Manuzio: An Object Language for Annotated Text Collections

Marek Maurizio and Renzo Orsini
Dipartimento di Informatica
Università "Ca' Foscari" di Venezia
Via Torino 155, Venezia Mestre, Italy
{marek,orsini}@dsi.unive.it

Abstract

Traditionally, textual collections are digitally represented as a set of files containing the text along with some kind of markup to define extra information, like metadata, annotations, etc. We propose a different approach which exploits the natural structure of a text to build specialized abstractions, called textual objects, over literary text's collections. These objects can be used to make non-hierarchically nested multi-level annotations, to create complex metadata, and to perform complex queries and analysis on the collection. Manuzio, the result of this approach, consists of a model, a language and a system to manage persistent text's collections and write complex applications over them. In this paper we introduce the main features of the Manuzio model and language, as well as a sketch of the system.

1 Introduction

The traditional way of representing textual information for automatic processing is through some kind of enrichment of the base text with other, distinguished, text carrying some information, like metadata, formatting instructions, etc., or by exposing the structure of the text by marking its components, like chapters, verses, etc. This approach, in which the distinguished text is called the *markup*, has been widely diffused also by the availability of standard markup languages, like SGML and, in the recent years, XML, which made possible the definition of standards specific for literary texts, like the TEI ones (Sperberg-McQueen et al., 1994). In this paper we propose a different vision of the textual information and we give the foundations of a system based on the representation of text through abstract structures.

The great advantages of a marked text is that it can be read and written with relative ease by a human being, as well as efficiently processed with a computer program. Moreover, when the marking of the text follows some widely accepted standard, it can be exchanged among different systems, processed by different applications, and, in general, used in a robust, interoperable way. Finally, the use of an extendible markup language, like XML, allows any kind of information to be added to the text in a string-encoded format.

These advantages are, however, balanced by several, noteworthy, shortcomings, both on the power and expressiveness of the representation and on the way in which computation can be carried over it. A first severe limitation is that marking can be applied only to contiguous segments of text that cannot overlap. Moreover, a text can be structured only in a strictly hierarchical fashion. Solutions exist to overcome some of these limitations, like the ones surveyed in (DeRose, 2004), but they tend to be cumbersome, to produce complex unreadable texts, and to notably increase the complexity of programs dealing with such texts. We could summarize these critics by saying the traditional markup approach is not *scalable*: it is a simple and elegant solution for simple text annotations, but it is not adequate to deal with very complex situations, where annotations are made on different levels of the texts, belongs to different categories of meaning, are created by different authors, and so on.

Another important disadvantage of the markup approach is, in our opinion, the fact that pro-

grams for processing marked text are not easily written, requiring the mastering of complex query languages, not specialized for the particular domain, like those typical of XML (for instance, XPath, XQuery, XSLT). In particular, they are not easily grasped by scholars and researchers in the humanities, which, on the contrary, should have the possibility of writing queries or even programs over such kind of data. This problem becomes particularly serious when one has the objective of developing complex text analysis applications, like for instance those in the field of text mining, or applications which perform sophisticated syntactic or semantic analysis.

For all the above reasons, in this paper we propose a radically departure from the traditional markup approach. Such approach, already successfully applied to represent, in a computer, knowledge and information in fields different from that of text processing, is known in the software engineering area as *object-oriented modeling*.

The objectives that we are trying to achieve through this approach are the following:

- to represent collections of texts with any kind of structure, including different overlapping structures for the same text;
- to represent any kind of annotations, even with complex information, on any part of the text, taking into account whatever text structure we are interested in;
- to provide a simple way to make queries, even sophisticated ones, on text and annotations;
- to provide tools to simplify the construction of complex, efficient, textual analysis programs;
- to lay out the theoretical and practical foundation of a general system to deal with multi-user annotated text corpora, or digital library.

Solutions which are not markup-oriented have been already presented in the literature. For instance (Coombs et al., 1987; DeRose et al., 1997) present a model where text is seen as one or more hierarchies of objects that is the foundation of more complex systems like those presented in (Carletta et al., 2003; Petersen, 2002; Deerwester et al., 1992). The approach that we propose presents a few similarities with those described in these papers, but it aims to provide a more complete solution. On one hand the Manuzio model is easily scalable, as the structure of each textual collection can be defined ad-hoc. On the other hand Manuzio provides a full programming and query language along with the model; such a language has been built to be expressive and easy to use in its specific domain of application. Finally, the Manuzio system is aimed to allow to store the data in a persistent database, to annotate it in a multi-user way, and to share results effortlessly.

The rest of the paper is organized as follows: in section 2 the foundations of the Manuzio data model are presented. In section 3 we have a look at the major features of the Manuzio language and finally, in section 4 an overview of the full system is given.

2 The Manuzio Model

We consider the textual information in a dual way: as a formatted sequence of characters, as well as a composition of logical structures called textual objects. This latter structural aspect has many similarities with other computer science's models called *object-oriented data models*, which are based on abstraction mechanisms to represent a certain reality of interest (Nierstrasz, 1989).

The Manuzio model is characterized by the notions of *textual object*, *composition* and *repetition* of textual objects, *attributes* of textual objects, textual objects and attribute *types*, *inheritance definition* and *specialization* among types, *underlying normalized text*. To make the presentation simpler to understand, we introduce these concepts through a graphical notation, while the language constructs for the complete model specification will be presented in the next section.

2.1 Textual Objects

A textual object is a computer representation of a text portion (called *underlying text*) together with its structural and behavioral aspects. In the following definitions we will use the term *full text* to refer to the full text being modeled. The precise notion of full text will be defined later in this section, but for now it can simply be considered as a sequence of Unicode characters.

Definition 2.1 A *textual object* is a software entity with an identity, a state and a behavior. The identity defines the precise portion of the text underlying the object. The state is constituted by a set of *properties* which are either component textual objects or attributes that can assume values of arbitrary complexity. The behavior is constituted by a collection of local procedures, eventually with parameters, called *methods*, which define computed properties or perform operations on the object.

For instance the Fig. 1 shows the structural aspects of a small set of textual objects. Each box represents a textual object and encloses its underlying text. If a box A is contained in another box B, then the textual object corresponding to A is a component of the object corresponding to B.

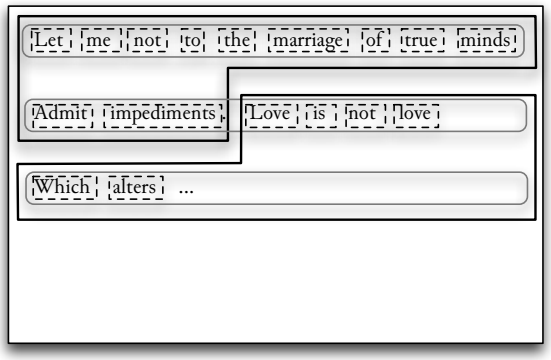

Figure 1: Example of Textual Objects

Attributes

As previously stated, the properties of a textual object are either other objects, called components, or attributes. The intended use of attributes is to complement textual objects with annotations, metadata, variants, and in general any other type of information of interest.

Definition 2.2 An *attribute* is a value of any complexity which is a property of a textual object.

An attribute has a type which can be one of the common data types present in many programming and database languages, like integers, strings, booleans, arrays, records, etc. In Fig. 2 the first line is associated with an attribute that represents its meter, while the word 'marriage' has another attribute which contains a comment about it.

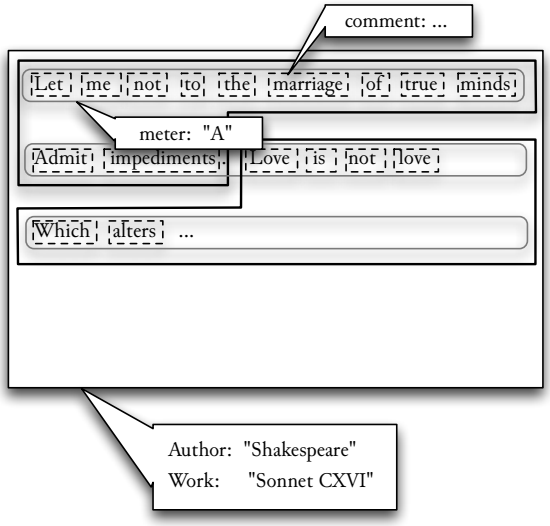

Figure 2: Example of Textual Objects with Attributes

Types

In computer science a *type* defines a set of possible values and the operations which can be applied to them. In the construction of our model, the types arise from the abstraction process of grouping different parts of text that share similar characteristics. Then the differences among the elements of those groups are ignored in order to put in evidence their similarities, i.e. their structure. For instance, the first two verses of the above example are considered (classified) as textual objects of type Line in order to stress the fact that they all have the same kind of properties (words, meter, etc.) and share the same behavior.

Every textual object is an instance of a *textual object type*. Each type has an associated interface that defines the ways that type's instances can be used to access their properties and methods.

Definition 2.3 A *textual object type interface* specifies the the names and types of the properties and the names and the parameter and result types of the methods.

In Fig. 3 we show the types corresponding to the different kind of boxes.

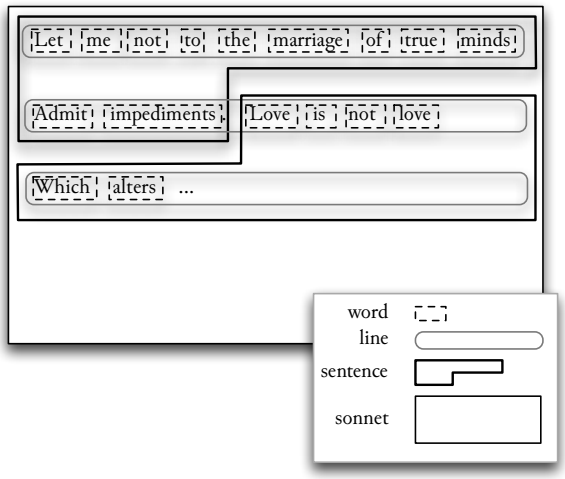

Figure 3: Example of Textual Object Types

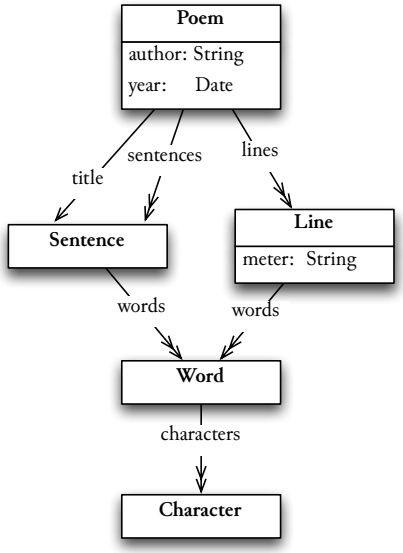

Figure 4: A simple model about poems

Graphical Representation

As previously mentioned, we will use a graphical notation to put in evidence the structure of a textual model in terms of its textual objects types, their attributes and the component relations among them. In our notation an object type and its attributes are represented by a rectangle split in two parts. The upper part contains the name of the type, while the lower one, if present, contains the name and the data type of its attributes.

To put in evidence the fact that if a textual object is component of another then there is a *component relation* between their types, this relation is graphically represented through an arrow which connects the two types, labelled with the component name. And, since this relation can be one to one (for instance each poem has only a title which is a sentence) or one to many (for instance each poem has many lines) we distinguish this fact with a different graphical notation. The former case is represented by a single-pointed arrow, the latter by a double-pointed arrow. For instance, in Fig. 4, we represent a very simple model about poems which arises from the previous examples.

In Manuzio, the textual object type which has no component is called Unit. A Unit type is always present, must be unique, and defines the minimal part of text which can be manipulated. For instance, in Fig. 4, the Unit type is Character. Different textual models can have different Unit types, depending on the granularity of the textual analysis in which the user is interested to. For instance, one could be interested in lemmas, or in syllables, instead of characters for different kind of analysis, or in written representation of phonemes to develop phonetic analysis programs, etc.

2.2 Type Inheritance

Another important information that can be modeled in Manuzio is that textual objects types are not always independent, but can exist a particular relation among them, called *specialization*, through which we can model objects at different levels of detail. If a type A is defined as specialization of type B, then the instances of A inherits all the characteristics of the instances of B, in addition to having other, proper ones. For example, an hendecasyllable has all the characteristics of a line (it *is* in effect a line), but it has also the property of having exactly eleven syllables.

The presence of the specialization between two textual object types A (which will be called the *subtype*) and B (the *supertype*) has the effect that every instance of the subtype (for example every hendecasyllable), can be treated both as a generic line (for instance when performing a textual search), and as a line with a specific number of syllables (for instance to define specific methods which are significant only for hendecasyllables).

Definition 2.4 A type A is *subtype* of a type B if it is defined as such; in this case A inherits all the properties and the behavior of B. A can also have new properties and methods, and can redefine the type of its components with a more specialized type.

Graphical Representation

A subtype is graphically connected to its supertype through an arrow with a hollow arrowhead, and shows only the new information (with respect to its supertype). For this reason the lower part of the rectangle contains only the new attributes, while only the arrows representing the new components are drawn. In Fig. 5 an example about simple works is shown.

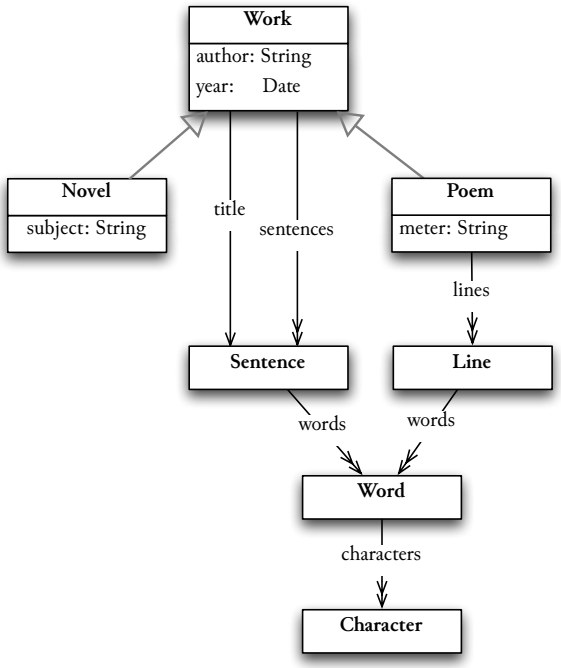

Figure 5: A model about poems and novels.

In the figure, both Novel and Poem are subtypes of Work so that they inherit the components title and sentences, as well as the attributes year and author. Moreover, the Novel type has the new attribute subject, while the Poem type has the new attribute meter and a component lines which allows to model lines of a poem.

2.3 Underlying Text

Each textual object has a direct correspondence to a part of the text to be represented in our model (for instance a single work, a corpora, a library, etc.). We assume that such texts exist, as sequences of Unicode characters, in a format which is chosen by some expert. The following definitions specify the exact relation between texts and objects of the model.

Definition 2.5 The *full text* is a sequence of Unicode characters that represents all the text described by a specific Manuzio model.

Each textual object has an underlying text, defined as:

Definition 2.6 The *underlying text* of a textual object is a subsequence of the characters of the model's *full text* associated to it.

As previously mentioned, the identity of a textual object is determined by the underlying text: two objects are identical if they have the same underlying text. Moreover, this concept is fundamental also in defining an important semantics property of the Manuzio model:

Definition 2.7 A textual model is *well-formed* if each component of that textual object has an underlying text which is a proper subsequence of its underlying text.

Finally, the underlying text will be useful in the system to provide the user a concrete representation of a textual object.

While the Manuzio model has been presented so far through a graphical notation, a formal language is necessary in order to implement a computer system for representing textual models according to our approach and to operate on them. In the following section the main features of such a language are introduced.

3 The Manuzio Language

The Manuzio language is a full programming language with constructs to define textual models and write complex query expressions or sophisticated textual analysis applications. Such a language is intended to be used by a multi-user system to store

persistently a digital collection of texts over which these programs are evaluated.

The full syntax and semantics of the language will be available in the forthcoming manual and are beyond the scope of this article. In particular, the language features aimed to describe the textual object types of a model will be only shown through the following example (Fig. 6).

```
Schema PlaySchema

        type Play
          has
            title : Sentence
            acts : Acts
          attribute
            author : String
            year   : Integer
        end

        type Act
          has
            scenes: Scenes
          attribute
            directions : String
        end

        type Scene
          has
            speeches : Speeches
        end

        type Prologue is Scene end

        type Epilogue is Scene
          has
            salutations : Speech
        end

        type Speech
          has
            sentences: Sentences
            lines : Lines
          attribute
            speaker : String
        end

        type Line
          has words : Words
        end

        type Sentence
          has words : Words
        end

        type Word
          has characters : Characters
        end

End
```

Figure 6: Schema definition example.

The model described in figure 6 concerns simple plays, where a Play[1] is composed by a title and some acts, and has attributes author and publication year. The Speech type has two components, lines and sentences, which are independent ways of considering a speech. The types Prologue and Epilogue inherits their properties from the supertype Scene. While Prologue does not have any additional property, the Epilogue type adds a salutations component. The type Word has components of the basic predefined type Character, which represents a unicode character.

3.1 Repeated Textual Objects

In Fig. 6, types like Words or Lines, which are not explicitly defined, are used. In fact, when we define a textual object type, there is an implicit definition of its "plural form", which is a type whose instances, called *repeated textual objects*, contains repetitions of objects of the "singular form" type. For example, an object of type Words is composed by a repetition of instances of type Word.

The existence of these types in the language has the following consequences:

- A repeated textual object is in effect a textual object, can have properties or methods, and in general can be treated as any other textual object.

- Specific operators exist which take into account the multiplicity of the elements of a repeated textual object. For instance, we can count how many words are present in an object of type Words, we can select the first word, and so on.

- The query language operators on single textual objects can be applied also to their repetitions, with the meaning that the operator is applied to all the repetition's elements and returns the collection of the results. For instance, the operation that returns the title of a single poem, when applied to a collection of poems, returns all their titles.

We are ready to present now the operational features of the language, but the reader must keep in mind that, while Manuzio is a full programming

[1] By convention, a type name is capitalized.

language, like java or prolog, which allows experienced users to write programs of arbitrary complexity, in the rest of this section we will discuss only its query-like operators. These operators allow non-programmer to work with the system in an easy way to fetch, refine, display and annotate query results, in a manner similar to other data query languages, like, for instance, the relational databases language SQL.

3.2 Basic Textual Objects Access

A uniform notation is used to select both properties and methods of a textual object. Such a selection is performed through the access operator 'of'. For instance, if P is a poem then:

```
title of P
```

returns the textual object of type Sentence which is the component title of the poem P, while

```
author of P
```

returns instead the value of the attribute author. When the result of the 'of' operator is a textual object, the operation can be repeated:

```
words of title of P
```

returns the textual object of type Words which contains the words of the title of P. As previously mentioned, when a component access is applied to a repeated textual object, the result is again a repeated textual object, as in:

```
words of lines of P
```

which returns a textual object containing *all* the words of *all* the lines of P.

As stated in the definition 2.4 in the Section 2, for the inheritance property of the subtyping mechanism, the 'of' operator for a certain type can be applied also the the instances of its subtypes. For instance, since Epilogue is a subtype of Scene, we can select the speeches of a epilogue E (since it is also a scene), in the same manner as its salutations:

```
speeches of E
```

Analogously, if there were methods defined on Scene, they could be applied also to instances of Epilogue.

3.3 Repeated Objects Operators

There exist operators specific to repeated textual objects which take into account the elements of those objects by performing some operations on all, or a subset, of them.

A first group of operators can be used to get a part of a repetition by specifying the elements in which we are interested either by using ordinal adjectives or numeric ranges.

```
first line of P
words(1..3) of second line of P
last sentence of P
```

It is also possible to count the elements of a repetition, like in:

```
number of lines of P
```

as well as use other operations on sequences, like concatenation of sequences, test for a condition holding on some or all the elements, test for inclusion of an element, etc., which are not described here since their are typical for data structures like sequences or arrays present in other languages.

The most important operator of this category, which is the foundation of the query-like part of the Manuzio language, is the 'select' operator. As suggested by the name, it has a syntax similar to SQL selection, and can be used to retrieve objects through conditional expressions. Here, it will be described through examples, in which we assume C a collection of poems.

The first example shows the simple form "select E1 from Id in E2", where E2 is an expression returning a repeated textual object whose elements are bound, in order, to the identifier Id, used in the evaluation of the expression E1. The result is the collection of such values.

```
select title of p
from p in poems of C
```

The first example returns a repeated textual object composed by the sentences formed by all titles of the poems of the collection C. The type of the resulting object is Sentences, and, in this simple form, it is equivalent to the expression:

```
title of poems of C
```

A select expression can have a 'where' clause, which can be used to give a condition to filter the elements over which the construct iterates. For instance, the following example returns only the titles of Shakespeare's poems.

```
select title of p
from p in poems of C
where author of p = "Shakespeare"
```

The last example shows how to use this construct to build complex queries. For instance, to find all the lines of Shakespeare's poems with exactly five words, we could write:

```
select l
from l in ( select lines of p
            from p in poems of C
            where author of p = "Shakespeare" )
where number of words of l = 5
```

The internal 'select' returns a repeated textual object, over which the external one iterates.

3.4 Access to the underlying text

Given a textual object, we can select its underlying text with 'text'. For instance, given a word W, the following expression:

```
text of W
```

returns a string that contains the unicode characters of the word W.

It is also possible to access the starting position of the underlying text of an object:

```
text_position of W
```

which returns a number which specify the offset position of the underlying text of W from the start of the full text.

Note that, since the language has a complete set of operators on regular data types, once we get the text of a textual object we can apply to it all the operators available on character strings (which includes, among others, pattern matching through regular expressions).

3.5 Comparisons and test operators

In the language, the usual comparison operators on strings and other simple values (=, >, <, etc.) are available.

On the other hand, two textual objects can be tested for identity (i.e. if they are in effect the same object) with the identity operator '==':

```
o1 == o2
```

When applied to textual objects, the string equality operator is an abbreviation for testing the equality of their underlying text, like in:

```
o1 = o2
```

which is equivalent to:

```
text of o1 = text of o2
```

Also the other comparison operators, when applied to textual objects, takes into account their underlying text. For instance:

```
o1 < o2
```

is an abbreviation for:

```
text_position of o1 < text_position of o2
```

that returns true when the object o1 precedes o2 in the full text (analogously for <, >=, <=).

In addition, the operator '><' returns true when an object overlaps another, so that, for instance, we can test if a sentence and a line overlap.

Finally, the language has a few other operators to test about the relative positions of two textual objects, known as the Allen relations (Allen, 1983). They allow, for instance, to know if an object is fully contained in another one, if one partially precedes another, and so on.

4 An overview of the Manuzio System

The purpose of Manuzio is to be a system, based on the presented language, with the capabilities of storing in a persistent way complex annotated text collections and allowing their manipulation by different users in a coherent and cooperative way. To reach such a goal the Manuzio language has other features, in addition to those shown in section 3, to deal with users permissions, dynamic annotations, and management of the model's persistency.

While the full system architecture is the subject of a forthcoming paper, here we will present an overview of its capabilities.

1. The system provides an efficient way of storing, in a persistent way, and querying very large quantities of textual material, together with annotations. To achieve this objective we are investigating different solutions including those based on relational database technology.

2. The system has as its main programming and administration language the Manuzio language. For this reason, the language has also constructs to extend the model's schema with new types, to extend a type with new attributes and methods, to make persistent textual objects retrieved by a query, and to add and modify annotations on them.

3. The system has tools that allow the access to concurrent users, through an appropriate set of permissions. For instance, different groups of users can work with different sets of annotations, that can later be compared and eventually merged.

4. Users can interact with the system either through the Manuzio language or through a friendly graphical interface to perform assisted queries whose results are visualized with a choice of different graphical formats and mediums.

5. To exchange texts and annotations with other systems the XML standard format can be used through a set of tools which facilitates the mapping between it and the Manuzio internal format. In particular, XML is the privileged way of loading the data into the textual database, an operation which is done by a parsing process that can be automatic, semi-automatic or manual, depending on the complexity of the source data.

The system is currently under development, but the Manuzio model capabilities and a subset of the language's features have been tested using a simple prototype built with the Ruby programming language. The prototype has a fixed scheme of medium complexity concerning epic latin poems, and has been used to successfully performs clause-related analysis on a medium-sized corpora.

5 Conclusions and future work

This paper is an introduction to a novel approach for dealing with annotated collection of texts. We have presented the Manuzio model, which has some similarities with other object-oriented models, although it is specialized for the specific domain. Our proposal includes also a full programming language, the Manuzio language, of which only the query features have been discussed. Finally, a sketch of a complete solution, consisting of a system based on that language, has been presented. While Manuzio is still a work in progress, our first experiments have shown the feasibility of the approach in dealing with collection of literary texts.

The next step in Manuzio development will be the complete language specification and implementation, along with a full functional architecture of the system.

Acknowledgment

This work has been supported in part by grants of the italian Ministero dell'Istruzione, Università e Ricerca Scientifica, under the PRIN Project "Musisque deoque II. Un archivio digitale dinamico di poesia latina, dalle origini al Rinascimento italiano" about a digital archive of latin poetry. We are indebted to Paolo Mastandrea and Luigi Tessarolo, who have stimulated our interest in this subject and provided useful suggestions and challenging examples through discussions about their previous work.

References

Sperberg-McQueen, C., Burnard, L., Bauman, S.: Guidelines for electronic text encoding and interchange. (1994)

DeRose, S.J.: Markup overlap: A review and a horse. In: Extreme Markup Languages. (2004)

Coombs, J.H., Renear, A.H., DeRose, S.J.: Markup systems and the future of scholary text processing. Commun. ACM **30**(11) (1987) 933–947

DeRose, S., Durand, D., Mylonas, E., Renear, A.: What is text, really? ACM SIGDOC Asterisk Journal of Computer Documentation **21**(3) (1997) 1–24

Carletta, J., Evert, S., Heid, U., Kilgour, J., Robertson, J., Voormann, H.: The NITE XML Toolkit: flexible annotation for multi-modal language data. Behavior Research Methods, Instruments, and Computers **35**(3) (2003) Special issue on Measuring Behavior.

Petersen, U.: The standard MDF model. Unpublished article. Obtainable from URL: http://emdros.org (2002)

Deerwester, S.C., Waclena, K., LaMar, M.: A textual object management system. In Belkin, N.J., Ingwersen, P., Pejtersen, A.M., eds.: SIGIR, ACM (1992) 126–139

Mastandrea, P.: De fine versus, repertorio di clausole ricorrenti nella poesia dattilica latina

Allen, J.F.: Maintaining knowledge about temporal intervals Communications of the ACM **26**(11) (1983) 832–843

Nierstrasz, O.: A Survey of Object-Oriented Concepts Object-Oriented Concepts, Databases and Applications (1989) 3–22