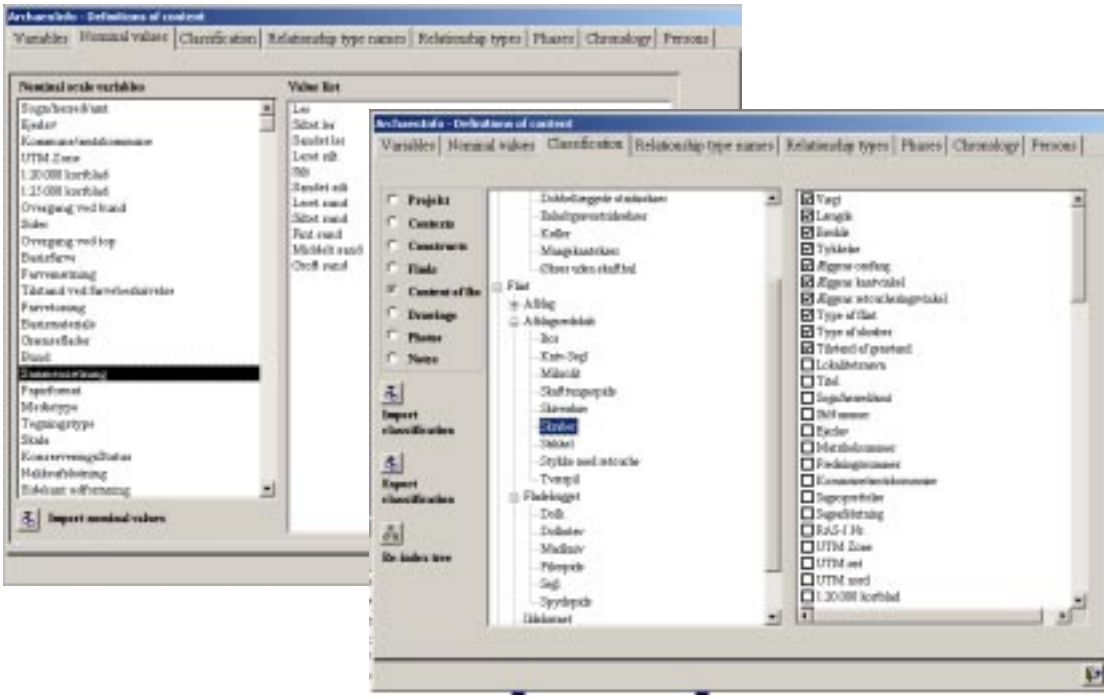
The database design of ArchaeoInfo is slightly modified compared to the previous slide. Classes are split into predefined classes and classes properly. The predefined classes are hard coded so to speak into ArchaeoInfo, whereas classes are entirely user defined. I believe I had some good arguments for splitting this information into two tables, but I have forgotten what they were. The other modification is that variables and values can also be associated with relationships between objects.

Predefined classes in ArchaeoInfo



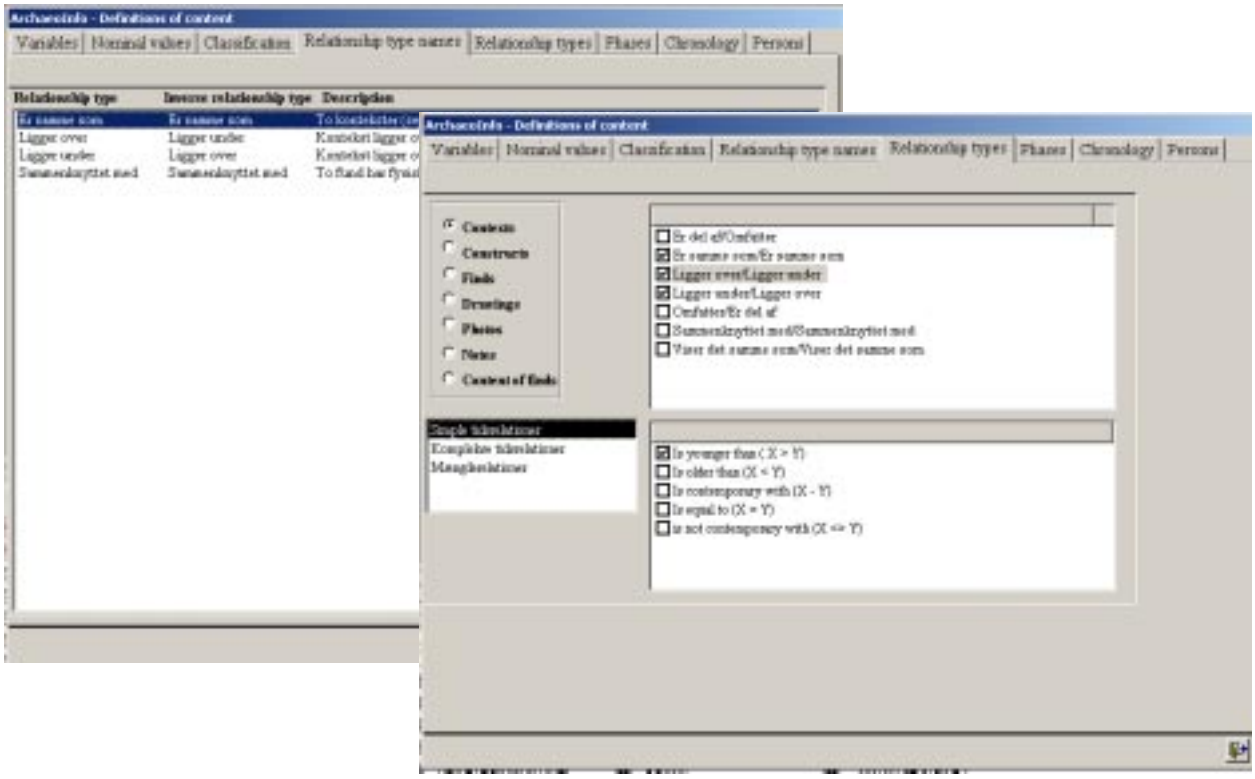
There are eight predefined classes: Institution, Excavation, Context, Construct (most would probably think about this as Feature), Find, Find Content, Drawing, Photo and Note. There is a predefined class hierarchy, where excavation is a subclass of Institution, where Context, Construct, Find, Drawing, Photo and Note are subclasses of excavation, and Find Content is a subclass of Find. There are also a few predefined variables attached to these classes. The rest of the content structure is user defined.

Defining variables and classifications



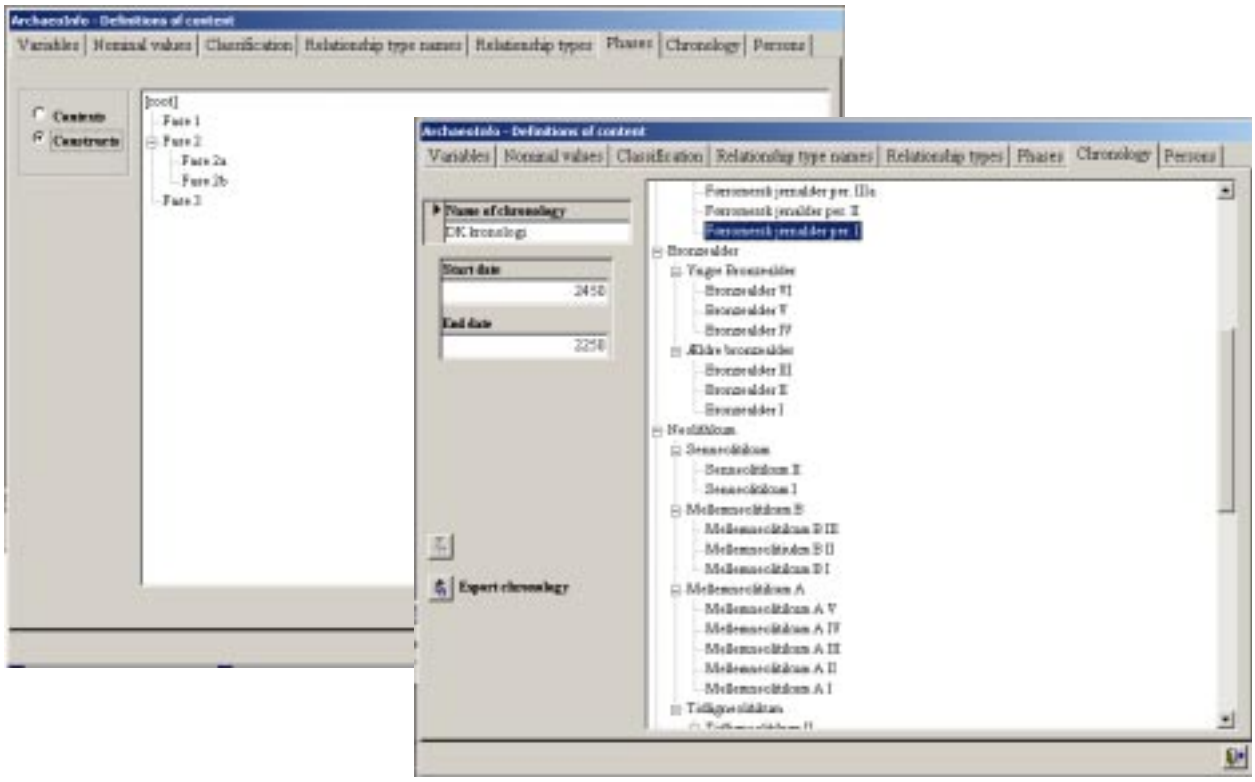
Central to ArchaeoInfo is the definition of Variables and Classes. Variables can be of different data types, and when Nominal scale variables are created, their values can be typed in or imported from external sources. Hierarchical subclass trees of the various predefined classes can be created, and variables can be associated with the individual classes. Rules of inheritance of variables apply to the class trees. You can export and import named sections of class trees including the associated variables to and from a special database, facilitating the creation of class trees in new databases.

Defining relationship types



You can define relationship names and add them as relationship types to the predefined classes. All objects of the predefined class can subsequently be cross-referenced using these relationship types. If the relationship types imply a logical or chronological relationship, you can attach predefined signifiers to the relationship types to make them operational according to their logic implications.

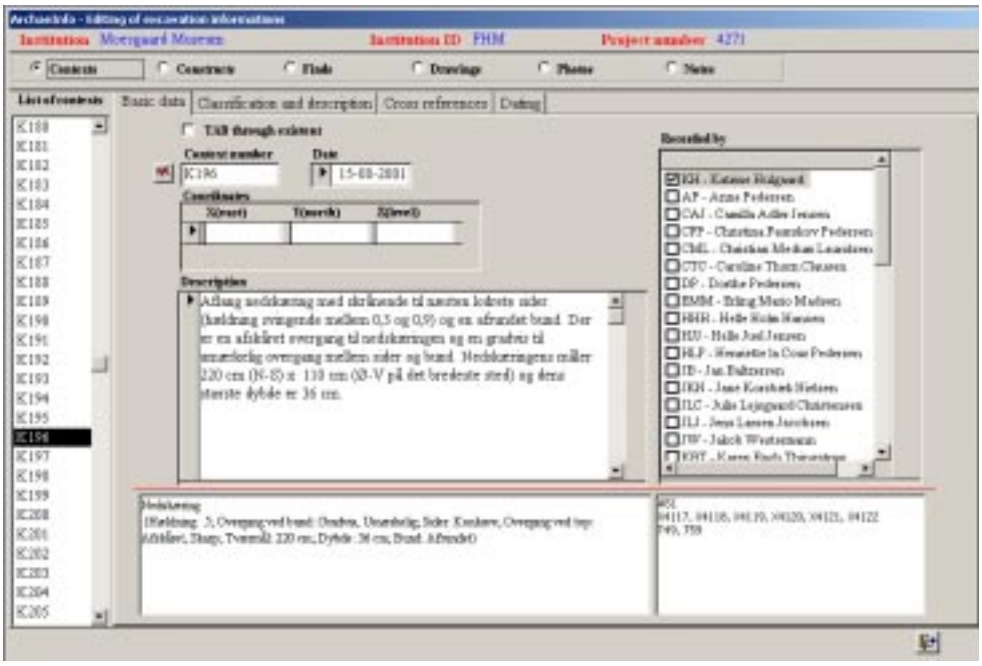
Defining phases and chronologies



You can define hierarchical phase division for contexts and constructs and in this way build up a chronological model of a site. You can also define a regular chronology, and apply this to your findings. In this the individual periods must have a start and end date on a scale stretching backwards – a BP scale if you wish. This is used to keep the chronology tree correctly sorted, and can be used in searches.

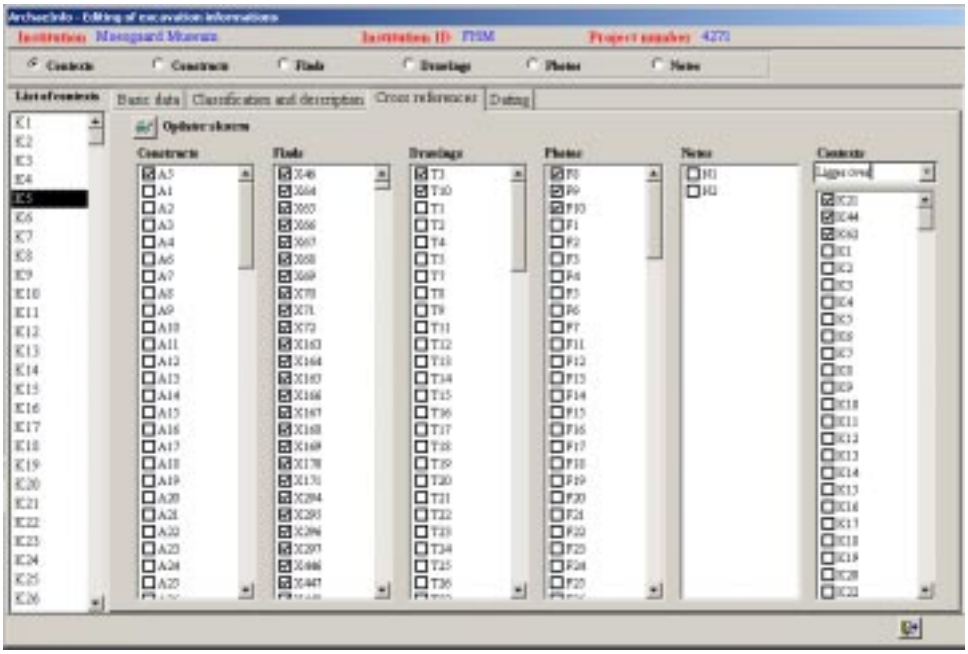
To facilitate the creation of a new excavation database, ArchaeoInfo can transfer the complete definition structure from one database to another. You can thus have template databases, which are ordinary ArchaeoInfo databases containing definitions but no data.

Recording by form, basic data



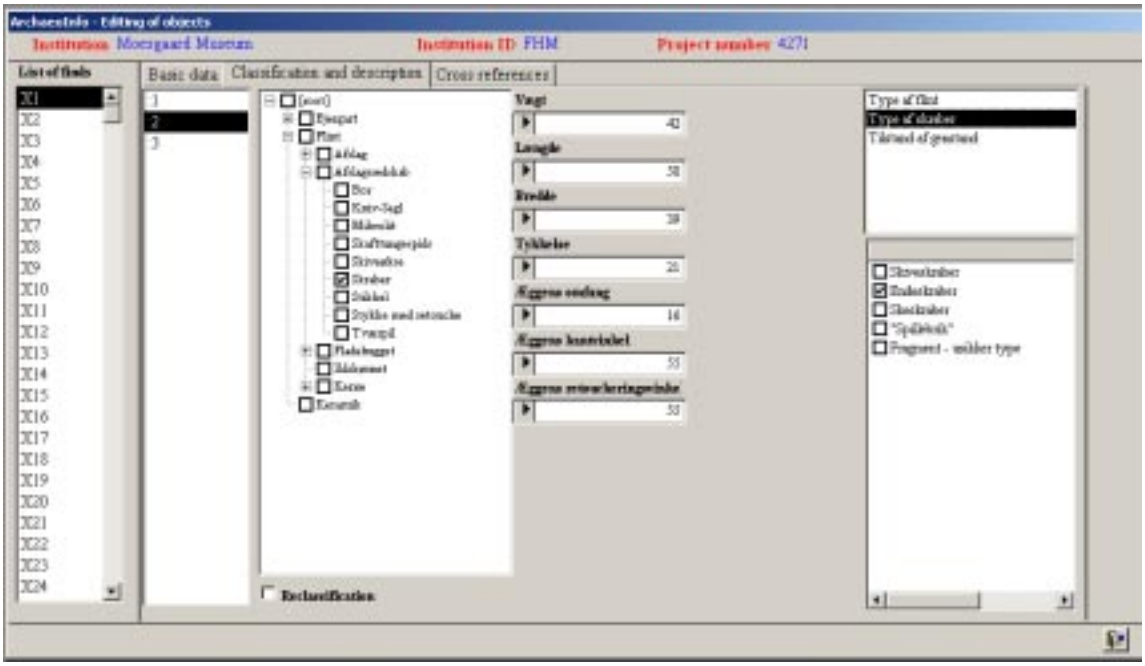
Data recording takes place through three forms. One covers information about the excavation itself. Another covers information about Contexts, Constructs, Finds, Drawings, Photos and Notes. The third covers information about Find content. Each form has a varying number of tabs. The central one is Basic Data, where you can enter new objects together with information in a few predefined variables.

Recording by form, cross references



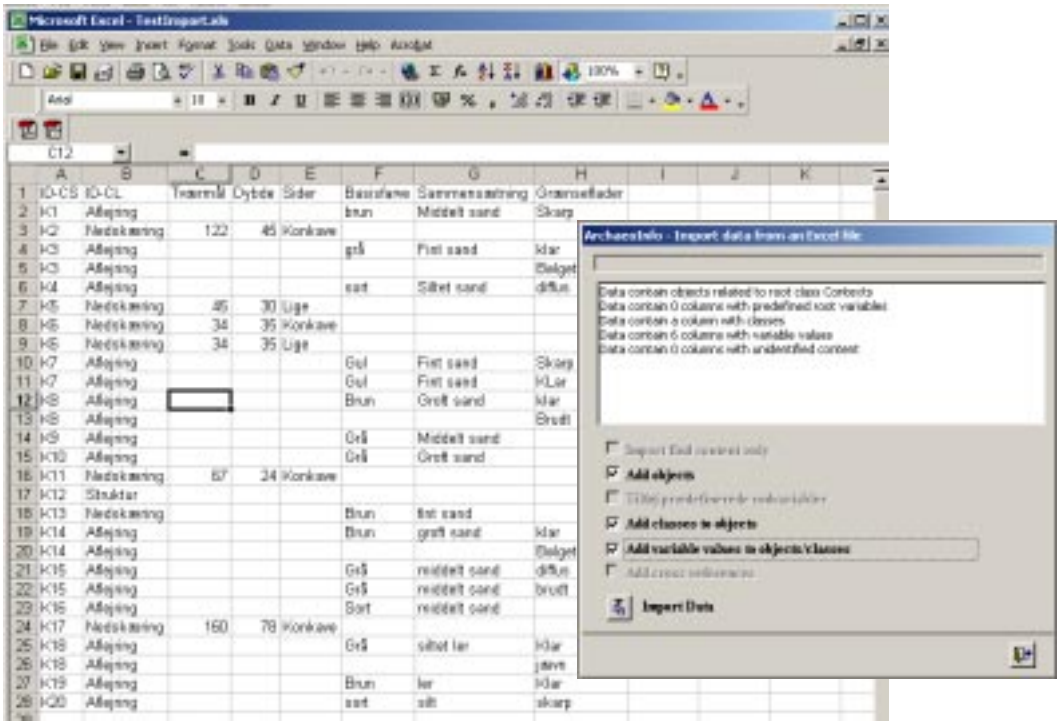
On the Cross-references tab, you check mark objects to cross-reference these with the current object of the form. When setting cross-references between objects within the same predefined class it is necessary to choose the relationship type first, as there may be a number of user defined relationship types here.

Recording by form, classification



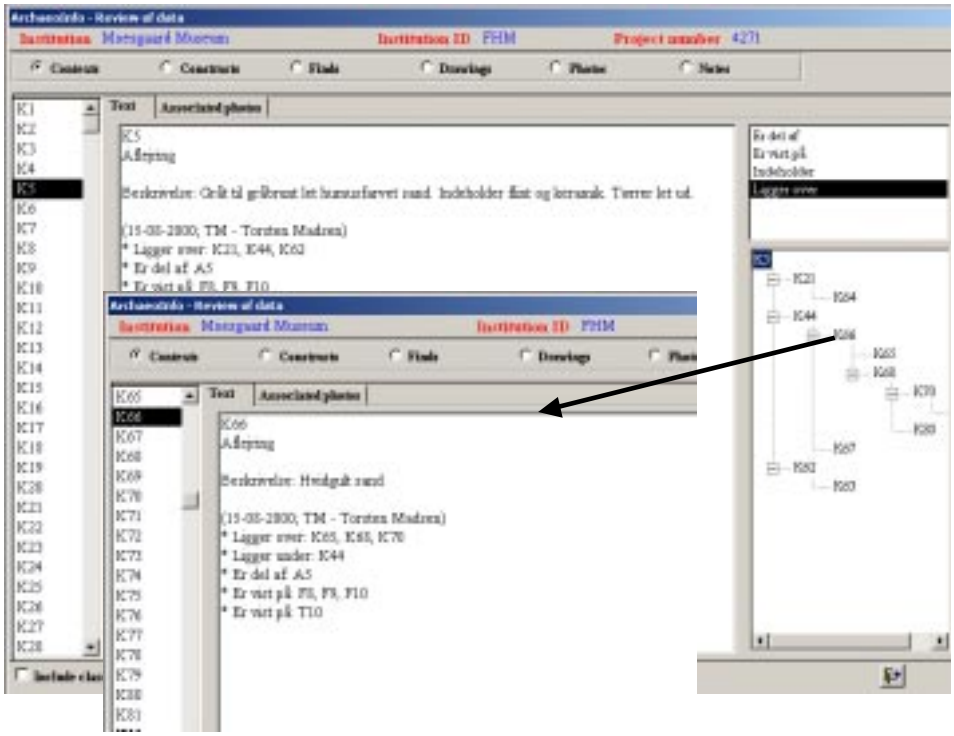
The recording of classes is also done by check marking. When a class has been checked the associated variables appear on the form divided into those, where you have to type a value, and those nominal scale variables where you have to choose between a number of preset values. Multiple choices are fully supported in the nominal scale values.

Import from external sources



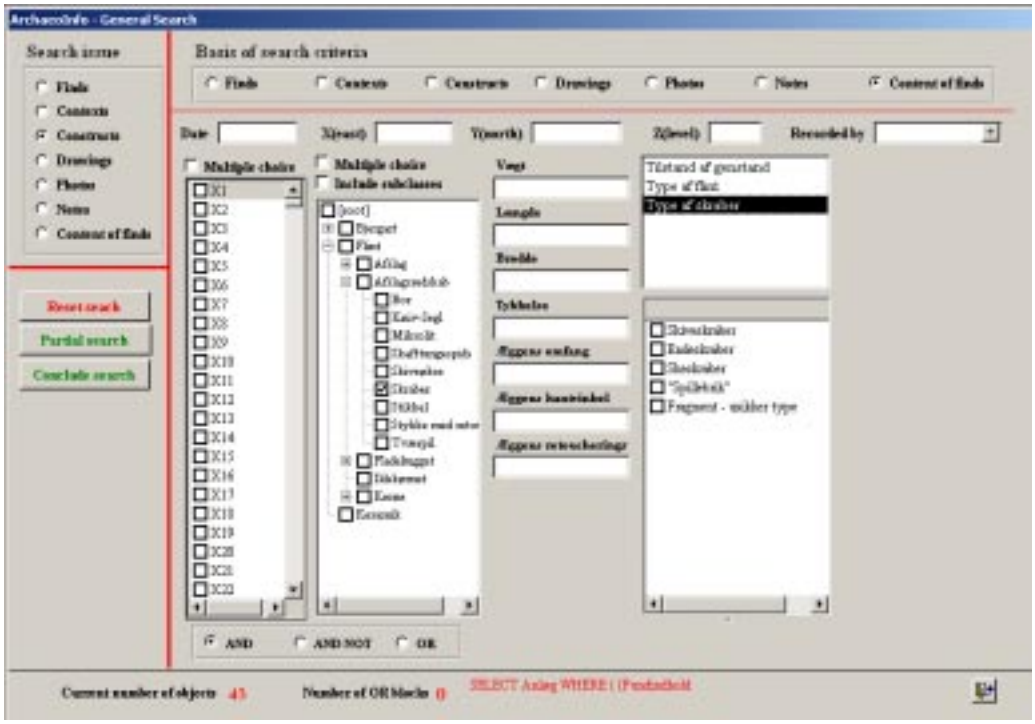
It is also possible to import data from external sources. These sources comprise Microsoft Excel worksheets, Microsoft Access tables and delimited text files. There are some naming rules for the columns that have to be followed, and the imported data must fit the existing definitions in order to be accepted.

Viewing information on screen



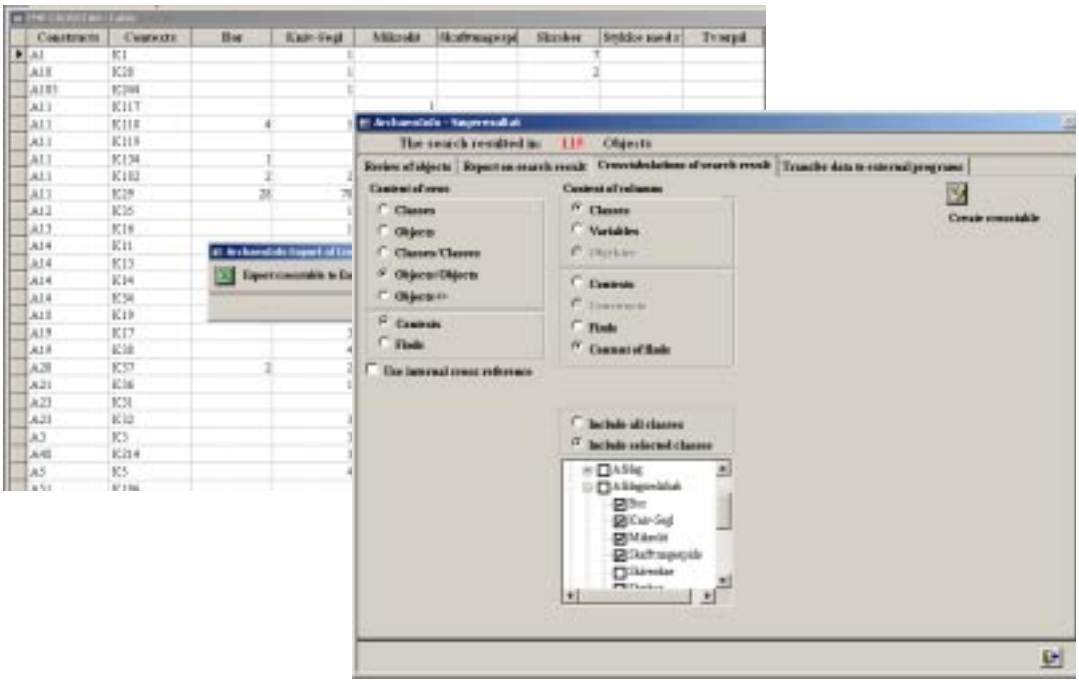
There are different ways of data retrieval. One of these is to view the individual objects with all their associated information on the screen. To the right a listing of cross-references organised by relationship type is available. If the relationship type refers to cross-references within the current predefined class, the complete chain of references from the current object is shown. If you subsequently click on one of the referenced objects the focus of the screen view moves to this object.

Searching for information



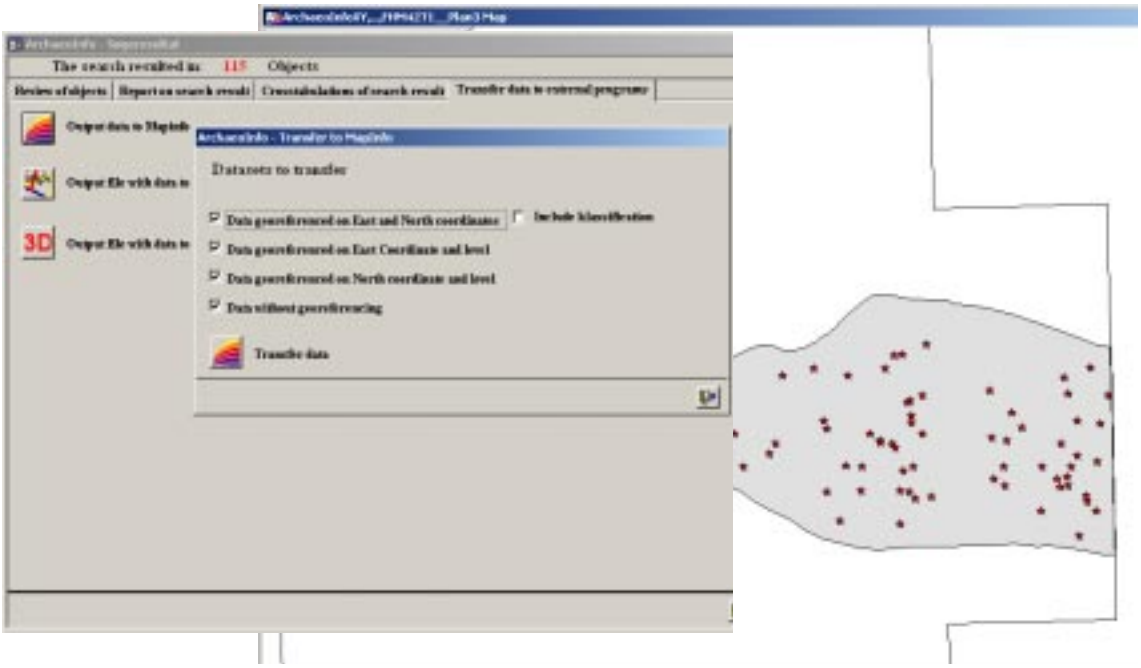
You can create searches using any information entered into the database. The search form is complex, and it takes a little practice to get used to it, but it is immensely powerful. You can search on the class hierarchy checking just one class and have all subclasses included. You can search on variable values using operators like equal to, larger than, and smaller than. You can use relationship types in your searches, and you can shift between the logical operators AND, AND NOT and OR. In the lower part of the form, you can follow how many objects are found in the search so far. In addition, through a pseudo SQL string, you can see what you have been searching for.

Crosstabulation of search result



When you conclude the search, you are given several possibilities for handling the search result. You can look trough it on the screen, you can print out a report, and notably you can also create cross tabulations. This facility is complex, but indeed very versatile and powerful. You can create almost any imaginable pivot table. The tables you create can be exported to Microsoft Excel for further analysis.

Transfer of data to MapInfo



A further possibility is to export the search result for use in other programs. This aspect is currently being developed, but among other things you can send data to MapInfo using an ODBC connection. If you use MapInfo for your excavation plans, it is a very fast way to plot your search results onto these plans.

Finally, you can write out reports. Direct printing is not supported. All output is send to text files with html tags. This makes it possible to produce an individualised report, where at least partly you can determine the ranking of the different elements, the start of new paragraphs, if text elements should be in bold and/or italic, and what font size to use.

The current version of ArchaeoInfo is 1.5. This version will be maintained and updated when necessary, but otherwise kept more or less, as it is. Work on a version 2 is about to begin. In many aspects, not least in the database structure, it will be different from version 1.5, and it will take a year or two before it appears in its first test versions.

Version 1.5 may be downloaded from the Internet at the address http://fc.hum.au.dk/~farktm and used freely. The user interface is currently in English and Danish. Other languages can be added. It is simply a question of finding somebody willing to do the translation. There is a very comprehensive manual in Danish, currently being translated into English. Meanwhile there is a very short getting started description available in English. There is currently no online help in ArchaeoInfo, but it will be added in some form when time allows.

Writing out reports in Html format