Time Automaton
A visual mechanism for temporal querying

Luís Certo
certo@fe.up.pt

Teresa G. Dias
tgalvao@fe.up.pt

José Borges
jlborges@fe.up.pt

Department of Industrial Engineering and Management
School of Engineering of the University of Porto, Rua Dr. Roberto Frias, s/n 4200-465, Porto

Abstract

Visual querying mechanisms are normally comfortable and accessible mechanisms. However, they generally do not provide the means for formulating a large set of different types of temporal interrogations. In this paper we propose the Time Automaton, a highly flexible visual language, capable of formulating many types of temporal queries. Time Automaton's logic is innovative and unlike available visual mechanisms it is not based on the direct conversion of visual queries into a predefined set of commands of a textual querying language, like for example, SQL.

Keywords

temporal query, visual mechanism, visual model, regular expression

1. INTRODUCTION

Using visual querying mechanisms is normally more comfortable than using traditional text-based querying languages. Most important, this type of mechanisms is useful when users do not possess programming skills. However, querying temporal data is a complex process, and, to the best of our knowledge, available visual querying mechanisms [Jin09] [Shneiderman92] [Edsall97] do not provide the means for formulating a significant set of different types of temporal interrogations.

Available visual mechanisms convert visual configurations directly into commands of traditional text-based languages [Snodgrass84] [Snodgrass95]. Since the operators of textual querying languages were not designed to be visually manipulated their adaptation to the logic of visual mechanisms is not natural and it generates functional limitations.

In this paper we propose the Time Automaton, a highly flexible visual mechanism that is capable of formulating a vast set of different temporal queries. Time Automaton's logic is quite different from the logic of available visual mechanisms, and unlike them, Time Automaton does not use any available text-based querying language to execute its queries. Time Automaton principles are simple and the algorithm that executes the Time Automaton queries is simple to implement. A specific data structure is required to work with the mechanism. Such structure is uncomplicated and an easily implementable algorithm can automate its creation.

2. DATA STRUCTURE

Time Automaton requires temporal data to be encoded as a string. We call this data structure the **temporal string**. Two types of words compose the temporal string, **anchors** and **facts**. Anchors define the beginning of temporal moments and facts encode temporal data records, that is, the data itself. The temporal string uses a *prefix* notation where each anchor affects the anchors and facts that are subsequently positioned.

For the artificial dataset represented in Table 1 the corresponding temporal string, with a month granularity and considering the year season for each month, can be defined as follows:

```
year,2009 month,Jan,Winter facts,a,b
month,Feb,Winter facts month,Mar,Spring
facts,c,d,e,f month,Apr,Spring facts
month,May,Spring facts month,Jun,Summer
facts month,Aug,Summer facts,g,h
month,Sep,Summer facts month,Oct,Autumn,
facts month,Nov,Autumn facts,k,l,m,n
month,Dec,Autumn facts
year,2010 month,Jan,Winter facts,p,q
month,Feb,Winter facts month,Mar,Spring
facts,s
```

DATE	DATA_RECORDS
Jan, 2009	a,b
Mar, 2009	c,d,e,f
Aug, 2009	g,h
Nov, 2009	k,l,m,n
Jan, 2010	p,q
Mar, 2010	s

Table 1: Artificial dataset containing temporal information

3. MECHANISM DESCRIPTION

A Time Automaton query is represented as an oriented graphs with the following properties: **1 -** One of the nodes is the root, which has no incident edges. **2 -** Every other node is defined by an expression, which corresponds to a word in the temporal string. **3.** Edges can have a value to define the maximum number of traversals.

Each node's expression is in fact a regular expression [Aho80]. A regular expression is written in a formal language that is interpreted by a regular expression engine, which analyzes a text and finds which parts match the provided specification. For example, a regular expression

engine can analyze a text in attempt to find a word with a specific syntax, and starting the search at a specified index.

In order to abstract the process of writing the regular expressions that define nodes, some predicates were created: **1. Fact()** corresponds to *'facts((?:,\w+)+)?'*. It defines the syntax of a fact and specifies that it must be captured [Friedl06]. That is, facts are stored in a **capture buffer**, where they can be accessed separately from other matched words. **2. Undef()** is the same as '\w+'. It is used to match any word, indifferently. In the temporal querying context this predicate is useful for creating queries that do not specify characteristics of temporal moments. For example, in the query *"all facts occurred in every month of Summer"* the months' names are not specified. **3. Neg(*charact*)** corresponds to *'(?!\bcharact\b)\b\w+)'*. In the temporal querying context this predicate is useful for creating queries that exclude certain temporal moments like *"all facts occurred in every month except March"*.

The algorithm that executes Time Automaton queries corresponds to a simple graph traversal algorithm. Every time a node is visited its regular expression is supplied to a regular expression engine that finds the next matching word in the temporal string. All matched words that are stored in the **capture buffer** correspond to the facts that answer the temporal query. As an example, for the temporal string in Section 2 and the question *"which facts occurred in March 2009?"*, the corresponding Time Automaton query can be formulated as represented in Figure 1.

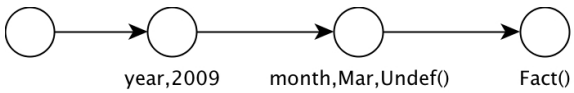

Figure 1 - Example of a Time Automaton query

Time Automaton allows nodes to have multiple outgoing edges. In such case, only one edge is traversed, and the selection criteria is to traverse the edge linking to the node that corresponds to nearest word. In terms of performance, searching for every word for each possible next node is not a good approach. A better approach is to construct a regular expression joining all regular expressions of all possible next nodes, i.e., if two regular expressions *r1* and *r2* are joined they form a regular expression '*r1* | *r2*', where '|' is the boolean operator OR. Then, the regular expression engine searches the next word that matches any of the elementary regular expressions. After finding this word the algorithm knows which node should be visited next. As an example of a multiple outgoing edge situation, for the temporal string in Section 2 and for the question "*which facts occurred in 2009 in all months except March*", the corresponding Time Automaton query is represented as depicted in Figure 2.

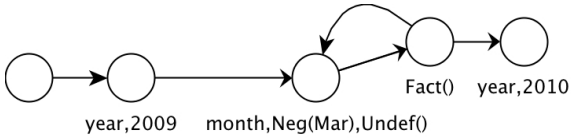

Figure 2 - Example of a Time Automaton query including nodes with multiple outgoing edges

Since Time Automaton visual layout is based on graphs it is simple to create a graphical interface that implements the mechanism. Moreover, the predicates created to encapsulate regular expressions can be easily mapped into interface controls, enabling regular expressions to be graphically built. As an example, Figure 3 shows a screenshot of an interface we've created to work with the Time Automaton.

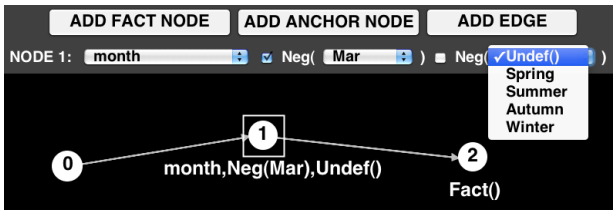

Figure 3 - Interface that implements the Time Automaton

4. CONCLUSIONS

In this paper we present the Time Automaton, a highly flexible visual mechanism that is capable of formulating many types of temporal queries. As a matter of fact, Time Automaton enables the formulation of a type of queries, which are ordinal queries (e.g *"all facts occurred in every second Saturday of every month"),* which in SQL [Codd83] are not directly formulable.

A Time Automaton query is represented as an oriented graph in which nodes are defined with regular expressions. In order to abstract the process of writing such expressions some predicates were devised. This abstraction layer and the fact Time Automaton's layout is based on graphs makes the mechanism simple to implement in a graphical interface.

Time Automaton's logic is innovative and, unlike available visual mechanisms, Time Automaton does use available text-based querying language to execute its queries. The algorithm that executes the queries is a graph traversal that follows some simple rules, thus, it is straightforward to implement. Since Time Automaton queries are graphs it is simple to store and reutilize them.

5. REFERENCES

[Aho80] A. Aho. Pattern matching in strings. Formal Language Theory: Perspectives and Open Problems, pp. 325-347, 1980.

[Codd83] E. F. Codd. A relational model of data for large shared data banks. Comun. ACM, 26(1), pp. 64-69, 1983.

[Friedl06] J. Friedl. Mastering regular expressions. O'Reilly Media, Inc., 2006.

[Jin09] J. Jin and P. Szekely. Querymarvel: A visual query language for temporal patterns using comic strips. Visual Languages and Human-Centric Computing, pp. 207-214, 2009.

[Edsall97] R. Edsall, D. Peuquet, A graphical user interface for the integration of time into GIS, Proceedings of the 1997 American Congress of Surveying and Mapping Annual Convention and Exhibition, Seattle, WA, 1997, pp. 182-189.

[Shneiderman92] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. Conference on Human Factors in Computing Systems, Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 619 - 626 , 1992.

[Snodgrass84] R. Snodgrass. The temporal query language tquel. In PODS '84: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, pp. 204–213, New York, NY, USA, 1984. ACM.

[Snodgrass95] R. T. Snodgrass, editor. The TSQL2 Temporal Query Language. Kluwer, 1995.