Time Automaton: a visual mechanism for temporal querying

Luís Certo[a,*], Teresa Galvão[a], José Borges[a]

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

Abstract

Available visual temporal querying tools do not provide the means for formulating complex temporal queries. For these queries users have to adopt text-based querying languages, such as SQL. The problem, however, is that using text-based languages is less comfortable than using visual tools and, most important, in some cases temporal queries can be extremely difficult to formulate for users that do not possess programming competences. In this paper we propose the Time Automaton, a highly flexible visual mechanism that enables the formulation of a large set of different types of temporal queries, ranging from the simple to the most complex ones. In order to test the Time Automaton we used a representative sample of temporal queries extracted from the matured OWL Time Ontology. Also, to test the Time Automaton in a real word scenario we created a visual interface that implements the mechanism and we used it to analyze a real dataset.

Keywords: visual mechanism, temporal querying, finite state automata

1. Introduction

Several research topics, like urban mobility analysis or social network mining are related with temporal data analysis and consequently with temporal querying. For example, in urban mobility analysis, transport planners usually query a set of relevant temporal moments and use GIS tools to visualize people's mobility in those moments [1]. Also, in social network mining

*Principal corresponding author
Email address: certo@fe.up.pt (Luís Certo)

researchers frequently use temporal querying tools to select representative temporal moments that are used as input in trend detection algorithms [2]. Since temporal querying plays a major role in temporal data analysis, the effectiveness of querying tools is crucial. To be effective, a querying tool must empower the user to formulate a significant set of different types of temporal queries, and should do it through a comfortable and accessible environment. In general, temporal querying can be conducted by using either text-based languages or visual tools. Text-based querying languages, such as SQL [3], are normally very flexible and they are the only option when visual tools are unable to express more complex queries. Such flexibility, however, comes with a cost, as writing queries with text-based languages is a demanding and time-consuming process, even for experienced users. Most important, text-based languages usually involve a steep learning curve for users that do not possess programming skills, who may be experts on the field under analysis. In contrast to text-based querying languages, visual querying tools [4] [5] [6] are normally easier to use although they are limited in terms of querying complexity.

The limitations in visual temporal querying tools are related to the difficulty in creating a visual logic that is both user-friendly and flexible enough to cover a large spectrum of temporal query types. Also, current visual solutions translate their visual queries directly into commands of traditional text-based querying languages which were not designed to be visually handled.

In this paper, we propose the Time Automaton, a visual temporal querying mechanism that is capable of formulating a large set of different types of temporal queries, including the most complex ones.

Time Automaton does not use any text-based querying language to execute its queries. Instead, it uses a mechanical logic in which a query is a graph defining the workflow of a simple and compact algorithm that processes a specific temporal data structure. This data structure is simple and an easily implementable algorithm can automate its creation.

On one hand, the flexibility of Time Automaton's logic makes it a very powerful visual tool, but on the other hand it makes it less accessible than standard available visual tools. In comparison with available text-based querying languages, we think Time Automaton is much more accessible and comfortable.

The remainder of this paper is structured as follows: in Section 2 we briefly describe some related work regarding visual temporal querying tools. In Section 3 we describe the characteristics and implementations details of the Time

Automaton mechanism. Section 4 provides a practical application in which Time Automaton is particularly useful. In Section 5, a sample of temporal queries used for testing the mechanism is presented. Finally, in Section 6, we provide the main conclusions of this work.

2. Related Work

From all text-based querying languages, SQL [7] is, arguably, the most popular. Even though it is very flexible, in terms of temporal querying capabilities, this language has some limitations. In particular, since ordering is secondary in SQL, ordinal temporal queries like 'the second Saturday of every month' are not directly formulable. By directly formulable we mean using a simple syntax and without requiring an intricate combination of multiple queries.

To answer SQL's temporal querying limitations, some authors have proposed SQL temporal extensions. Among these, TSQL2 [8, pp. 127–146] is the most noteworthy. However, even though this extension is thoroughly defined in theory, to the best of our knowledge, it is not implemented in any publicly available data management system. TQuel [9] is another temporal extension, but for the querying language Quel [10]. It can be seen as the predecessor of TSQL2, and similarly to it, is not implemented in any publicly available data management system.

When compared to text-based languages, like SQL, visual querying tools have advantages in terms of learning effort, prevention of errors, comfort and accessibility. Such tools are commonly found in websites and information management systems, and in the most well-known cases, they follow the same logic of two popular solutions, the Dynamic Query [4, pp. 336–351] [5] [11] and TimeWheel [6].

DynamicQuery (see Figure 1 extracted from [4]) is a visual tool for temporal interval querying. It represents the timeline as a segment in which the user selects a temporal portion.

Since DynamicQuery is designed for interval querying only, expressions involving recurrence, such as 'every Saturday', are not formulable. Another limitation comes from the fact that the interface requires the complete representation of the timeline, which limits the ability to display very large intervals. Such limitation is more evident when working with fine temporal granularities, such as hours or minutes. In addition, since the properties of temporal moments are not shown, interval queries such as 'from the first

Saturday of January to the first Wednesday of February 2009' are also not directly formulable.

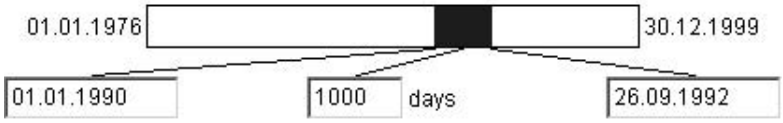

Figure 1: Dynamic Query representing the interval query 'from 1/1/19990 to 26/09/1992' - extracted from [4].

The TimeWheel (see Figure 2, extracted from [4]) is another visual temporal querying tool, aimed for cyclic temporal querying. It responds to the need of selecting recurring temporal events like 'every Saturday'.

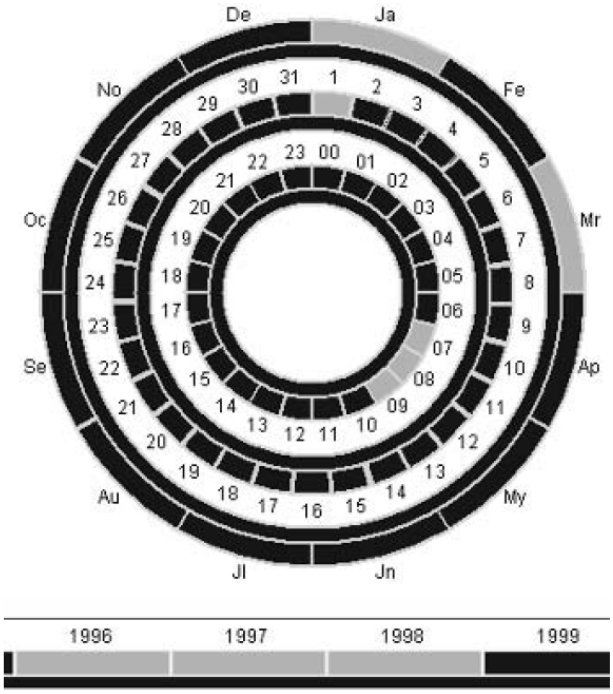

Figure 2: TimeWheel selecting 'the hours 7,8 and 9 of the days 1 of January and 1 of March of 1996,1997 and 1998 - extracted from [4]

One limitation of the TimeWheel is that it does not provide the means to formulate cyclic expressions involving temporal ordering, like for example,

'the second Saturday of every month of 1997' or 'every other Saturday in 1997'. In most websites or information systems that deal with temporal data it is common to find the TimeWheel's logic implemented as a series of check boxes in which users specify the properties of the required temporal moments (see Figure 3).

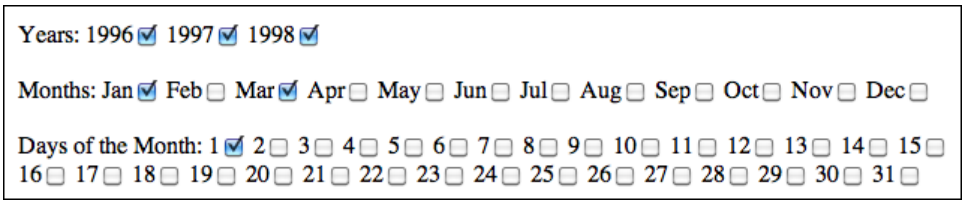

Figure 3: Standard checkbox based temporal querying interface, in which 'days 1 of January and 1 of March of 1996,1997 and 1998' are selected.

In the literature, most visual temporal querying solutions do not provide details on how queries are implemented and executed. From the few literature addressing this issue, a relevant work can be found in [12]. In that paper the author presents eight generic SQL [3] skeleton functions that are used to implement a predefined set of visual temporal queries. The set of queries just covers some basic temporal queries and it is not possible to combine them.
An inspirational work that influenced the creation of the Time Automaton is presented by Niemi et al. [13]. In that paper the authors propose a semi-formalized term representation to model temporal expressions as regular expressions. Regular expressions are intimately related with finite-state automata [14], which are models that are visually represented as graphs and define how a string including temporal data is processed. Despite the fact that their work is not focused on temporal querying, the idea of using a string to encode temporal data and employ regular expressions to interact with it, has strongly influenced our work.

3. Time Automaton

Time Automaton is a two tier mechanism formed by the temporal the temporal string and the query model.
The logic of the Time Automaton model is based on finite-state automata [14]. Finite-state automata are models, visually represented as graphs, that define how an input string is processed, in particular, which portion of the

string is read and which actions are performed when some symbols are read. Similarly, a Time Automaton query is a model, represented as a graph defining how an input string, built from temporal data, is processed.
These two main components of Time Automaton, the temporal string and the query model are described in detail in the following sections.

3.1. Temporal String

The temporal string (TS) is a sequence of words with temporal meaning, in which some of the words are temporal markers (named Anchors) while the others are temporal data (named Facts).
Anchors define the temporal structure of the temporal string and Facts are positioned according to the moment they occurred. The position of Anchors and Facts in TS follows a prefix notation. In our notation an Anchor defines the beginning of a temporal moment. A temporal moment may also serve as container to the Anchors of finer granularities. The Anchors that precede a Fact define moments in time in which the fact occurred. For a practical example, consider the artificial dataset and the corresponding TS, with month granularity (see [15] for more details), depicted in Figure 4.

Time	Data
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

Temporal String:
year,2009 month,Jan **fact,a,b** month,Feb **fact** month,Mar **fact,c,d,e,f** month,Apr **fact** month,May **fact** month,Jun **fact** month,Jul **fact** month,Aug **fact,g,h** month,Sep **fact** month,Oct **fact** month,Nov **fact,k,l,m,n** month,Dec **fact**
year,2010 month,Jan **fact,p,q** month,Feb **fact** month,Mar **fact,s**

Figure 4: Artificial dataset containing temporal information and the corresponding temporal string with a month granularity.

The terminology used to distinguish Anchors and Facts in the TS can vary according to the user's specification. In the example given in Figure 4, a Fact is described with the preceding word "fact" followed by the data itself, while the Anchors are described using a natural language terminology that makes its meaning explicit to the reader.
As can be seen in in the example above, the TS has a simple structure and,

therefore, an easily implementable algorithm [16] can automate its creation. For example, a possible approach could consist on using a Depth First Search algorithm on a tree in which Anchors are represented as nodes and Facts are represented as leafs. In this case Anchors with greater granularity are the ancestors of those with finer granularity and Facts are the tree leafs (see Figure 5 for an example).

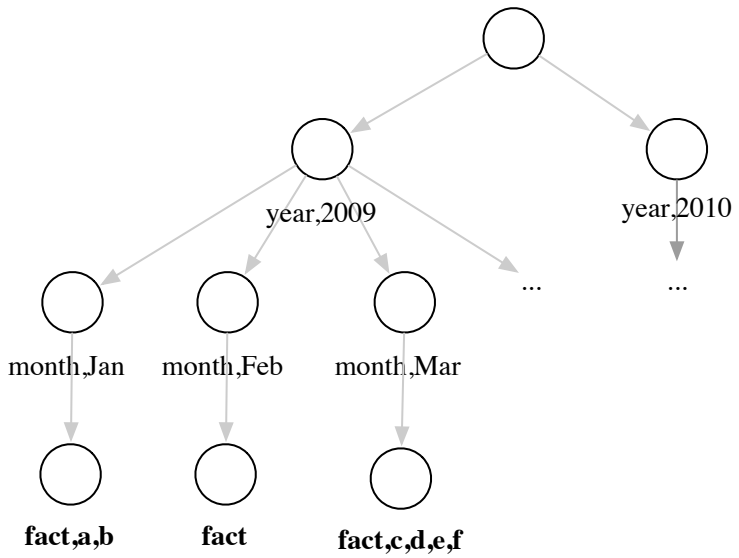

Figure 5: Tree structure that can be used for generating a Temporal String.

3.2. Query Model

In Time Automaton, a query is defined as a Connected Digraph [17] in which the vertices are words or expressions of the TS and the arcs define which word is the next to be read. This graph can be viewed as query in the sense that it defines where is the data (the portion of TS to read) and which data is retrieved (the Facts) to the user. Exploring the TS with a graph traversal procedure, the Time Automaton mechanism acts like a query execution algorithm.
In order to systematize and limit the number of possible configurations for the Time Automaton queries (graphs) we devised the following rules:

1. There is one root vertex, which only has outgoing arcs.

2. Every vertex other than the root is labeled with an expression, which corresponds to the definition of the syntax of a word in TS.

3. Every arc *(v,u)* may have a weight value that defines the maximum number of traversals from v to u.

For each particular query (graph), every time a word in the TS matches the expression of one of the direct successors of the current vertex, the graph is traversed to that vertex. Time Automaton keeps reading words until the last vertex of the graph is visited or TS is entirely scanned.
In order to describe and specify the syntax of the words to be matched, regular expressions [14] are used in the vertices' expressions.
Regular expressions are a highly flexible and compact instrument for matching strings of text, such as particular characters, digits, or punctuation. A regular expression is written in a formal language interpreted by a regular expression engine, which analyzes a text and identifies which parts match the provided specification.
One of the advantages of using regular expressions in Time Automaton is that all the burden of word matching is passed to the regular expression engine, which simplifies the implementation of a query execution algorithm. Additionally, the process of saving Facts into memory can be easily done with regular expressions. Finally, the flexibility that results from the combination between graphs and regular expressions allows Time Automaton to express some complex temporal queries.
In Time Automaton, the most basic type of regular expression is the literal definition, which corresponds to having all letters of the word explicitly defined. For example, the regular expression 'year,2010' literally defines all letters of the Anchor "year,2010". More complex definitions require more sophisticated regular expressions. In order to make the reasoning and writing processes more simple and avoid requiring users to learn how to use regular expressions, three high level predicates were created. These are defined as follows:

any() - Corresponds to the regular expression '$\backslash w+$'. It is used to match any sequence of letters. In the temporal querying context, this predicate is useful for creating queries that do not specify some properties of temporal moments. For example, in the query 'every second month', the month's name is not specified.

not(*exp*) - Stands for the regular expression '(?!\b*exp*\b)\b\w+'. In the temporal querying context, this predicate is useful for creating queries that exclude certain temporal moments, like for example 'every month except March'.

fact() - Corresponds to the regular expression "$fact, ?([\hat{}\backslash s]+)$'. It is used to match Facts and mark them to be saved into memory. In a regular expression everything that is enclosed by curved parenthesis is saved to the regular expression engine's memory and can be easily accessed afterwards. Specifically, the presented regular expression defines that when a Fact is read, the list of data records that follows the term "fact" is saved into memory.

The algorithm for executing Time Automaton queries can be straightforwardly implemented. Again, we highlight that the regular expression engine carries out all the word matching labor, which simplifies the implementation of the algorithm. Next, the query execution algorithm is presented (see Algorithm 1).

Algorithm 1 - Query_Exec(GQ, RE, TS)

/*

Input: a graph query GQ, a regular expression engine RE and a temporal string TS.

Output: a list containing all facts that match the provided graph query GQ.

Notes: The algorithm uses an object-oriented notation; For each arc of the graph for which the maximum number of traversals was not defined is assumed that it was initialized with a very large number; $RE.analyze(text, regex)$ is the method by which the regular expression engine analyzes a given $text$ against a provided regular expression $regex$. The method returns a boolean informing if the match was successful. The form, the inputs and output of this method is similar to the ones in the regex analyzing methods of several programming languages (e.g. Perl, Java or C++).

*/

$currNode \leftarrow GQ.root$

$RE.savedFacts \leftarrow \{\}$

```
for each word w in TS do
    listAdj ← currNode.adjacentNodes
    for each node adjNode in listAdj do
        if GQ.arc(currNode, adjNode).maxTraversals > 0 then
            match_ok ← RE.analyze(w, adjNode.label)
            if match_ok then
                maxTrav ← GQ.arc(currNode, adjNode).maxTraversals−1
                GQ.arc(currNode, adjNode).maxTraversals ← maxTrav
                currNode ← adjNode
                exit_this_loop
            end if
        end if
    end for
end for
return RE.savedFacts
```

Alternative algorithms could be used. The objective of the presented algorithm is to provide a compact description that can be simple to understand rather than being extremely efficient. A more efficient alternative, for instance, would consist on using an index structure for fast word searching, instead of scanning all words linearly.
Given the full description of the query model we can now provide some examples (see Figure 6) that illustrate how the model is put into practice. For these examples we use the TS depicted in Figure 4.

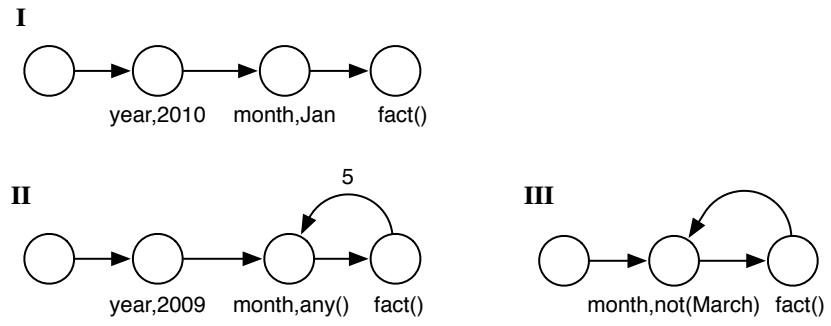

Figure 6: Basic examples of Time Automaton queries in which all predicates are used.

Query I retrieves the facts that occurred in January 2010. The TS is read until the word "year,2010" is found. Then, the algorithm searches for the next word, "month,Jan". Sequently, the predicate *fact()* matches and saves the next fact that is found. The result of this query is "p,q".
Query II provides the facts occurred in the first six months of 2009. The result is "a,b,c,d,e,f". Notice that this example has an arc with the weight value of 5, indicating that the corresponding traversal is performed five times.
Query III searches for the facts occurred in any month except March. In this query the regular expression engine captures all facts that follow a month Anchor with a name different from "March". The result is "a,b,g,h,k,l,m,n,p,q".

4. Practical Application

The original motivation for this work was the creation of a visual analysis framework for urban mobility data analysis, in which a Visual Temporal Querying Module is linked to a series of Spatial Visualizations. Such framework would empower urban and transport planners who, in general, do not possess programming skills to visually formulate complex queries and analyze how people have moved during the selected temporal moments. In a broad perspective, this framework aims to respond to one of the recent challenges in Geovisual Analytics, identified by Andrienko et al. [18]: give more attention to time and to users. With this they mean that Geovisual tools should be more "temporal" and they should be developed to be used by different types of users, not only by those that possess advanced computer competences.
The referred framework was implemented and tested for the analysis of an urban mobility dataset containing, for each day of 2009 and 2010, the number of bus ticket validations in 17 regions of a city (Porto, Portugal). These regions were named: **C1, C10, C11, C2, C3,C4, C5, C6, C8, C9, N1, N10, N11, N16, S1, S2 and S8**.
The Visual Temporal Querying module is an implementation of the Time Automaton. This tool and the Spatial Visualization are linked so that the results of the temporal queries are sent to the spatial visualization. For each region of the city a visual marker, colored and sized in proportion to the number of ticket validations, is displayed. More precisely, the color hue and the radius of markers are proportional to the percentage of ticket validations.
In order to use the Time Automaton, the urban mobility dataset was converted into a Temporal String, with day granularity. One of the objectives of the analysis was to study how the weather conditions affect urban mobility,

so a temporal property describing the weather condition was added to each day's Anchor. Next, we provide a small excerpt of the resulting Temporal String (one single day):

year,2010 month,Jan day,1,Tue,Rainy fact,C1,935,C10,1433,C11,11209, C2,2618,C3,555,C4,5628,C5,15366,C6,5435,C8,7693,C9,19,N1,233, N10,187,N11,535,N16,438,S1,956,S2,6924,S8,1002...

We have created an interface that allows the users to configure the layout of Time Automaton queries (see Figure 7 Top) and create the required regular expressions using only graphical controls (see Figure 7 Bottom). These controls are combo boxes that are automatically created and populated based on the content of the Temporal String. Also, there is a checkbox beside each combo box that, when checked, negates the selected property.

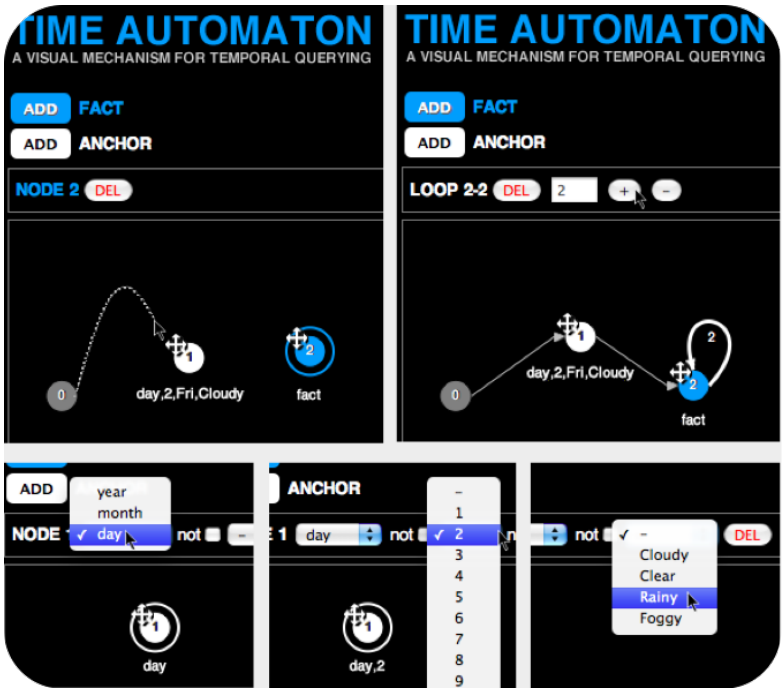

Figure 7: Top) Edition of the graph's layout. Bottom) Selection of the parts that compose node's 1 expression.

Next, in Figure 8, two queries and the corresponding visualizations are shown. These queries were formulated with the purpose of comparing the differences in urban mobility on clean and rainy days (see Figure 8). Based on this visualization it seems that on rainy days urban mobility is more concentrated at the center of the city. This can be related to the fact that central stops normally offer the best infrastructures, which are crucial on rainy and windy days.

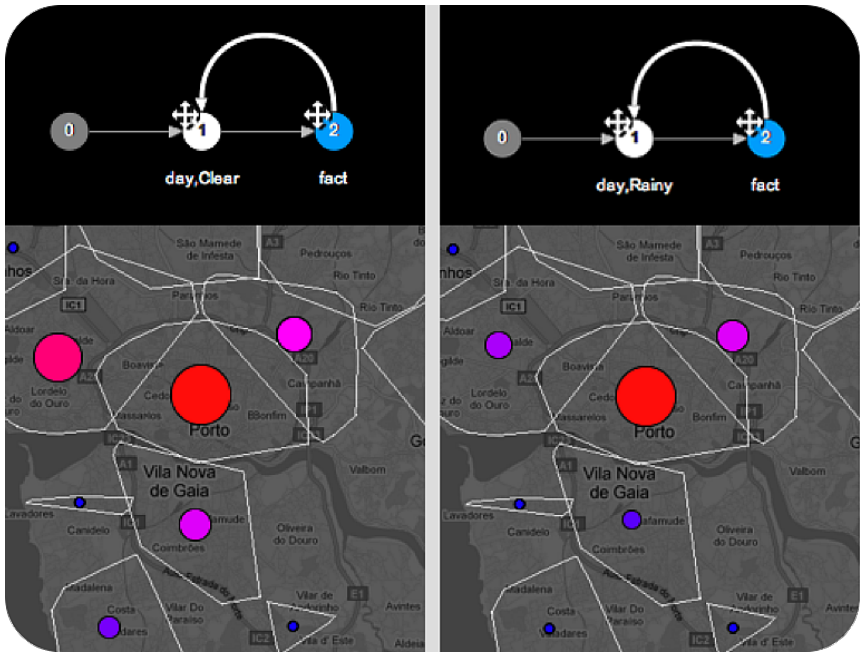

Figure 8: Two spatial visualizations used to compare the bus stops' activity on clean and rainy days.

We could also analyze how urban mobility is affected by some economic factors. For example, we have created the query that selects the validations on 'the weekend immediately after the day 23 of every month' (see Figure 9), which is the day when, in Portugal, public employees receive their salary. In this visualization shown in Figure 9 it is interesting to notice that the most active region corresponds to the location of one of the biggest shopping malls in the city. This may be an indicator that when people have more money they tend to travel more around shopping areas.

Figure 9: Spatial visualization of the bus stops' activity in the weekend after the payment of public employees' salary.

Besides the insights we extracted from this visual data exploration it should be noticed that the temporal questions we addressed were only possible to be visually formulated by using the Time Automaton mechanism. For the best of our knowledge, none of the available visual temporal querying solutions are capable to expressing such temporal queries.

5. Validation

In order to assess the Time Automaton capabilities we have created a sample of representative temporal queries based on the temporal expression types included in the OWL - Time Ontology [19] [20]. Such ontology is a body of knowledge that defines a thorough catalog of temporal expression types found in online content. Even though it is not focused on temporal querying we believe that this matured and well-established ontology provides the essential notions required to understand which types of temporal constructions are involved in temporal reasoning and, in our particular case,

in temporal querying.
For the presented sample of temporal queries we used a Temporal String with day granularity, containing data from 2009, 2010 and 2011, and including the following temporal properties: year's name, month's name, day of the month and day of the week. Next, we present the sample of representative temporal queries and the corresponding formulation. We will provide examples of queries that are possible to formulate using both the time automaton and the other visual mechanisms and examples of queries that can be formulated only with our mechanism.

5.1. Trivial queries

Query a) Select all facts occurred in every Monday.

In this basic temporal query a set of temporal instants that match a specified property is selected. Figure 10 depicts how this basic temporal query is formulated using Time Automaton.

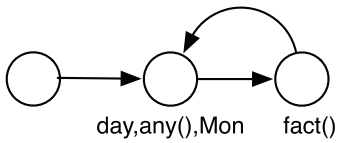

Figure 10: Temporal query that returns all facts occurred in every Monday.

For this basic query type, an alternative such as TimeWheel (see Figure 2) could also be used and with SQL this basic temporal query type is simple to formulate.

Query b) Select all facts in any day that is a Monday, a Wednesday or a Friday.

In this temporal query a set of temporal instants that match one of a series of temporal properties is selected (see Figure 11).

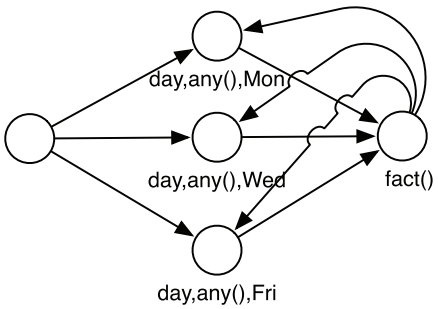

Figure 11: Temporal query that select all facts occurred in any day that is a Monday or a Wednesday or a Friday.

As seen be seen in the Figure above, for a query of this type a solution like the TimeWheel (see Figure 2) may be more usable than the Time Automaton. With SQL such a query is also simple to formulate.

Query c) Select all facts that occurred in the days between February 24, 2010 and September 2, 2011 (inclusive).

This query is a basic temporal query in which an interval is selected. Figure 12 depicts how Time Automaton can be used to formulate it.

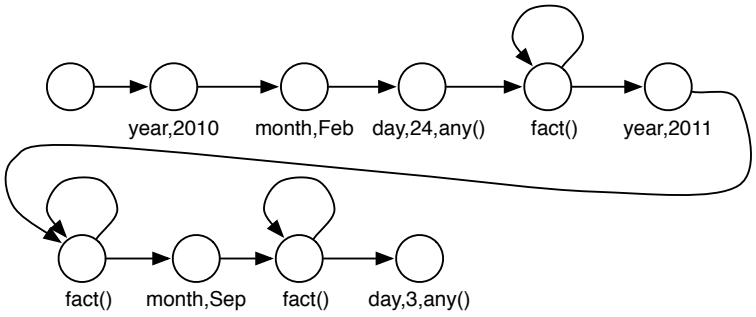

Figure 12: Temporal query that provides all facts that occurred in the days between February 24, 2010 and September 2, 2011 (inclusive).

This type of query can also be easily formulated with the DynamicQuery (see Figure 1) tool. In SQL this query type is also simple to implement.

Query d) Select all the facts that occurred between February 24, 2010 and the fifteen days afterwards.

This query, which is represented in Figure 13, also covers the basic operation of selecting an interval, but in this case it uses an instant and a duration. Since we are using a temporal string with a day granularity each fact has a direct correspondence to one day, therefore, selecting the fifteen facts after February 24, 2010 corresponds to selecting the facts occurred in the fifteen days after.

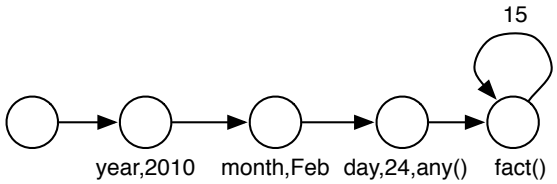

Figure 13: Temporal query that provides all facts that occurred between February 24, 2010 and the fifteen days afterwards.

For this basic query type the DynamicQuery (see Figure 1) may be simpler to use. In SQL this query type is also simple to formulate.

5.2. Complex queries

Query e) Select all the facts that occurred in each Monday, Wednesday and Friday.

In this query (see Figure Figure 14) Monday, Wednesday and Friday is treated as a single sequential pattern. Since Time Automaton's logic is based on the sequential reading of a temporal data structure, queries involving sequential patterns, even more intricate ones, are usually simple to formulate.

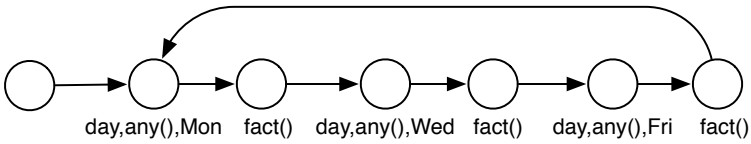

Figure 14: Temporal query that provides all facts occurred in each sequence Monday, Wednesday and Friday.

In other available visual solutions this type of queries is not possible to express. Moreover, with SQL this query is also difficult to formulate and it normally produces large blocks of code.

Query f) - Select all facts occurred in every second Monday.

This same query is another example involving a sequential pattern. As consequence Time Automaton works fine for this query too (see Figure 15).

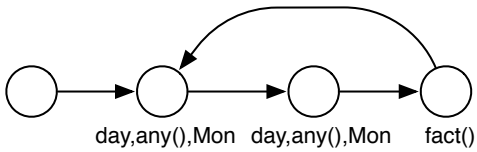

Figure 15: Temporal query that provides all facts that occurred occurred in every second Monday

This type of temporal query cannot be formulated using other available visual solutions. Also, using the basic syntax of SQL this query is impossible to formulate.

Query g) Select all the facts that occurred in the last two Mondays of every month.

This query, which the selects last *nth* instants inside a recurrent temporal period, is not possible to formulate using the Time Automaton. Still, it should be noticed that with the other available solutions (visual tools and

SQL) this query is also impossible to formulate.
The mentioned limitation in the Time Automaton is due to the fact that it's query model uses a prefix logic by which every temporal moment is entered after the Anchor that marks its beginning is read. As a general rule, Time Automaton cannot formulate queries that involve distances to temporal moments that are marked by an Anchor that has not been read yet.
A possible solution for this limitation could be to build an inverted temporal string in which the positioning of Anchors follows an inverted temporal order, or in other words, using a future to past direction. For example, in this inverted temporal string, the Anchor marking December would be positioned before the Anchor marking November.
By using an inverted temporal string, the mentioned temporal query would be possible to formulate with Time Automaton. However, for queries that use both past to future and future to past directions, like for example 'the two last Mondays of the first six months of 2010', the suggested solution would not work.
In a more radical approach we could incorporate new elements to the Time Automaton, but this would make the mechanism and the querying execution algorithm more complex and it could make it difficult to support them using exclusively the language of graphs.

Query h) - Select all facts occurred in every day between February 24, 2010 and the second Monday afterwards (inclusive)

In this case the temporal query selects an interval from a specific instant until the end of a temporal pattern (see Figure 16).

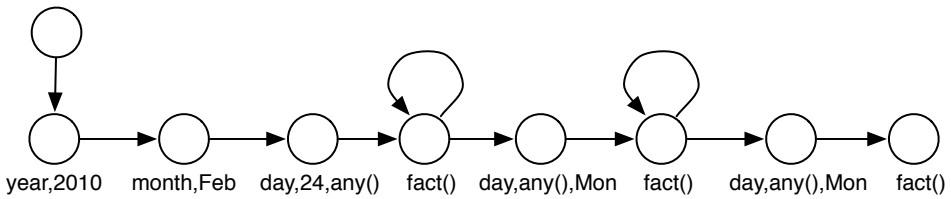

Figure 16: Temporal query that selects all facts that occurred every day between February 24, 2010 and the second Monday afterwards (inclusive)

Such a query is not formulable using the available visual solutions. With

SQL this query is possible to formulate but is not straightforward.
In this particular case, the temporal query involves a small temporal sequence. If the objective was to select 'all facts occurred in every day between February 24, 2010 and the 100th Monday afterwards (inclusive)' the Time Automaton query would be much more complicated (see Figure 17).

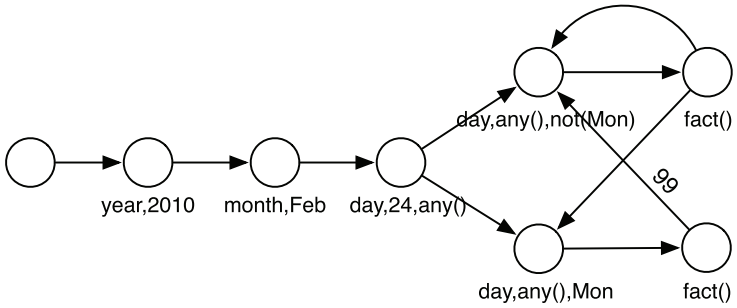

Figure 17: Temporal query that selects all facts occurred between February 24, 2010 and the 100th Monday afterwards (inclusive)

Query i) - Select all the facts that occurred every other Monday in every 4th month in every year.

In this query (see Figure 18) one recursion is contained in another. This temporal query illustrates how intricate temporal queries can be and how Time Automaton can be useful for such complex temporal constructions.

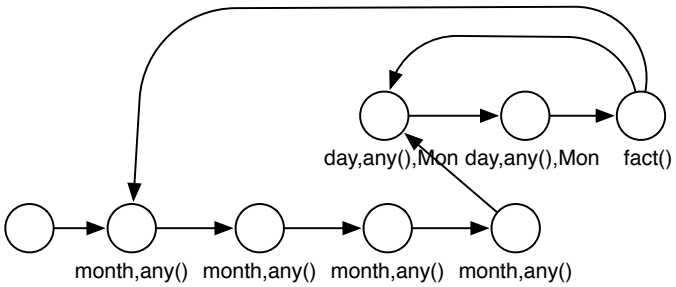

Figure 18: Temporal query that selects all facts that occurred every other Monday in every 4th month in every year.

Such a complex temporal query cannot be expressed using the available visual solutions and, for the best of our knowledge, it is also impossible to formulate using SQL.

5.3. Discussion

Trivial queries can be formulated with the Time Automaton, however, for these types of queries the other available visual solutions may be easier to use. Time Automaton is specially convenient for complex temporal queries, in particular, those involving sequential patterns like for example 'every second Monday' or ' every other Monday in every 4th month in every year'. For these type of queries available visual tools and SQL are of no use.

Temporal queries involving sequential patterns can be frequently found in studies regarding cause-effect relations like for example in the query 'select all facts occurred in the weekend immediately after the day 23 of every month', depicted in Figure 9, in Section 4.

In addition to it's versatility, Time Automaton is also advantageous in terms of query implementation. Since queries are naturally configured as a graph layout, designing an SQL backend for the translation and execution of temporal queries is not necessary. This is very convenient because some temporal queries are not simple to formulate using SQL operators. As an example, we now show how the Query h, presented in Figure 16, is formulated using SQL. For this SQL implementation it is assumed that there is a table with a column named 'time' that includes the date of data records and a column named 'data' including the data itself. It is also assumed that every day is represented in this table, i.e., there is at least one data record for each day. The provided SQL implementation is written in MySQL, which is one of the many different available SQL DBMSs, and like others, it includes generic SQL operators and specific MySQL operators. The presented temporal query uses two of those specific operators: DAYNAME and LIMIT.

```
SELECT data FROM times
WHERE time > '2010-24-10' AND
time < ( SELECT DISTINCT(time) FROM times
WHERE time > '2010-24-10' AND
DAYNAME(time) = 'Monday'
ORDER BY time LIMIT 1,1 )
```

As can be seen in Figure 16, a temporal query that can straightforwardly expressed by the Time Automaton can be much more difficult to formulate using SQL. Moreover, if one thinks about the endless number of combinations between different types of temporal queries it becomes evident how challenging the SQL implementation of some temporal queries would be. Most important, we should keep in mind that SQL implementations are not accessible to users that do not possess advanced programming skills, who may be experts in the field under analysis.

6. Conclusions

In this paper we present the Time Automaton, a highly flexible visual mechanism that is capable of formulating a vast set of temporal queries.
Time Automaton is inspired in finite-state automata [14], which are models that are visually represented as graphs defining how an input string is processed. Similarly, a Time Automaton query is a model, visually represented as a graph, that defines which portion of an input string containing temporal data is processed.
The structure of the string used in Time Automaton has a plain-text format and an easily implementable algorithm automates its creation. Using a plain-text data format makes temporal datasets highly portable, which is advantageous in scenarios where data sharing is required, like for example, in team working. Data enrichment is also facilitated by the format's simplicity. Hence, if analysts need to formulate temporal queries based on temporal properties such as the weather conditions, or the seasons of the year, their inclusion in the temporal string is quite easy.
The algorithm to execute Time Automaton is simple to implement, since it is nothing more than a simple graph traversal algorithm, involving some basic rules.
Time Automaton does not use any available text-based querying language to execute queries. This independence enabled us to create an innovative logic that is both visually coherent and capable of expressing a significant set of different types of temporal constructions. In order to test Time Automaton's logic we created a sample of representative temporal queries including all types of temporal expressions contained in the matured OWL-Time Ontology [19] [20]. Almost all queries in the sample are formulable with the Time Automaton mechanism and the majority of them are impossible to

formulate using the available visual temporal querying solutions.
For simple queries Time Automaton may not be as user friendly as other available visual solutions. However, it allows a larger set of temporal query types than the other approaches and it can be particularly useful in contexts where analysts need to perform complex temporal queries but do not possess programming skills.

References

[1] G. Andrienko, N. Andrienko, S. Wrobel, Visual analytics tools for analysis of movement data, ACM SIGKDD Explorations Newsletter 9 (2) (2007) 38–46.

[2] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, A. Tomkins, Visualizing tags over time, in: Proceedings of the 15th international conference on World Wide Web, WWW '06, 2006, pp. 193–202.

[3] J. Groff, P. Weinberg, SQL The Complete Reference, 3rd Edition, 3rd Edition, McGraw-Hill, Inc., New York, NY, USA, 2010.

[4] N. Andrienko, G. Andrienko, Exploratory analysis of spatial and temporal data: a systematic approach, Springer Verlag, 2006, Ch. 4.6.1.2.

[5] C. Ahlberg, B. Shneiderman, Visual information seeking: tight coupling of dynamic query filters with starfield displays, in: Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence, ACM, 1994, p. 317.

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

[7] E. Codd, A relational model of data for large shared data banks, Communications of the ACM 13 (6) (1970) 377–387.

[8] R. Snodgrass, The TSQL2 temporal query language, Kluwer Academic Pub, 1995.

[9] R. Snodgrass, The temporal query language TQuel, ACM Transactions on Database Systems (TODS) 12 (2) (1987) 247–298.

[10] D. Maier, The Theory of Relational Databases, Computer Science Press, 1983, Ch. 15.

[11] H. Hochheiser, B. Shneiderman, Dynamic query tools for time series data sets: timebox widgets for interactive exploration, Information Visualization 3 (1) (2004) 1–18.

[12] L. Chittaro, C. Combi, Visualizing queries on databases of temporal histories: new metaphors and their evaluation, Data & Knowledge Engineering 44 (2) (2003) 239–264.

[13] J. Niemi, L. Carlson, Modelling the semantics of calendar expressions as extended regular expressions, in: FSMNLP, 2005, pp. 179–190.

[14] J. E. F. Friedl, Mastering Regular Expressions, 2nd Edition, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[15] J. Hobbs, Granularity, in: In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Citeseer, 1985.

[16] T. H. Cormen, C. Stein, R. L. Rivest, C. E. Leiserson, Introduction to Algorithms, 2nd Edition, McGraw-Hill Higher Education, 2001, Ch. 22.3, pp. 540–549.

[17] M. Golumbic, Algorithmic graph theory and perfect graphs, North Holland, 2004.

[18] G. Andrienko, N. Andrienko, D. Keim, A. MacEachren, S. Wrobel, Challenging problems of geospatial visual analytics, Journal of Visual Languages and Computing 22 (2011) 251–256.

[19] J. Hobbs, F. Pan, An ontology of time for the semantic web, ACM Transactions on Asian Language Information Processing (TALIP) 3 (1) (2004) 66–85.

[20] F. Pan, J. Hobbs, Temporal aggregates in OWL-Time, in: Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS), 2005, pp. 560–565.