Manuzio: An Object Language for Annotated Text Collections

[Extended Abstract]

1. INTRODUCTION

More and more large repositories of texts which must be automatically processed represent their content through the use of descriptive markup languages. This method has been diffused by the availability of widely adopted standards like SGML and, later, XML, which made possible the definition of specific formats for many kinds of text, from literary texts (TEI) to web pages (XHTML). The markup approach has, however, several noteworthy shortcomings. First, we can encode easily only texts with a hierarchical structure, then extra-textual information, like metadata, can be tied only to the same structure of the text and must be expressed as strings of the markup language. Third, queries and programs for the retrieval and processing of text must be expressed in terms of languages like XQuery [4]. In the XQuery data model, every document is represented as a tree of nodes; for this reason, in documents where parallel, overlapping structures exists, the complexity of XQuery programs becomes significantly higher.

Consider, for instance, a collection of classical lyrics, with two parallel hierarchies lyric > stanzas > verses > words, and lyric > sentences > words, with title and information about the author for each lyric, and where the text is annotated both with commentary made by different scholars, and with grammatical categories in form of tree-structured data. Such a collection, if represented with markup techniques, would be very complex to create, manage and use, even with sophisticated tools, requiring the development of complex ad-hoc software.

To overcome the above limitations due to the use of markup languages partial solutions exist (see for instance [3]), but at the expense of greatly increasing the complexity of the representation. Moreover, markup query languages need to be extended to take these solutions into consideration [1], and program can be tied only to the same structure of the text and must be expressed as strings of the markup language. Third, queries and programs for the retrieval and processing of text must be expressed in terms of languages like XQuery [4]. In the XQuery data model, every document is represented as a tree of nodes; for this reason, in documents where parallel, overlapping structures exists, the complexity of XQuery programs becomes significantly higher.

Consider, for instance, a collection of classical lyrics, with two parallel hierarchies lyric > stanzas > verses > words, and lyric > sentences > words, with title and information about the author for each lyric, and where the text is annotated both with commentary made by different scholars, and with grammatical categories in form of tree-structured data. Such a collection, if represented with markup techniques, would be very complex to create, manage and use, even with sophisticated tools, requiring the development of complex ad-hoc software.

To overcome the above limitations due to the use of markup languages partial solutions exist (see for instance [3]), but at the expense of greatly increasing the complexity of the representation. Moreover, markup query languages need to be extended to take these solutions into consideration [1], and program can be tied only to the same structure of the text and must be expressed as strings of the markup language. Third, queries and programs for the retrieval and processing of text must be expressed in terms of languages like XQuery [4]. In the XQuery data model, every document is represented as a tree of nodes; for this reason, in documents where parallel, overlapping structures exists, the complexity of XQuery programs becomes significantly higher.

Consider, for instance, a collection of classical lyrics, with two parallel hierarchies lyric > stanzas > verses > words, and lyric > sentences > words, with title and information about the author for each lyric, and where the text is annotated both with commentary made by different scholars, and with grammatical categories in form of tree-structured data. Such a collection, if represented with markup techniques, would be very complex to create, manage and use, even with sophisticated tools, requiring the development of complex ad-hoc software.

To overcome the above limitations due to the use of markup languages partial solutions exist (see for instance [3]), but at the expense of greatly increasing the complexity of the representation. Moreover, markup query languages need to be extended to take these solutions into consideration [1], and program can be tied only to the same structure of the text and must be expressed as strings of the markup language. Third, queries and programs for the retrieval and processing of text must be expressed in terms of languages like XQuery [4]. In the XQuery data model, every document is represented as a tree of nodes; for this reason, in documents where parallel, overlapping structures exists, the complexity of XQuery programs becomes significantly higher.

Consider, for instance, a collection of classical lyrics, with two parallel hierarchies lyric > stanzas > verses > words, and lyric > sentences > words, with title and information about the author for each lyric, and where the text is annotated both with commentary made by different scholars, and with grammatical categories in form of tree-structured data. Such a collection, if represented with markup techniques, would be very complex to create, manage and use, even with sophisticated tools, requiring the development of complex ad-hoc software.

To overcome the above limitations due to the use of markup languages partial solutions exist (see for instance [3]), but at the expense of greatly increasing the complexity of the representation. Moreover, markup query languages need to be extended to take these solutions into consideration [1], and program can be tied only to the same structure of the text and must be expressed as strings of the markup language. Third, queries and programs for the retrieval and processing of text must be expressed in terms of languages like XQuery [4]. In the XQuery data model, every document is represented as a tree of nodes; for this reason, in documents where parallel, overlapping structures exists, the complexity of XQuery programs becomes significantly higher.

Consider, for instance, a collection of classical lyrics, with two parallel hierarchies lyric > stanzas > verses > words, and lyric > sentences > words, with title and information about the author for each lyric, and where the text is annotated both with commentary made by different scholars, and with grammatical categories in form of tree-structured data. Such a collection, if represented with markup techniques, would be very complex to create, manage and use, even with sophisticated tools, requiring the development of complex ad-hoc software.
the whole collection, and the least is the type of the most basic objects of the schema. Types can also be defined by inheritance, like in object-oriented languages. For instance, the types Novel and Poem are both subtypes of Work. An example of textual schema is given, by the means of a graphical notation, in Figure 1.

![Figure 1: Example of Manuzio Model.](image)

3. THE MANUZIO LANGUAGE

Manuzio is a functional, type-safe programming language with specific constructs to interact with persistently stored textual objects. The language has a type system with which to describe schemas as that illustrated in Figure 1, and a set of operators which can retrieve textual objects without using any external query language. A persistent collection of documents can be imported in a program and its root element can be referenced by a special variable collection of type Collection. From this value all the textual objects present in the collection can be retrieved through operators that exploit their type’s structure: the get operator retrieve a specific component of an object, while the all operator retrieve recursively all the components and subcomponents of a certain type of an object. Other operators allow the creation of expressions similar to SQL or XQuery FLOWR expressions\(^2\). Since the queries are an integrated part of the language, they are subject to type-checking and can be used in conjunction with all the other language’s features transparently.

The program in Source Code 1, for instance, assigns to a variable the first three sentences of each work. This portion of text can be subsequently refined or used in any retrieval context. In Source Code 2 a more complex example is shown, where an analysis of Shakespeare’s plays extracts the top three “love speaking” characters in “A Midsummer Night’s Dream”. The results of such code are then reported in Source Code 3.

let most_relevant_sentences = select all SENTENCE 1...3 of works of c;

Source Code 1: Retrieve the most relevant sentences of each work.

\(^2\)The full syntax and semantics of the Manuzio language can be found in [6].

let loveSpeeches =
    s in (get all Speech of play)
    where some w in (get all Word of s)
    with (get stem of w) = "love";

Source Code 2: Compute a new structure of the most love-speaking characters.

The top 3 love speakers are:

\[ \{ \text{speaker} = \text{LYSANDER}, n=17 \}, \{ \text{speaker} = \text{HERMIA}, n=13 \}, \{ \text{speaker} = \text{OBERON}, n=12 \} \]

Source Code 3: Results of Source Code 2.

4. CONCLUSIONS

To evaluate the usefulness of our approach a first prototype of the Manuzio language has been developed by mapping the textual objects onto a relational database system. We are aware that a great deal of work on data representation and query optimization must yet be done to provide a satisfying performance for large collections of texts. However, we think that work on modeling and linguistics aspects of retrieval of texts and computations over them is very important, and prerequisite to enrich the solutions offered by research areas such as information retrieval and digital libraries. In particular, we believe that our language allows the user to take into account structural information in a simple way in queries, and this could improve the quality of their results (a term is certainly more significant when used in a title instead that in a footnote).

5. REFERENCES


