Oleg Missikoff

Application of an object-oriented approach to the formalization of qualitative (and quantitative) data

1 Introduction

'Archaeology is the discipline concerned with the recovery, systematic description and study of material culture in the past.' (Clarke 1968)

This description of the targets of archaeological research, given by David Clarke in the fundamental 'ANALYTICAL ARCHAEOLOGY', allows us to single out the three main phases of the approach to the material culture in archaeology: the recovery phase, represented by surveys and excavations; the systematic description phase, operated through the classification of the data obtained from surveys and excavations; and the study of the results deriving from the two preceding phases, in order to obtain a deeper comprehension of the social, economical, and technological development of a certain area over a certain period.

The present work is oriented towards an enhancement of the conceptual and methodological tools for the management of the second phase which, compared to the others, shows a much lower degree of maturity. The problem appears even more urgent if we think that this phase occupies a conspicuous share of the archaeological research. In fact, according to K. Chang (1967), 'it is reasonable to estimate that 80 or 90 percent of an archaeologist's time and energy is spent in classifying his material.' The acknowledged importance of this aspect of research has led, along the history of archaeological studies, to the production of a large number of publications oriented towards the design of a methodological approach which could be universally accepted. Unfortunately, unlike what happened, for example, to the techniques of archaeological excavations, any such attempt has, sofar, inevitably failed.

The target of this work is to carry forward a new proposal which, supported by advanced tools of analysis borrowed from the information technology and specifically designed to perform this task, could guarantee a reasonable chance of success. The paper is structured in two main parts: the first part shows a synthesis of the history of classification theory in archaeology, for providing a frame of reference for the problem; this synthesis is followed, in the second part, by a description of the informatic tool the use of which is proposed, together with a brief example of the implementation of this tool, to a case study.

2 A historical outline of classification in archaeology

With the benefit of hindsight it is possible to identify four main phases in the history of archaeological thought concerning classification problems. For a better understanding of the fourth phase, currently active, it is necessary to give a brief examination of the key points which have characterized the previous three.

2.1 THE 'INTUITIVE' PHASE

The first phase, defined as 'intuitive', represents the period of the history of classification in which the artifact analysis is performed on a totally empirical base. This phase, the higher expression of which is represented by the work of Oscar Montelius (1874, 1885, 1899), was characterized by the production of increasingly refined typologies, but without any need, from the archaeologists, to explain the principles on which the typologies were built: 'Montelius was not concerned in a methodology for determining types, whose existence he implicitly accepted' (Klejn 1982).

2.2 THE 'SUBJECTIVE' PHASE

The second phase is represented by the emergence, in the early thirties, of the awareness of the fundamental importance of a clearer exposition of the principles on which typologies were based (Gorodzov 1933; Kluckhohn 1939). During this phase typology is recognized as a substantially subjective operation (Brew 1946; Krieger 1944), but the lack of conceptual tools able to formalize a qualitative-based approach led to a general dissatisfaction with the traditional model and, consequently, to the search for a new paradigm which could offer a greater warranty of formalization. In fact the subjective phase, though having recognized the importance of an explicit formalization of the constituent elements of typologies, failed to provide a classificatory paradigm able to handle the problem.

2.3 THE 'POSITIVIST' PHASE

That paradigm has instead been envisaged, by the supporters of the mathematical approach to artifact study and classification, in the numerical codification of attributes

to be analysed by means of statistical techniques (Bordes 1950; Brainerd 1951; Ford 1949; Robinson 1951). Imported from natural and social sciences, where they had been utilized with success, statistics seemed to offer what archaeological research needed: a tool of formal analysis which used standardized and explicit mechanisms, thus ensuring a greater comprehension of the conceptual paths followed by analysts and, consequently, a possibility of verification and replication of the single analysis.

The fifties provided a stage for harsh epistemological disputes between the supporters of the traditional approach (Ford 1954) and the proponents of the new techniques of analysis (Spaulding 1953). These disputes ended with an explosion of popularity for the quantitative approach, leitmotif of the third phase (Binford/Binford 1966). Though moving from a sound principle, for a number of reasons, this approach has proven of a little practical use in archaeological research.

2.4 The contemporary scene

Since the loss of popularity of the quantitative paradigm (Aldenderfer 1987; Christenson/Read 1977; Thomas 1978), archaeologists have become very cautious in dealing with the problems of classification (Seitzer 1978) and, consequently, very few have been the proposals of new or revolutionary approaches to classification (Kampfmeier 1986). In fact the current phase has not yet produced an algorithm which could have enough impact to characterize it.

At this stage it appears necessary to stress two key points which constitute the conceptual base on which the present research has been structured. The first point concerns the traditional approach that, if failing at a theoretical level for its poor possibilities of formalization, has shown a demonstrable empirical value and does not need any improvement at a technical level, in fact, as Thomas has put it: 'To propose a computer technique for deriving morphological types presumes that traditional methods have failed, and nobody has demonstrated that yet' (Thomas 1978). The second point refers to the need, stressed by the proponents of the quantitative approach, of a tool of analysis able to produce formal descriptions: such a need is even more impelling now than it was thirty years ago.

The conclusion drawn from these considerations is that an integrated reformulation, thus allowing the formalization of typologies operated through a traditional methodology, could offer a reasonable possibility of solving the paradigmatic disputes which have characterized more than sixty years of typological debate.

The present work represents an attempt to produce a synthesis between the two approaches. In fact the model presented hereafter agrees with the definition of typology as a subjective task but, at the same time, believes in the necessity for a rigorous formalization of the principles on which each typology is based. The target therefore results in a subjective but formal and explicit approach to artifact typology. This paradigm is made complete by the support of a tool of analysis which has been borrowed from computer science.

3 The 'Mosaico' Project

The 'Mosaico' Project has been developed in Italy within the CNR (National Research Council), and consists of an environment for conceptual modelling according to the paradigm Object-Oriented (Coad/Yourdon 1991; Khoshafian/Abnous 1990). This modelling, performed using the formal language TQL++ (see sec. 6), has the structure of a knowledge base (KB). The system allows the formal description of any kind of entity using attributes both qualitative and quantitative, whatever the epistemological position of the user. It is however necessary to exhaustively explain the conceptual path chosen for the entity definition.

Mosaico assists the analyst in the formal and correct description of the application domain, operating a syntactical and semantical verification of given definitions. Following this procedure it is possible to avoid those little 'arrangements' performed by authors in the presence of practical inconsistencies of typologies not perfectly formulated at a theoretical level.

4 Terminological specifications

Before going any further, it is necessary to clarify some key points of this method.

To begin with a clear definition of the three key words mentioned above, Typology, Taxonomy and Classification, will be given. These terms are currently used in archaeology but their meanings vary according to the different epistemological positions of the various authors, for that reason the definitions given here are drawn from Artificial Intelligence.

- Typology: The word 'typology' specifies the definition of a conceptual entity, the 'type', which describes a group of similar phenomena, the 'class', by means of a number of attributes considered to be 'significant', together with the type and range of values that those attributes can assume in order to consider a certain phenomenon as a member of that specific group.
- Taxonomy: In the real world phenomena do not have isolated lives, rather they are organically connected with other phenomena by a network of hierarchical relationships. These relationships can be of various kinds, but for what concerns the present work, just one type will be taken into consideration: the Generalization/

Specialization relationship (also called 'ISA' relationship). The word 'Taxonomy' is used to indicate the organization of gen/spec relationships between all types within a certain application.

– Inheritance: As previously stated, the various entities are interconnected through a network-type organization of generalization/specialization relationships forming a hierarchical structure, graphically expressed by an inverted tree. This has at its root a very general type which is then refined using two mechanisms: 1) Restriction of the range of values for one or more attributes; 2) Addition of new attributes. The mechanism of inheritance simplifies and speeds up the definition of more specialized concepts further down the tree. In fact the ISA relationship requires only to show the differences from the more generalized concept as all the rest are equal by default.
– Classification: Compared with the two concepts explained above, the term 'classification' appears very easy to define. It indicates the operation of assigning a phenomenon (be it an object, a decoration, a culture, or else) to a certain class by matching the types and values of its attributes to the types and ranges given in the definition of types.

5 Description of the system

The target of Mosaico is to support the designer in the formal and correct specification of an application domain and in the rapid prototyping of the application software. The fundamental component of the Mosaico System is the 'type' intended as the abstraction of a group of objects in a particular application domain. In other words an archetypal representation describing common aspects of individuals belonging to the same group. In Artificial Intelligence types are also referred to as 'entities'.

In Mosaico, the KB is organized using a hybrid methodology: 'Frames' are used to represent concepts and a 'Semantic Network' to define relationships between frames (Colombetti 1985; Giarratano/Riley 1989).

In structuring a KB much like a database we have two main levels of organization: the schema definition and the input of actual data. In the Mosaico environment the two levels are referred to as 'intensional' and 'extensional' respectively. At the intensional level the entities are described by listing their characteristic properties and defining the hierarchical relationship between each other. At the extensional level, on the other hand, are stored the instances of the entities represented at the intensional level. The instances are entered by associating values to the properties listed in the corresponding entities.

The operation of type definition is performed by using a conceptual language specifically conceived: 'TQL++'.

6 'TQL++': a Conceptual Modelling Language

TQL++ (Type Query Language). Because of the complexity of the whole model we will present just the aspects more likely to be useful for the needs of an archaeologist attending to artifact classification.

TQL++ has been conceived for the description of the entities of specific application domains; the static definition of a type is structured in five main sets:

– Structural Specification
– Properties Typing
– Integrity Constraints
– Hierarchical Organization
– Type Specialization

6.1 Structural Specification

The type structural specification consists in supplying the list of properties and, for each of them, the type corresponding to the values that they can assume.

<type-name> := [<prop-name> : <prop-type>
<prop-name> : <prop-type>]

Properties can be single or compound: a single property can have just one value associated to it; a compound property can be either 'multivalued' or 'structured'. In the first case it is possible to associate more than one value to it, whereas in the second it is not possible to associate any value directly to it. Structured properties have 'subproperties' and values will be associated with them (unless they are structured themselves as well).

6.2 Properties Typing

A property type is used to show the kind of value it can assume when ascribed to an application object. The properties of a hypothetical type of vase, for example, could be: 'id' (number for object identification), 'max_h' (for maximum height), 'max_w' (for maximum width), 'dec' (for decoration). Those properties shall be typed showing that the values for 'id' should be integer numeric values *(integer)*, those for 'max_h' and 'max_w' should be real numeric values *(real)*, whereas for 'dec' the values should be indicated through a string of characters *(string)*.

The indication of a property type *(typing)*, can either use a base type such as *integer* or *string*, or it can be more precisely specified by the user through some property typing tools. Here follows the description of the two simpler property typing tools offered by the type specification language TQL.

Listing: Through a listed set, it is possible to explicitly indicate allowed values for a given property (this construction is used mainly with categorical variables). For

example to the property 'dec' we can associate the values: 'painted' or 'scratched'; this implies that for any object of the type 'vase', the property 'dec' can assume just one of the two indicated values:

vase := [..., dec: (painted, scratched), ...]

Range: Like the previous case this construction allows the indication of the values that a given property can assume, but without listing them all. The present construction can be used for continuous variables, in which case it suffices to indicate the minimum and maximum values allowed (it is also possible to use the extreme values):

vase := [max_h: (3...150), max_w: (5...80)]

Tuple: Typing through tuple is requested when a property is structured. In this case the property is defined by its subproperties, referred in the associated tuple:

vase := [..., dec: [lip_dec: string, body_dec: string, base_dec: string], ...]

Sofar the possibilities have been listed that the language TQL++ offers to type the properties that define the information structure of an entity, and consequently of the objects associated to it, which all together form the corresponding *class*. In the following paragraph, the possibilities of imposing constraints in the phase of type definition will be shown. Although those constraints may be of different sorts, they will have to be respected by all object introduced in the KB.

6.3 Integrity Constraints

TQL has been conceived with a particular consideration for integrity constraints and it appears to be very powerful in this respect. As stated above, because of the complexity of this language, many of the TQL features will not be mentioned in this exposition but, for sake of clarity, will be limited to the following:

i. Typing constraints: As already explained, giving the type of a property implies itself a limitation to the values that the corresponding objects can assume. Thus, having indicated that, for instance, the maximum height of a vase is of type *real* provokes an automatic checking by the system and an error message in case of an attempt to input a datum inconsistent with the correspondent typing like, for example, a string of characters.
ii. Domain constraints: these constraints are associated with either the listing or the range of a property value. Here the allowed values are explicitly indicated by the user through those two typing tools.
iii. Functional constraints: It has been previously stated that properties can be single or multivalued. It is assumed for default that a property be single, like for instance the identification number of an object. However in the phase of typing, the property type is enclosed in curly brackets. If for instance we want to express that a vase can have multiple types of decoration or be undecorated we have:
vase := [..., dec: {string}, ...]

After having performed the structural specification, together with the properties typing and the description of integrity constraints, the type definition ends with its hierarchical allocation within the whole knowledge-base structuring which is based on a conceptual tool borrowed from semantic networks: namely the ISA relationship.

6.4 Hierarchical Organization

In a knowledge-base, and more precisely in the K-Schema, it is possible to organize the type definition within an ISA hierarchy. That is basically a generalization/specialization relationship between types. For instance we can declare that: 'cup ISA vase'. Intuitively this statement shows that all the characteristics of 'vase' are encountered in 'cup' as well, although the latter could have additional characteristics which are not necessarily encountered in all vases. This principle is often referred to as *principle of inheritance* because the type cup *inherits* all the characteristics of the type vase.

In the extensional level the ISA relationship turns into an inclusion relationship between classes. The example shows that the class of cups is contained in the class of vases. These qualitative considerations are rigorously described by the language, through strict criteria that guide the building of ISA hierarchies.

6.5 Type Specialization

As already stated, the ISA relationship implies that the type being defined be a specialization of the types appearing under ISA. Moreover, the principle of *inheritance* is also used to obtain a more compact schema description. Inheritance can be single or multiple, if in the ISA construction appear one or more supertypes. We talk instead of *absolute* inheritance when the properties of the supertype are inherited without being modified.

Having given a type, the creation of a subtype is performed through specialization. The mechanisms of specialization must always be respected in defining a type using the ISA construct. Those mechanisms are explained in the following sections and are of two basic sorts: specialization by specification and specialization by restriction.

i. Specialization by specification: This mechanism of specialization requires the addition of new properties to

those already defined in the supertypes (which, as stated above, are inherited by the subtype). If the supertypes are two or more and have properties in common, inconsistencies can arise. This is a critical point in multiple inheritance, the problem has already been faced in the literature and there are different ways to solve it, but their description goes beyond the scope of this paper.

ii. Specialization by restriction: The mechanism of restriction allows to refine in the subtype one or more properties already defined in the supertype (this mechanism is called *overriding)*. The overriding is performed essentially on the two property typing tools mentioned above: the explicit listing of allowed property values in the case of categorical variables and/or the range of allowed property values in the case of continuous variables.

7 Architecture of the system

The design of an application starts with the definition of the schema and proceeds by verifying its syntactic and semantic correctness. The schema contains the definition of the types of the application domain which describe the structure of objects and the relative integrity constraints. The cycle definition/verification can be iterated several times and, at each cycle, the schema is expanded progressively. Finally, a prototype for the designed application is generated. Each of these design steps corresponds to a subsystem of Mosaico: *Editor-Browser, Semantic Verifier, Code Generator, Functional Verifier, ODB Manager, and Stand Alone Prototyper*. These subsystems will now briefly be described.

1. Editor-Browser (EdiBro): This subsystem provides all the tools necessary to compose the specifications of an application domain, according to the OO paradigm (see sec. 4). As already stated a domain is described through the definition of its types. Types can be defined *ex-novo* or imported from a type library, in which case they can be refined and subtyped. New types can be inserted in the type library for future use.
2. Semantic Verifier (SemVer): By using the Semantic Verifier it is possible to check the syntactic and semantic correctness of the developed KB. The first step of the semantic verification is the parsing of the TQL++ specification. If any errors are detected at this level, the designer can go back to the *EdiBro* subsystem and change the incorrect type definitions. Otherwise the TQL++ specification is first translated into *Intercode* (an internal representation of the specification), and then semantically checked by using theorem-proving techniques. The Intercode is also used for the final, executable code generation, as described below.
3. Code Generator (CodGen): The Code Generator is devoted to the production of executable code, implementing a rapid prototype of the designed application (or part of it). This is done by mapping Intercode into a computer language (also called *object language*). The designer can generate the rapid prototype by choosing from the following languages: C++, Prolog, and O2C.
4. Functional Verifier (FunVer): This subsystem allows the user to run the prototype and monitor the execution.
5. ODB Manager (ODBman): To actually test the application it is necessary to populate the object database. This subsystem allows for the initial generation of a set of test objects (i.e. the specification of an ODB), by using the language Lobster (Missikoff/Toiati 1993). Once created, the ODB is processed to check its correctness, by matching the objects with the corresponding type definitions (declared in the schema). If no error occurs the ODB is loaded and its content can be used by the prototype during execution.
   The ODB Manager also provides a *Query Tool* to retrieve data interactively from the database. The querying is performed by using the same language conceived for the data definition: TQL++.
6. Stand Alone Prototyper (MOSAP_Gen): After the application has been semantically and functionally verified, the stand alone prototype, referred to as MOSAP (MOsaico Stand Alone Prototype), can be generated. In particular, MOSAP_Gen takes the Intercode representation of the application, generated by CodGen, as input and produces the executable prototype as output. A stand alone prototype is an autonomous executable program. It can be installed on a machine different from the one on which Mosaico runs.

In developing the specification of an application, the designer can interact with all Mosaico subsystems. The interaction is guided by the iconic interface described below. The only component implemented in the interface to date is represented by the EdiBro subsystem, through which it is possible to specify a KB using the language TQL++.

8 The iconic interface of Mosaico

The development of good user interfaces based on the iconic paradigm is a difficult task, since sound techniques, which guarantee that the interface will be easy to learn and easy to use, are still lacking. In the development of InterMos, the iconic interface of Mosaico, the methodology Iconit has been applied (Constabile/Missikoff in press). Iconit distinguishes two major design phases: (i) the design of the interface scheme and (ii) the detailed design of the windows. By interface scheme (also called dialog scheme)

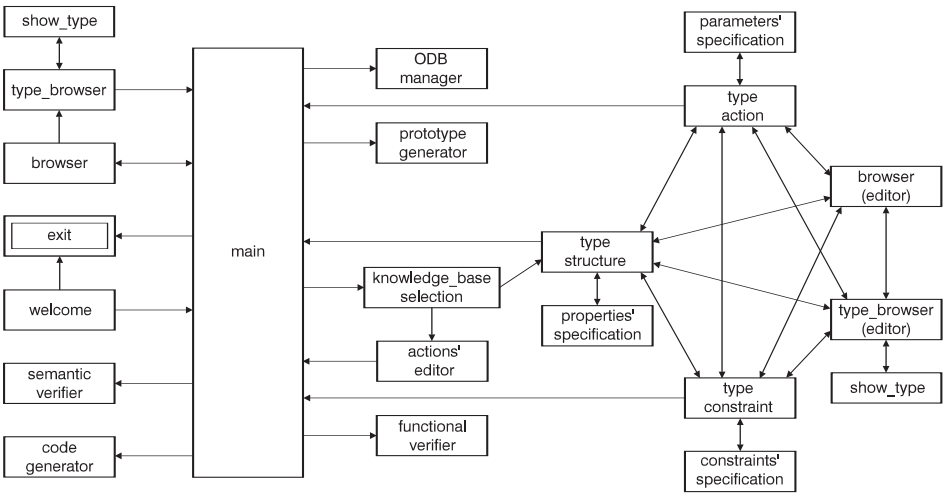

Figure 1. A partial ISTD (Interface State Transition Diagram) of Mosaico.

is meant the design of the overall interface organization, which refers to both the content and sequencing of the windows, omitting the description of each window appearance

8.1 The interface development methodology

Existing tools for interface design are mainly targeted at the construction of the windows and do not address explicitly the definition of the dialog scheme. Usually, the designer sketches the overall structure of the interface on paper, using some diagrammatic representation of the window organization and sequencing, and then creates and implements the interface windows, which are linked explicitly one to another. The consequence of this approach is that any modification in the interface organization, after the implementation, requires the recompilation of a certain number of windows. This is one of the reasons why we believe that it is useful to separate the two issues, thus creating the specific windows independently, and later organizing them in the interface as indicated by the dialog scheme. According to this approach, we have conceived a methodology which allows for an explicit separation of the two above design phases, in particular the first phase produces the dialog scheme (referred to as ISTD: Interface State Transition Diagram) and the second one the set of windows.

8.2 The ISTD of Mosaico

Once a preliminary analysis of the entities and functions required by the target system is performed, it will be possible to start the first phase of the interface development, namely the ISTD (Interface State Transition Diagram) definition. A partial ISTD of Mosaico is shown in figure 1. Note that the ISTD is fully specified only for the Browser and Editor components. In the diagram, it is possible to distinguish the root, corresponding to the initial WELCOME window (fig. 2), in which a password must be provided by the user. If the password is correct, there is a transition to the MAIN window (fig. 3), in which the user can choose the subsystem of interest; this choice will determine a transition to a specific window. If the user chooses the Editor subsystem, the interface prompts the user with the name of the KB to be edited, and then performs the transition to the TYPE STRUCTURE window (fig. 4). From this window the user has several options: 1) edit a type (thus remaining in the same window); 2) call the Browser (e.g. for loading a type defined in another schema); 3) go back to the MAIN window (see the links in the ISTD in fig. 1) and so forth.

Once the ISTD has been designed, the interface windows corresponding to the ISTD nodes are also created. Some functionalities of the developed interface are shown in the next section through a working example.

8.3 Interacting with the system

It will now be shown with an example how the EdiBro subsystem works. A prototype of Mosaico, with the interface, has already been developed on a Sun workstation and the figures included here are hard copies of the screen.

The case study is represented by the conceptual structuring of the type ‘Fibula’ (and of some of its specializations), using materials from the Villanovan cemetery of Quattro Fontanili near Veii, in Southern Etruria. A particular interest derives from the fact that the materials have been

Figure 2. The initial WELCOME window of Mosaico.

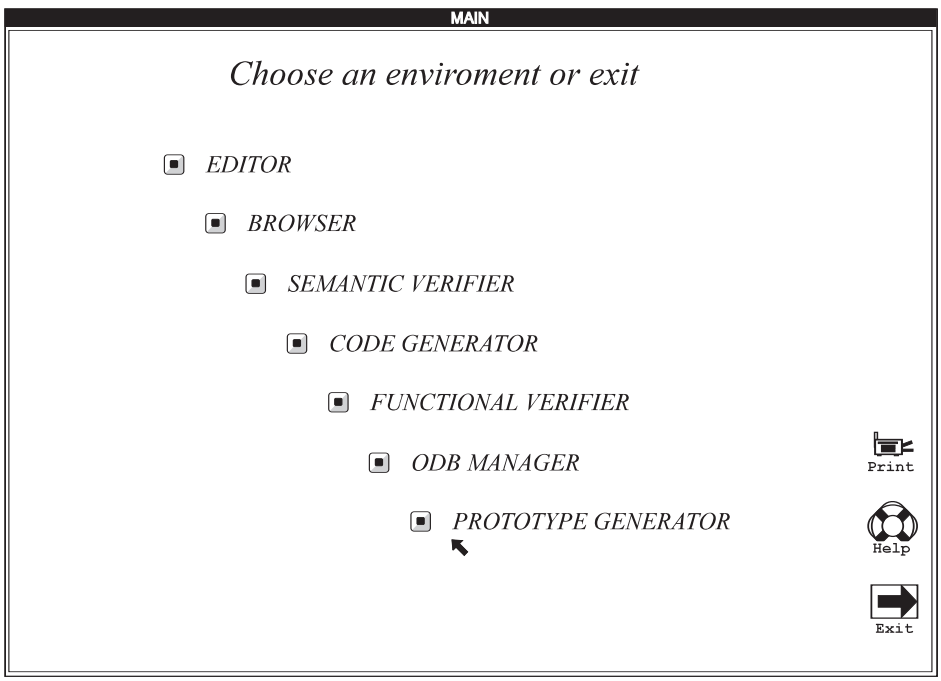

Figure 3. The MAIN window of Mosaico.

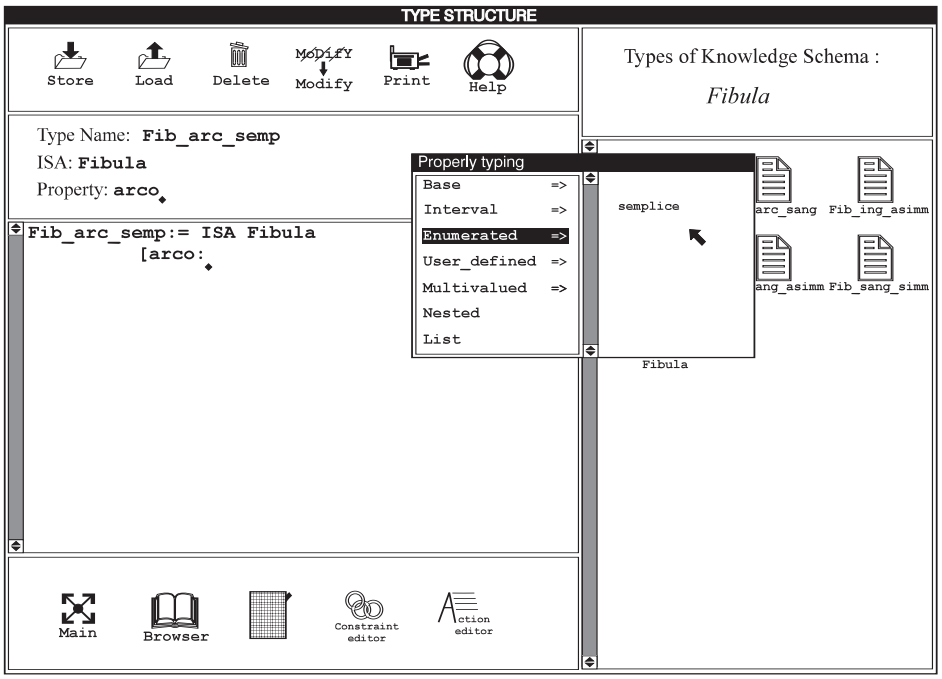

Figure 4. The TYPE STRUCTURE window of Mosaico.

previously studied by several authors (e.g. Close-Brooks 1965; Toms 1986; Kampfmeier 1986; Guidi 1993), making a comparison possible between the various approaches and results. The description of materials has been performed on the basis of *Die alteren italienischen Fibeln,* written by J. Sundwall in 1943, and the *Dizionari Terminologici*, published by the Italian Ministero dei Beni Culturali in 1980. Such a description is developed according to a hierarchy of attributes which generates, consequently, a hierarchy of types progressively more specialized.
The attributes hierarchy is the following:

1. form of the arco (bow);
2. decoration
3. form of the staffa (catch).

Supposing that the user wants to define a new type for the KB, the Editor is selected from the MAIN window and the TYPE STRUCTURE window appears (fig. 4). The structure of this window, composed of several panes, is the same for all Editor windows, so that the user will keep a consistent view. On the right side, the types already defined for the current KB are shown. In our example seven types have already been defined, namely: Fibula; Fib_arc_ing (fibula ad arco ingrossato); Fib_arc_sang (fibula ad arco a sanguisuga); Fib_ing_simm (fibula ad arco ingrossato e staffa simmetrica); Fib_ing_asimm (fibula ad arco ingrossato e staffa asimmetrica); Fib_sang_simm (fibula ad arco a sanguisuga e staffa simmetrica); Fib_sang_asimm (fibula ad arco a sanguisuga e staffa asimmetrica).

The top and bottom panes on the left side of the window contain some icons. The icons in the top pane indicate four operations: 1) *store,* for storing the current type into the KB schema; 2) *load,* for extracting an already defined type; 3) *delete,* for eliminating a type from the schema; 4) *modify*, for updating some characteristics of an existing type. The usual *help* icon is also in this pane.

In the bottom pane, the icons indicating navigational actions are included. A navigational action allows the user to move to other windows of the interface. Going from left to right, the first icon allows to go back to the MAIN window, the second one calls the Browser, the third one goes to the TYPE STRUCTURE window (in fig. 4 it is not active because the TYPE STRUCTURE window is the current one), the fourth and the fifth icons go to the other two windows of the editor, for editing constraints and actions respectively. Such icons, in the same position in all windows, are shown in reverse when not active.

Figure 4 shows the creation of the type ‘Fibula’; the first operation to perform is to enter a value for the property ‘type name’. According to the TQL++ syntax, one or more

supertypes can be specified using the ISA construct. After that it is necessary to proceed to the properties definition; in doing this, the user is helped by the interface which translates the definitions in the TQL++ syntax, thus alleviating the user from knowing all the syntax details.

10 Conclusions

In the previous pages a new classificatory paradigm has been presented that could contribute to drawing the archaeologists' attention again to an aspect of archaeological research characterized, in recent years, by a substantially static period. The reason for this is to be found, we believe, in the climate of disillusionment which took place after the loss of popularity of the quantitative paradigm, proposed by the new archaeologists to assure a good degree of formalization in the process of typology production.

The appearance of information methodologies allowing the formalization of classifications performed on a mainly qualitative base, opens up new perspectives able to offer reasonable possibilities of solving this fundamental and aging debate.

An application of the techniques described in this paper to the material from the Villanovan cemetery of Quattro Fontanili is however in progress, and the relative results will be the subject of a further and more extensive publication.

references

Aldenderfer, M.S. 1987 Assessing the impact of Quantitative Thinking on archaeological research: Historical and Evolutionary Insights. In: M.S. Aldenderfer (ed.), *Quantitative Research in Archaeology*, 9-29, Newbury Park: Sage Publications.

Binford L.R. S.R. Binford 1966 A preliminary analysis of functional variability in the Mousterian of the Levallois facies, *American Anthropologist* 68, 238-295.

Bordes, F. 1950 Principles d'une méthode d'étude des techniques de débitage et de la typologie du paléolitique ancien et moyen, *L'Anthropologie* 54, 19-34.

Brainerd, G.W. 1951 The place of chronological ordering in archaeological analysis, *American Antiquity* 16, 301-313.

Brew, J.O. 1946 The use and abuse of taxonomy, *The archaeology of Alkali Ridge, Papers of the Peabody Museum* 21, 44-66.

Chang, K.C. 1967 *Rethinking archaeology*. New York: Academic Press.

Christenson, A.L. D.W. Read 1977 Numerical Taxonomy, R-Mode factor analysis, and archaeological classification, *American Antiquity* 42, 163-179.

Clarke, D. 1968 *Analytical Archaeology*. London: Methuen.

Close-Brooks, J. 1965 Proposta per una suddivisione in fasi della necropoli veiente di Quattro Fontanili, *Notizie degli Scavi*, 53 ss.

Coad, P. E. Yourdon 1991 *Object-Oriented analysis*. Hemel Hempstead: Prentice Hall International.

Colombetti, M.	1985	*Le idee dell'Intelligenza Artificiale*. Milano: Arnoldo Mondadori Editore.
Costabile, M.F. M. Missikoff	in press	Iconit: An environment for Design and Prototyping of iconic interfaces, *Journal of Visual Languages and Computing*.
Ford, J.A.	1954	The Type Concept Revisited, *American Anthropologist* 56, 42-54.
Giarratano, J. G. Riley	1989	*Expert systems principles and programming*. Boston: PWS-KENT.
Gorodzov, V.A.	1933	The Typological Method in Archaeology, *American Anthropologist* 35, 95-103.
Guidi, A.	1993	*La necropoli veiente dei Quattro Fontanili*. Firenze: Leo S. Olshki.
Kampfmeier, U.	1986	ARCOS – A video-computer-documentation system for the use in archaeology and historic sciences, *Computer Applications in Archaeology*, 91-147.
Khoshafian, S. R. Abnous	1990	*Object-Orientation*. New York: John Wiley and Sons.
Klejn, L.	1982	*Archaeological Typology*. BAR International series 153, 3. Oxford: British Archaeological Reports.
Kluckhohn, C.	1939	The place of theory in anthropological studies, *Philosophy of Science* 6 (3), 328-344.
Krieger, A.D.	1944	The Typological Concept, *American Antiquity* 9 (3), 271-288.
Missikoff, M. M. Toiati	1993	*Object-Oriented Databases, the language TQL++, Usage Notes*. Leysin: EDBT Summer School.
Montelius, O.	1874	*La Suède prehistorique*. Stockholm: P.A. Norstedt.
	1885	Sur la chronologie de l'Age du Bronze, *Materiaux pour l'Histoire de l'Homme*.
	1899	*Der Orient und Europa*. Stockholm.
Robinson, W.S.	1951	A method for chronologically ordering archaeological deposits, *American Antiquity* 16, 293-301.
Seitzer, D.J.	1978	Problems and Principles of Classification in Archaeology, *Helinium* 18, 3-34.
Spaulding, A.C.	1953	Statistical Techniques for the Discovery of Artifact Types, *American Antiquity* 18 (4), 305-313.
Thomas, D.H.	1978	The awful truth about statistics in archaeology, *American Antiquity* 43, 231-244.
Toms, J.	1986	The relative chronology of the Villanovan Cemetery of Quattro Fontanili at Veii, *AION* 7, 41 ss.

Oleg Missikoff
Via di Vigna Filonardi 7
00197 Rome
Italy