FORMALIZING VISUAL INTERACTION WITH HISTORICAL DATABASES

SONIA FERNANDES SILVA¹, TIZIANA CATARCI² and ULRICH SCHIEL³

¹Dep. de Engenharia Elétrica, Universidade Federal da Paraíba C.P.10053 Campina Grande PB, Brasil
²Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza” L00198, Rome, Italy
³Dep. de Sistemas e Computação, Universidade Federal da Paraíba C.P.10106 Campina Grande PB, Brasil

(Received 18 June 1995; in final revised form 1 November 1995)

Abstract — Recent database applications are typically oriented towards a large set of non-expert users, and therefore, they need to be equipped with suitable interfaces facilitating the interaction with the system. Moreover, the incorporation of the time dimension in database systems is a desirable feature. Indeed, several temporal data models and the corresponding textual query languages have been proposed. However, there is a limited amount of research concerning the investigation of user-oriented languages for querying temporal databases. Our proposal addresses such a need. In particular, we propose a visual query environment, namely TVQE (Temporal Visual Query Environment), which provides an easier interaction of the user with temporal databases. The system adopts a diagrammatic representation of the database schema (including temporal classes and relationships) and a “graphical notebook” as interaction metaphor. In our approach, non-database experts are released from syntactical difficulties which are typical of textual languages, and they can easily express temporal queries by means of elementary graphical operations (e.g. click on a node label). Differently from many proposals in the field of visual query languages, the language underlying TVQE is provided with formal syntax and semantics. It is based on a minimal set of temporal graphical primitives (TGP), which are defined on a Temporal Graph Model (TGM), with visual syntax and object-based semantics. In this paper we mainly concentrate on the formal aspects of TVQE, and provide some hints on the visual interaction mechanisms and implementation issues.

1. INTRODUCTION

The availability of graphical devices at low cost and the advent of direct manipulation paradigm [41] have given rise in the last years to a large diffusion of interfaces using visual techniques.

Concerning the database area, databases are designed, created and possibly modified by experts, but there are different kinds of users whose job requires access to databases specifically for extracting information.

Thus, visual interfaces for databases, in particular, the so-called Visual Query Systems - VQSs (see [12] for a survey), have arisen as alternatives to traditional query languages, such as SQL. VQSs include both a language to express queries in a visual formalism and a query strategy. They are oriented to a wide spectrum of users who generally ignore the inner structure of the accessed database.

VQSs can be seen as an evolution of traditional query languages provided by DBMSs. They are characterized by several notable features, such as:

- The use of icons and visual metaphors, which attract the user’s attention and stimulate her/his curiosity.
- The unnecessary for the user of having a previous knowledge of the database schema and being able to use a query language.
- The availability of interactive mechanisms to support the typical process of query formulation. Such a process can be seen as constituted by three phases [9]: the user selects the part of the database s/he wants to operate on (location phase); then, s/he defines the relations within the selected part in order to produce the query result (manipulation phase); finally, s/he operates on the query result (visualization phase).
Existing VQSs essentially deal with conventional query operations, while there is a limited amount of research concerning the investigation of friendly environments for querying temporal databases, in spite of numerous papers that were published considering the temporal factor relevant in several database applications such as banking, medical records, geographical and multimedia systems, decision support systems, etc (see [46] for an up-to-date bibliography on temporal databases).

Conversely, there are many proposals of temporal textual languages, where special clauses and predicates are added to the original language in order to deal with the temporal aspects (important references can be found in [13], [37], [47], [46]).

These languages retain the usability problems of the originating query languages, such as:

1. The intrinsic syntactical complexity;
2. The lack of a global view of the data of interest together with their interrelationships;
3. Considering specifically the temporal relational languages, the semantical complexity of the adopted concepts. For instance, the user must be familiar with concepts such as tuple, attribute time-stamping and temporal joins, as well as syntax and semantics of temporal predicates.

As efforts were made to find new visual query mechanisms for accessing conventional databases, this should be done for temporal databases. Adequate conceptual schemata are needed and new visual mechanisms must be found in order to manipulate the temporal aspects.

While attempting to meet this need, we have initially developed a VQS described in [24] as an effort to put in an easy-to-use visual form the task of formulating queries on databases (including temporal databases). Then, we have extended the original VQS with new features, such as:

- Hierarchical visualization of complex schemata;
- Use of a familiar metaphor in order to facilitate the query specification [22];
- Intensional visualization of the query result;
- Visualization of the query result by using the dynamic query approach introduced in [1], [42];
- Processing of data about the user's interaction within the query environment (preliminary results can be seen in [19], [21]).

The extended version of the system has been called Temporal Visual Query Environment (TVQE). The main idea of the system is to provide the user with a simple visual environment for formulating conventional as well as temporal queries. In such an environment the different query specification activities are performed in a homogeneous way, through elementary graphical operations, exploiting a model-independent visual metaphor. In this way, non-expert users do not need to understand neither the underlying data model nor the syntax and semantics of a textual query language.

This paper completes and extends the results published in [20], [23] on the first three system extensions above. We also describe the data model the system is based on, namely the Temporal Graph Model (TGM), and the corresponding set of temporal graphical primitives, by showing both examples of usage and formalization. [20]). Finally, implementation issues are discussed.

This paper is organized as follows. Section 2 describes the TVQE architecture. Section 3 sketches the concrete scenario of TVQE, in order to show how a temporal query can be visually expressed. Section 4 summarizes the temporal extension of the adopted data model. Section 5 introduces the temporal graphical primitives. Section 6 describes how the result of the execution of the query is produced. Section 7 is about implementation issues of TVQE. Section 8 deals with some related work. Finally, Section 9 draws the conclusions and gives the research directions of this work.
2. TVQE ARCHITECTURE

TVQE has been developed in the context of a global system architecture [11] which allows one to visually interact with heterogeneous databases through an adaptive visual interface. Each query can be stated through different interaction modalities (switching among different visual representations is allowed during the query formulation) and its formal semantics is given in terms of the TGM query primitives. Translation algorithms provide conversion of visual queries to the query language of the various underlying DBMSs. Hence, the end-user is not conscious of the existence of several heterogeneous databases whenever accessing them.

More specifically, the global system architecture is illustrated in Figure 1, with the following basic functionality. Based upon the user model provided by the User Model Manager, the Visual Interface Manager selects the visual representation most appropriate for the user. The User Model is responsible for collecting data and maintaining a knowledge base of the user model components, namely the class stereotype, the user signature, and the system model. The Visual Interface Manager and the User Model Manager are described in more detail in [8].

At the bottom of the figure, different databases structured according to several data models are shown. Each database is translated into a Temporal Graph Model Database, TGMDB, through the TGMDB & Query Manager, using the mappings described in [11].

It is up to the TGMDB & Query Manager to manage such mappings and to translate the visual queries into queries that can be executed by the appropriate DBMS.

The user-system interaction proceeds as follows. The user visually specifies a query (conventional or historical) through the TVQE interface. The graphical schema (TGM schema) is the basic instrument on which the user formulates a query. Actually, the query is represented as a subschema of the initial schema. The Translator module converts the subschema of interest into an internal representation that the DBMS can process. The Data Visualization module analyses the query result, chooses the most appropriate interactive visualization (time-oriented or not) and presents the user with it. Note that such a module is out of the scope of this paper. Issues related with the user model management and the interaction with heterogeneous databases are also out of the scope of this paper.

3. VISUALLY EXPRESSING TEMPORAL QUERIES

In this section, we describe the main features of the environment to visually express queries on temporal databases. First a taxonomy for temporal queries in historical databases is presented.
Then, we introduce an example of user-system interaction, which refers to a historical database concerning an employment agency, with the following basic functionalities (for the sake of simplicity, the conceptual schema of this database contains a few classes and relationships).

An employment agency searches employees for its customers (enterprises). Every person is registered as an employment candidate and the agency gets the history record of the candidates. As soon as the enterprises provide the agency with some requirements (e.g., teams which need new employees for a specific project and with a certain salary), the agency analyzes the candidates so that it can select qualified employees, according to the enterprise request. Suitable candidates are then chosen and become employees, being assigned a specific task and project. We will use this example throughout the paper.

The envisaged user of TVQE environment has the following features: s/he only occasionally interacts with the database, but s/he is familiar with the database content. Also, we assume that the user may have superficial database knowledge, (including notions about classes and attributes which are components of conceptual schemas and some representation of historical information) as well as having already interacted with direct manipulation interfaces. Nevertheless, we assume the user will undertake some short training about the environment functionalities.

3.1. A Taxonomy of Historical Queries

A taxonomy for temporal queries was proposed in [32]. According to [32], a temporal query has two orthogonal components: temporal selection and temporal projection. Temporal selection is a logical condition, based on a predicate that involves the time associated with facts, and temporal projection returns the time values associated with data derived from temporal selection.

A new taxonomy for temporal queries has been proposed in [15]. In [15] the possible combinations between temporal selection/projection over time and data were analysed resulting into: data selection/data projection, where conditions and results apply to data values only; temporal selection/temporal projection, where conditions and results apply to temporal values; and mixed selection/mixed projection, where conditions and results apply to both data and temporal values.

Since we consider only the valid-time measure [43], the combinations of data/temporal/mixed selection and projection, described in [15] are applied to two histories: past instantaneous data and historical data, as can be seen from Table 1. Using TVQE, the user may specify the various types of temporal queries in Table 1.

Note that we do not consider the combinations data selection/data projection and temporal selection/temporal projection (the former represents a conventional query and the latter is impossible to occur in a temporal query since conditions and results do not apply to temporal values only).

<table>
<thead>
<tr>
<th>Data Selection/Temporal Projection</th>
<th>Past Instantaneous Data</th>
<th>Historical Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Selection/Mixed Projection</td>
<td>ex.3</td>
<td></td>
</tr>
<tr>
<td>Temporal Selection/Data Projection</td>
<td>ex.1</td>
<td></td>
</tr>
<tr>
<td>Temporal Selection/Mixed Projection</td>
<td></td>
<td>ex.6</td>
</tr>
<tr>
<td>Mixed Selection/Data Projection</td>
<td>ex.2</td>
<td></td>
</tr>
<tr>
<td>Mixed Selection/Temporal Projection</td>
<td>ex.7</td>
<td></td>
</tr>
<tr>
<td>Mixed Selection/Mixed Projection</td>
<td>ex.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Different Query Types

In the following examples we illustrate seven distinct cases in Table 1:

1. What were the salaries of employees at 10/01/95?

2. What was the salary of John Smith when he changed his status?

3. Since when does John Smith own more than 5,000?
4. What is the history of the salaries of the database group?

5. What was the last salary of employees who started working for more than 1,000 and since when?

6. What is the history of the salaries of the employees during the period 01/01/95-03/25/99?

7. Which is the period that John Smith worked at personal department during 1995-1998?

3.2. TVQE Initial Design

The TVQE interface was initially designed as a main window containing two panels, namely Schema window and Query window.

The Schema Window is basically constituted by some pull-down menus for accessing a database schema and a panel on which the database schema is visually represented. Through the Schema Window, the user selects a class as the target class of the query [34]. Then, the schema, rooted at the selected target class, is visually represented in the panel of the Query Window, which corresponds to the manipulation phase of the query.

In the following, we define the concept of context, which is still used in the current design.

Since visualizing all classes of a complex schema in a single visual structure may be cumbersome, the Schema window displays the database schema as a top-down tree, a so-called context tree. The user’s navigation on the context tree exploits the concept of top-down refinement of an E-R schema defined in [4].

In our approach, we start from a context \( C \). A context represents a concept in an abstract way, but, differently from an entity, it does not contain real-world objects as its instances. It can be seen as an abstraction of a set of classes. A context \( C \) is refined into a set \( C_1, \ldots, C_n \), where each \( C_i \) may represent either a more specific context or a class. If \( C_i \) is in turn a context, it must be refined into another set.

Hence, the Schema Window displays the conceptual schema as one level tree having its root as a context and its leaf nodes representing other contexts or classes, as shown in Figure 2. Note that contexts and classes are visually represented as dark gray squares and light gray squares respectively, while temporal classes are shadowed squares.

![Fig. 2: Initial visualization of a conceptual schema](image_url)

Since the user does not visualize the desired class of the query in the first refinement of the schema in Figure 2 (left), s/he may expand a context, just by clicking on it. As a consequence, a new panel appears with the refinement of the selected context, as shown in Figure 2 (right). When the user selects the target class (class EMPLOYEE) and the go to query button, the user can specify its query. Next, the user interacts with the query window.
The Query Window is composed by a panel on which a sub-schema is visually displayed. Such a sub-schema contains the target class which has been selected in the Schema Window and all its properties.

Since the target class may contain several properties, the sub-schema in the Query Window may be visually represented as a top-down tree, a so-called property-group tree, similarly to the context tree used for representing the database schema in the Schema Window. The difference being that a property-group in the Query Window may be refined into a set $C_1, C_2, \ldots, C_n$ of classes directly reachable from the target class, where each $C_i$ may be a printable class (whose instances are domain values), an unprintable class (whose instances are object identifiers) or a temporal class. If $C_i$ is either an unprintable or a temporal class, it may be expanded in terms of its properties.

For example, Figure 3 (left) illustrates the target class EMPLOYEE as root and the property-groups PERSONAL-ATTRIBUTES and HISTORICAL-ATTRIBUTES as leaf nodes, while Figure 3 (right) illustrates the refinement of the property-group PERSONAL-ATTRIBUTES into a set of printable and unprintable classes, visually represented as oval nodes and square nodes respectively. Note that the binary role-nodes between the target class and the associated classes are displayed as arcs.

![Fig. 3: The Query Window](image)

An n-ary relationship is visually represented as an unprintable class (square node) and it is refined into its associated classes.

A temporal relationship is also refined into its associated classes and is visually represented as shadowed oval node, in order to distinguish it from a temporal class. Figure 4 (left) shows the property-group HISTORICAL-ATTRIBUTES refined into a set of temporal relationships formed by LEVEL-HISTORY, JOB-HISTORY and COMPANY-HISTORY.

Moreover, nodes in the panel may assume three states, unselected for the query, selected for the query, which means included in the schema of interest, and displayed, which means included in the query result. It corresponds to coloring them light grey, red and black respectively. The user can continuously change the status of a node by clicking on it. As a consequence, the selected node will assume the corresponding color. In our example, LEVEL-HISTORY is the selected class, while EMP-ID and NAME are displayed.

Operations filtering (historically or not) values of target class properties are visually represented by menu items, icons, radio buttons, sliders, etc. For example, Figure 4 (right) shows a temporal condition over a time period.

More details about the initial interface of the system can be seen in [18].
3.3. TVQE Improved Design

The initial interface has been improved based on the suggestions of users who have participated in a preliminary experiment with the TVQE prototype (in this paper, we do not report this experiment with more details).

Some problems were detected in this phase:

- Users did not feel comfortable with the direct manipulation of nodes on the conceptual schema;
- Since the schema was visualized in a compact way as a tree of contexts and classes, there was not a complete visualization of the detailed conceptual schema.

Aiming to solve the above problems, we have enriched TVQE with an iconic representation of the database schema, which exploits a "graphical notebook" metaphor. The specification of some queries by searching indices in a graphical notebook (see the following sections) has been enjoyed by the users more than the direct manipulation on the diagrammatic representation, since s/he prefers to interact with something more familiar. Moreover, icons have a significant metaphorical power and it is shown that iconic VQSs are mainly addressed at users who are not familiar with the concept of data models and may find it difficult to interpret even an E-R diagram.

However, the visual representation of a schema as a notebook is in some sense less rich than the diagrammatic one, since a diagram favors the visualization of relationships between concepts. So, we decided to integrate the diagrammatic and iconic representations, emphasizing the usage of the iconic representation as the interaction media.

The new TVQE interface presents the user with a main window (whose layout is shown in Figure 5), which comprises the following components:

1. Tool area, which contains a set of buttons for accessing a database schema and the following icons: Where? and When? icons, representing the data selection and temporal selection/projection, respectively; Data Vis and Time Vis icons, where the objects which are current and historical instances of a class or relationship are visualized.

2. Schema visualization area and Interaction area, called Schema Window and Interaction Window, respectively (the two windows will be described in more detail later on).

Through the use of dialog boxes, the user specifies the query condition. For example, Figure 6 shows the layout of the dialog box used in a temporal condition, which comprises the following components:

1. Option panel of a temporal condition, which contains the all history (i.e., the entire lifespan of a selected object, default option), instant (i.e., a specific time point) and period (i.e., a specific time interval) radio buttons and a menu of temporal references.
2. Panel which contains a menu of time granularities (year, month, day, etc).

3. Panel which contains spin boxes for specifying a temporal constant (the number of active spin boxes depends on the granularity previously selected by the user).

4. Panel which contains a slider for specifying a predicate on a time instant (such a slider is activated when the user selects the instant option).

5. Panel which contains two sliders for specifying a predicate on a time period (such sliders are activated when the user selects the period option).

6. Panel on which the user may filter the time intervals, by selecting the first, nth-interval or last interval radio buttons and by specifying whether the selection concerns the initial, final instants or the duration of each interval (represented by the buttons begin, end and duration). If the duration is selected, the system offers a pop-up menu of aggregate functions, min, max, count, avg and sum.

7. Panel which contains a list box used for specifying temporal aggregations.

8. Panel which confirms or cancels the specified temporal condition.

The query result will be visualized in the Data Visualization window. In this paper, we concentrate on the Schema and Interaction windows, the Data Visualization window is under development.

3.3.1. The Schema and Interaction windows

The Schema window is a panel on which the database schema is visually represented as a top-down tree, as a graph and as a subgraph, where the three representations share the same panel. Through the Schema window, the user has a global view of the classes of the schema and their interrelationships.
Formalizing Visual Interaction with Historical Databases

Fig. 6: Dialog box used in a temporal condition

Exploiting the concept of context (see Section 3.2) the Schema window displays the conceptual schema as a top-down tree, having a context as root and its leaf nodes representing other contexts or classes, as shown in Figure 7 (left). Note that contexts, classes and temporal classes are visually represented as square nodes, oval nodes and shadowed oval nodes respectively.

The Schema window also displays the database schema either as a graph, namely Graph Schema, as shown in Figure 8 (left), or as a subgraph, namely Query Schema, as shown in Figure 9 (left), which represents the subgraph of interest comprising only the classes and relationships selected by the user for his/her query.

In the Graph Schema, we have square, rounded square and oval nodes representing classes (e.g., PERSON), attributes (e.g., NAME) and relationships between classes (e.g., LIVES), respectively, while shadowed square, rounded square and oval nodes represent temporal classes (e.g., EMPLOYEE), temporal attributes (e.g., LEVEL) and temporal relationships (e.g., JOB), respectively. Moreover, is-a relationships are expressed through the edge label "is-a" (e.g. EMPLOYEEis-aPERSON).

The Interaction window is a panel on which a database schema is visually displayed as a "graphical notebook" (see Figures 7 and 8). Whenever the notebook sheet represents a context, the notebook indices represent other contexts or classes, whereas, whenever the notebook sheet represents a class, the notebook indices represent all its properties (attributes and relationships). The user interacts with the graphical notebook by selecting its indices.

Through the Interaction window, the user selects a class as the target class of the query. Then, the schema is visually represented as a Graph Schema in the Schema window.

3.3.2. Example

We show the user-system interaction by means of a simple temporal query, which contains only one target class.

Assuming that the user is interested in knowing "Which salaries did the employees earn when they changed their level for the last time". First, s/he accesses a database schema (selecting
Fig. 7: The selected nodes EMPLOYMENT-CONTEXT and EMPLOYEE

the Open button]. The notebook indices are two contexts, namely PERSONAL-CONTEXT and EMPLOYMENT-CONTEXT, which represent information about personal and employment data, respectively.

The user may expand a context, just by clicking on it (more precisely, by clicking on the index that represents such a context). As a consequence, a new sheet appears with the refinement of the selected context, as shown in Figure 7, where the user has selected EMPLOYMENT-CONTEXT.

Note that the two windows are synchronized: for each selected index, the corresponding node in the Schema window is also selected. After the selection of a context, the user may switch to a different context (or class). S/he may also return to the previous refinement or to the first refinement by clicking on the back or begin buttons.

Since the user has selected the class EMPLOYEE as the target class, it appears as a sheet and the indices represent its properties, as shown in Figure 8 (the system automatically displays the Graph Schema in the Schema window). The user may expand a relationship in order to let appear the classes and attributes directly reachable from the class EMPLOYEE through that relationship.

A related class may be expanded in terms of its properties.

As in the first version of the interface, nodes in the panel in the Schema window may assume three states, unselected for the query, selected for the query and displayed for the query result. The user can continuously change the status of the node, by clicking on the corresponding index, through which the status of a node is switched from the value unselected, to either the value selected, or displayed. In our example, JOB and LEVEL are the selected nodes, while SALARY, EMP-ID and NAME are displayed. Next, the system displays the sub-schema, as shown in Figure 9.

The user can either save the corresponding sub-schema for further manipulation (selecting the Save As button) or immediately use it (selecting the Create View button). The user may create several views by selecting other target classes, and relate them in the same query. This approach facilitates the specification of a complex query which can be partitioned into several simple queries.

Considering the current specification of an attribute, the user may select the attribute and the Where? icon. As a consequence, a dialog box called “Domain Editor” appears, as shown in Figure 10. The dialog box is parametrized according to the type of the attribute, such as, string or menu of predefined values, and boolean expressions can be specified on the attribute. For example, Figure 10 shows the effects of the selection of the CITY-NAME attribute and the Where icon.
Our query involves a temporal selection with a temporal reference to another data, since it compares the history of certain data (salary) with the history of other data (level). So, the user specifies it by selecting the LEVEL temporal attribute and the When? icon. Next, a dialog box with the all history, instant, period options and a temporal reference to... check box appear (see top of Figure 11).

Since the history of levels of each employee may contain several time intervals, the user may filter them, by selecting the first, nth-interval or last interval and by specifying whether the selection concerns the begin, end instants, or the duration of each interval. If the duration is selected, the system offers a menu of aggregate functions, min, max, count, avg and sum, which applies to the set of instances in the temporal class or relationship. In the example, the all history option and the last interval of the level history have been chosen (Figure 11).

The spin boxes and sliders of the dialog box are used to specify the temporal condition. The spin boxes are used for specifying either an instant or a period. The number of active spin boxes depends on the granularity previously selected by the user. The next slider is used when the user selects the instant option. Let t be a instant. The begin and end operators retrieve only the instances whose time intervals start and finish at t, respectively, whereas with at, t must be in between the time intervals of the object.

The next two sliders are used when the user selects the period option, which contains the temporal predicates between time intervals, defined by Allen [3] (e.g. before). After the user has specified the desired period, s/he may use either the slider which contains the nine operators or the other slider with three operators.

The order of the temporal operators in the first slider is based on the concept of neighbor temporal primitives, introduced in [26] and further discussed in [29]. Two temporal relationships are neighbors if a continuous change of the events transforms a relation into another without passing through an additional temporal relationship [26].

For example, Figure 12 illustrates the temporal relationships During and Finish 08/26/1996 to 05/19/1998 (the figure only shows the slider with nine operators). Note that the white square located above the slide bar represents the period of the query, and the spatial location of the rounded grey square with respect to the white square evokes the temporal relationship among them (e.g. when selecting the cross (overlaps) operator, the rounded square partially overlaps the white square).
Since the operators `finished-by`, `contains` and `started-by` are not part of such an order, they were included in the second slider.

Figure 13 also shows the effects of the user’s selection of the `When?` icon applied to the temporal relationship `JOB`. In the example, the user selects the `instant` option and the `temporal reference to...` check box. S/he also selects the `LEVEL` option in the menu of temporal references. As a consequence, the label “Level” appears above the instant slider, where s/he has selected the `at` operator, as shown in Figure 13.

At this point, the temporal condition has been completely specified, and the system generates the subgraph of interest that is then translated into a SQL query.

### 4. THE TEMPORAL GRAPH MODEL

In this section we present the *Temporal Graph Model (TGM)*, i.e., a graph-based formalism for representing and querying temporal databases, in which the visual representation is part of the model itself.

A *Temporal Graph Model Database (TGMDB)* is a triple \( (g, c, m) \), where \( g \) is a *Typed Graph*, \( c \) is a set of *Integrity Constraints* and \( m \) is an *Interpretation*. The database schema (intensional part), is represented in the TGM by the Typed Graph and the set of Constraints. The database instances (extensional part), are represented by the Interpretation. A database schema is expressed in the Typed Graph in terms of classes and relationships among classes (called roles).

In the TGM, classes and roles can also be modeled as *temporal* components of the schema, so allowing the user to retrieve data concerning not only the current state of a database, but also its past states. In particular, in this work we deal with *historical* databases, where each instance is associated with a set of disjoint intervals of an ordered and discrete time domain.

In short, according to the *temporal structure* defined in [28], the TGM shares with the majority of temporal data models the temporal structure, consisting of *discrete, determinate* and *anchored* (absolute time) temporal data. Moreover, TGM supports linear *temporal order* and *valid time* histories.

More formally, the *Typed Graph* is a 7-tuple \( g = (\mathcal{N}, \mathcal{E}, L_1, L_2, f_1, f_2, f_3) \), where:
Let $N = N_c \cup N_r$ be the set of nodes; $N_c$ is the set of so-called class-nodes, and $N_r$ is the set of the so-called role-nodes. Moreover, $N_c$ is partitioned into $N_{cp}$, the set of printable nodes, which represents the set of classes whose instances are domain values (e.g., integer, string); $N_{cu}$, the set of unprintable nodes, which represents the set of classes whose instances are object identifiers (e.g., Person); $N_{ct}$, which represents the set of temporal printable nodes; and $N_{ctu}$, which represents the set of temporal unprintable nodes. $N_r$ is partitioned into $N_{rn}$, the set of non-temporal roles, and $N_{rt}$, the set of temporal roles.

$E \subseteq N \times N$ is the set of edges;

$L_1$ is the set of node labels;

$L_2$ is the set of edge labels, including a special label $L$;

$f_1$ is a function from $N$ to $L_1$, associating a label to each node.

$f_2$ is a function from $E$ to $L_2$, associating a label to each edge.

$f_3$ is a function which characterizes the selection state of the elements of the Typed Graph, mapping each node to a value in \{unselected, selected, displayed\}.

The labels in $L_1$ are simply node names, whereas the edge labels in $L_2$ represent either set-oriented operations or boolean expressions, and are used in the process of query formulation, which will be described in the next section.

In order to formalize the notion of Interpretation, we introduce the following definitions:

We denote with $AD(n)$ the set of nodes adjacent to a given node $n$ (a node $n_i$ is adjacent to $n$ if there is an edge connecting them). If $n$ is a role-node, then $\forall n_i \in AD(n) \; n_i \in N_c$. We denote with $|AD(n)|$ the cardinality of $AD(n)$.  

\[ E \subseteq N \times N \]
We follow an approach to temporal databases that is based on a notion of discrete and ordered time. We denote with $T = \{t_1, t_2, \ldots\}$ such an ordered and discrete set of time points. Moreover, we use a special term called *now* which is assumed to be always bound to the current time point.

A time interval is defined as $<\text{begin}_k, \text{end}_k>$, with $\text{begin}_k, \text{end}_k \in T$ and, for every $t \in T$, if $\text{begin}_k \leq t \leq \text{end}_k$, then $t \in I_k$. We consider all time intervals as closed ones. So, an instant $t$ can be identified by the interval $(t, t)$.

A lifespan is a temporal element, that is, a finite and disjoint set of time intervals $\{I_1, \ldots, I_n\}$ [27]. We denote with $LS = \{I_1, \ldots, I_n\}$ the set of lifespans on $T$.

$O = O_p \cup O_u \cup O_{p_u} \cup O_{u_u}$ is the set of all atomic objects. $O_p$ are the printable objects, $O_u$ are the unprintable objects, $O_{p_u}$ are the temporal printable objects and $O_{u_u}$ are the temporal unprintable objects. Note that the components of $O$ are pairwise disjoint.

We define the universe $U$ as a set of structured objects, i.e., the smallest set containing $O$, and a set of labeled tuples (of any arity) $\langle l_1 : o_1, \ldots, l_k : o_k \rangle$, where $l_1, \ldots, l_k$ are elements of $L_1$, that is, labels of class-nodes, and $o_1, \ldots, o_k$ are elements of $O$.

$U_1 \subseteq U$ is the so-called temporal universe, that is, the subset of $U$ constituted by the temporal objects $O_{p_u}$ and $O_{u_u}$, and a set of labeled tuples (of any arity) $\langle l_1 : o_1, \ldots, l_k : o_k \rangle$, where $l_1, \ldots, l_k$ are labels of class-nodes, and $o_1, \ldots, o_k$ are elements of $O$.

\footnote{What we call time interval in our model corresponds to the period in the standard terminology described in [33].}
θ : U → LS is a total function that associates to each element x of U a lifespan, denoted as θ(x) = ls. For each t ∈ θ(x), we write t = \langle begin_t(x), end_t(x) \rangle, with x ∈ U.

Given a universe U, an interpretation for a Typed Graph g over U is a function mapping the printable (unprintable) class-nodes of g to subsets of printable (unprintable) objects of U, the temporal printable (unprintable) class-nodes to a subset of temporal printable (unprintable) objects of U, and the role-nodes to a subset of labeled tuples of U. In particular, given a role-node r, its Interpretation is constituted by a set of labeled tuples whose arity is equal to the number of class-nodes which are adjacent to r. Each tuple component is labeled with the label of one adjacent class-node, and takes its values in the corresponding Interpretation.

More formally, an Interpretation of a Typed Graph is a function m : N → 2(U ∪ U) ∪ 2(U × LS) mapping each node n ∈ N to a subset of U, as follows:

- If n ∈ Ncp, then m(n) ⊆ Ocp;
- If n ∈ Ncu, then m(n) ⊆ Ocu;
- If n ∈ Nct, then m(n) ⊆ \{ (o, θ(o)) | o ∈ Oct \};
- If n ∈ Ncf, then m(n) ⊆ \{ (o, θ(o)) | o ∈ Ocf \};
- If n ∈ Ncp and \{ n_1, n_2, \ldots, n_k \} = AD{ n }, then m(n) is a set of tuples of the form \langle f_1(n_1) : o_1, \ldots, f_1(n_k) : o_k \rangle, where f_1(n_1), \ldots, f_1(n_k) ∈ L_1 and \langle o_1, \ldots, o_k \rangle ∈ m(n_1) × \ldots × m(n_k);
- If n ∈ Ncq, and \{ n_1, n_2, \ldots, n_k \} = AD{ n }, then m(n) ⊆ \{ x, θ(x) \} where x is the tuple of the form \langle f_1(n_1) : o_1, \ldots, f_1(n_k) : o_k \rangle, with f_1(n_1), \ldots, f_1(n_k) ∈ L_1 and \langle o_1, \ldots, o_k \rangle ⊆ m(n_1) × \ldots × m(n_k) and θ(x) is the lifespan of x.

The set of Integrity Constraints c is specified by the designer using a suitable language (see [10]), in order to express relevant conditions on and meaningful properties of the classes and the roles in the Typed Graph.

Those constraints which are mostly relevant for the purpose of this paper are the following:

n_1 ISA n_2 ⇐⇒ m(n_1) ⊆ m(n_2) ∧ (n_1 ∈ Nc_p ∨ n_1 ∈ Nc_u) ⇒ (n_2 ∈ Nc_p ∨ n_2 ∈ Nc_u);
It means that \( n_1 \) is a subclass of \( n_2 \) in \( g \). Note that if \( n_1 \) is a temporal class, the superclass must also be temporal, in order to guarantee temporality of inherited attributes. Conversely, a temporal class may have non-temporal subclasses.

\[ \text{AT LEAST}(k, n_1, n_2), \text{ where } n_1 \in N_c, n_2 \in N_r, k \in \text{Nat} \text{ (the natural numbers), is satisfied by } \]
\[ m \text{ if for each } x \in n_1 \text{ the number of labeled tuples in } m(n_2) \text{ that contains } x \text{ in the } n_1\text{-component is greater than or equal to } k. \]

\[ \text{AT MOST}(k, n_1, n_2), \text{ where } n_1 \in N_c, n_2 \in N_r, k \in \text{Nat} \text{ (the natural numbers), is satisfied by } \]
\[ m \text{ if for each } x \in n_1 \text{ the number of labeled tuples in } m(n_2) \text{ that contains } x \text{ in the } n_1\text{-component is less than or equal to } k. \]

The \textit{ISA} construct corresponds to the abstraction of \textit{generalization} (subclass-class-relationships) as defined in semantic data models [31], whereas the \textit{AT LEAST} and \textit{AT MOST} constructs permit expressing cardinality constraints for roles (relationships). The \textit{AT LEAST} construct means that every object that is an instance of the class-node \( n_1 \) is linked to \textit{at least} \( k \) instances of the role-node \( n_2 \), whereas the \textit{AT MOST} construct means that every object that is an instance of the class-node \( n_1 \) is linked to \textit{at most} \( k \) instances of the role-node \( n_2 \).

In order to incorporate the inheritance in generalization hierarchies, we denote with \( AD'(n) \) the set defined as follows:

\[ \text{If } n \in N_c \text{ then } AD'(n) = AD(n) \cup \bigcup_{i} \{ \text{AD}(n_i) \} \text{ where } n_i \in N_{c_0} \text{ or } n_i \in N_{c_0}, \text{ and } n ISA^* n_i \text{ holds, where } ISA^* \text{ is the transitive closure of the } ISA \text{ relation (note that if } n \in N_{c_0} \text{ or } n \in N_{c_0}, \text{ then } AD'(n) = AD(n) \}. \]

\[ \text{If } n \in N_r \text{ then } AD'(n) = AD(n) \cup \{ p \in N_c | n \in AD(p) \}. \]

In other words, if \( n \) is a class-node, the set \( AD'(n) \) contains both the nodes adjacent to \( n \) and the nodes adjacent to its ancestors in the \textit{ISA} hierarchy; if \( n \) is a role-node, the set \( AD'(n) \) contains both the nodes adjacent to \( n \) and the descendants of such nodes.

We also consider the \textit{PARTOF} constraint, which is defined as follows:

\[ \{ n_1, \ldots, n_k \} \text{PARTOF}(n, r) \iff n_1, \ldots, n_k, n \in N_c \wedge r \in N_r \wedge \]
\[ (\forall o \in m(n) \exists ! x \in m(r) \text{ such that } x = \{ \langle f_i(n) : o_i, f_i(n_1) : o_1, \ldots, f_i(n_k) : o_k \rangle \} \wedge \]
\[ (\forall a, o_j \in m(n), \text{ with } i \neq j \text{ and } x_i \text{ as } x_{i} = \{ \langle f_i(n) : a, f_i(n_1) : o_1, \ldots, f_i(n_k) : o_k \rangle \wedge \]
\[ x_j = \{ \langle f_j(n) : a, f_j(n_1) : o_1, \ldots, f_j(n_k) : o_k \rangle \} \Rightarrow \prod_{f_i(n_1) = f_j(n_2), x_i \neq x_j} \prod_{f_i(n_1) = f_j(n_2), x_j \neq x_i}. \]

Note that the \( \prod \) operator is the equivalent of the \textit{projection} operator as defined in relational algebra [14].

The constraint described above models the hierarchy of \textit{aggregation} (component-aggregate relationship), expressed by using the \textit{PARTOF} construct, as usually defined in semantic data models [31], and its meaning is as follows: a) there is no aggregate object without its component objects; b) there are not two or more aggregate objects sharing the same component objects. It is worth noting that we use the \textit{PARTOF} constraint only in conjunction with temporal queries. Indeed, complex objects are not explicitly modeled in the Typed Graph (the choice of not modeling complex objects is mainly motivated by the fact that users are typically not familiar with record and set structures).

As for temporal role-nodes, we define the following constraint:

For each \( n \in N_r \), and \( m(n) \subseteq \{ x, \theta(x) \} \), where \( x \) is the set of tuples \( \{ f_i(n_1) : o_1, f_i(n_k) : o_k \} \), then the lifespan of \( n \) must be a subset of the intersection of the lifespans of the \( h \) temporal related objects (with \( h \leq k \), i.e: \( \theta(\{ f_i(n_1) : o_1, \ldots, f_i(n_k) : o_k \}) \subseteq \theta(o_1) \cap \ldots \cap \theta(o_h) \).
Formalizing Visual Interaction with Historical Databases

An example of using the TGM, in order to model the information concerning employment agencies is presented in Figure 14 and Table 2. Figure 14 shows the Typed Graph $g$ with the constraints. Note that the generalization association is graphically represented as an arrowhead edge. Table 2 contains a possible interpretation $m$ for $g$. For the sake of simplicity, the interpretation is listed only for a subset of the nodes.

![Typed Graph](https://via.placeholder.com/150)

**Fig. 14: A Typed Graph $g$ and a Set of Constraints $c$**

- $m(\text{Person}) = \{(o_1, o_2, o_3, o_4)\}$;
- $m(\text{Employee}) = \{(o_1, (1,10)), (o_2, (3,7)), (o_3, (2,now))\}$;
- $m(\text{City}) = \{\{o_5, o_6\}\}$;
- $m(\text{Integer}) = \{1, 2, \ldots, 8\}$;
- $m(\text{Name}) = \{\{\text{Person} : o_1, \text{String} : \text{Mary}\}, \{\text{Person} : o_2, \text{String} : \text{Peter}\}, \{\text{Person} : o_3, \text{String} : \text{John}\}$,
  \{\text{Person} : o_4, \text{String} : \text{Ann}\}\}$;
- $m(\text{C-name}) = \{\{\text{City} : o_5, \text{String} : \text{NY}\}, \{\text{City} : o_6, \text{String} : \text{Boston}\}\}$;
- $m(\text{Lives}) = \{\{\text{Person} : o_1, \text{City} : o_5\}, \{\text{Person} : o_2, \text{City} : o_5\}, \{\text{Person} : o_3, \text{City} : o_5\}, \{\text{Person} : o_4, \text{City} : o_5\}\}$;
- $m(\text{Emp-id}) = \{\{\text{Employee} : o_1, \text{Integer} : 00001\}, \{\text{Employee} : o_2, \text{Integer} : 00010\}, \{\text{Employee} : o_3, \text{Integer} : 00007\}\}$;
- $m(\text{Job}) = \{\{\text{Employee} : o_1, \text{Task} : T1, \text{Project} : p_1, (1, 5), (8, 10)\}, \{\text{Employee} : o_2, \text{Task} : T6, \text{Project} : p_2, (5, 10)\}, \{\text{Employee} : o_3, \text{Task} : T8, \text{Project} : p_3, (3, 6)\}, \{\text{Employee} : o_4, \text{Task} : T3, \text{Project} : p_1, (2, 10)\}, \{\text{Employee} : o_5, \text{Task} : T4, \text{Project} : p_2, (11, 30)\}\}$;
- $m(\text{Salary}) = \{\{\text{Employee} : o_1, \text{sal} - \text{value} : 5000, (1, 5)\}, \{\text{Employee} : o_2, \text{sal} - \text{value} : 6000, (6, 10)\}\}$;
- $m(\text{Level}) = \{\{\text{Employee} : o_1, \text{Integer} : 2, (1, 10)\}, \{\text{Employee} : o_2, \text{Integer} : 7, (3, 7)\}, \{\text{Employee} : o_3, \text{Integer} : 3, (2, 10)\}, \{\text{Employee} : o_4, \text{Integer} : 4, (11, now)\}\}$.

**Table 2: An Interpretation $m$ for $g$**

5. THE TEMPORAL GRAPHICAL PRIMITIVES

The basic idea of our work is to express any query-oriented user interaction with a database in terms of a simple set of fundamental Graphical Primitives - GPs. In [9] two primitives has been presented, namely, selection of a node and drawing of an edge, and it is demonstrated that all first order queries can be expressed by composing the two primitives, which have the same expressive power of well-known query languages.

However, temporal queries can not be expressed by composing only the two primitives, it is necessary to add further graphical primitives to deal with temporal information.

In this paper we extend the GP set with other graphical primitives, namely Temporal Graphical Primitives - TGP. They are called selection of node(s) shadow, temporal extension of change of edge label, selection of node label and change of node label.
Since both the GPs and TGP's consist of elementary graphical elements, they can be used as basic components for visually expressing temporal queries. The semantics of the GPs and TGP's is characterized in terms of database transformations, so that, starting from the initial database, the query results into a new database, containing exactly the requested information.

Let $D = (g, c, m)$ be a TGMDB. The application of either the GP selection of a node or the TGP selection of node(s) shadow on $D$ results in a new TGMDB $D' = (g', c', m')$. The application of either the GPs selection of a node or drawing of an edge or the application of the other TGP's on $D$ results in a new TGMDB $D'' = (g'', c'', m'')$.

The user’s clicking on a node corresponds to the GP selection of a node, through which the state of a node is switched from the value unselected, to either the value selected, or displayed, as shown in Figure 15, which presents the three different states for the class-node PROJECT.

![GP selection of a node](image)

**Fig. 15**: GP selection of a node

The GP drawing of an edge is a function $E(D', F, n, q)$, which corresponds to drawing a labeled edge between the nodes $n$ and $q$, and to either restricting the node interpretations according to the rules stated in the label, or performing a set operation on them. This primitive can only be applied when no edge between $n$ and $q$ is in $D'$. Its effect on $D'$ depends on the label $F$ and the nodes $n$ and $q$.

Let $n$ and $q$ be role-nodes, and let $F$ be a boolean expression, then, during the building of the result database $D'$ (see Section 6), $E(D', F, n, q)$ will give rise to a restriction of the final interpretation.

Let $n$ and $q$ be unprintable class-nodes, and let $F$ be the ‘$\equiv$’ symbol. In this case, $E(D', F, n, q)$ corresponds to the renaming of a tuple component in all the adjacents of node $q$ (it is useful for handling queries involving more than once the same node).

If $F$ is a set operator, then $n$ and $q$ have to be class-nodes. The resulting $D'$ will contain both a new node $s$ and new ISA constraints (e.g., if $F$ is union, then the constraints are $n$ ISA $s$ and $q$ ISA $s$).

The GPs constitute the minimal set of elementary interactions. However, for the sake of simplicity, we add a further operation, namely the change of edge label linking a class-node $s$ to a role-node $q$. The change of an edge label is denoted with $C(D', F_1, s, q)$, where $F_1$ is a propositional formula whose atoms are of the form $\alpha R \beta$, where $R$ is a comparison operator; $\alpha$ and $\beta$ are either the $s$ component of the tuples belonging to the Interpretation of $q$ (referred through the label of $q$) or constants. The presence of labels different from the true value $L$ will give rise to restrictions of the final interpretation.

The formalization of the GPs selection of a node and drawing of an edge can be seen in [9].

5.1. Selection of node(s) shadow

The user’s clicking on the shadow of temporal class(es) or role(s), corresponds to the TGP selection of node(s) shadow. The application of this TGP on a TGMDB $D$ results in a new TGMDB $D'$, on which the temporal information is visually explicit, that is, a class which represents the time intervals may be created and separately manipulated by other GPs (which are applied on $D'$). It is necessary to differentiate the selection of temporal nodes $n_1, \ldots, n_k$ from the selection of the shadows of $n_1, \ldots, n_k$, since the former means that the query does not involve temporal aspects of the node(s) whereas the latter does.

When this TGP is applied to a set of temporal (printable or unprintable) class-nodes $\{n_i\}$, with $1 \leq i \leq k$, the class-nodes are transformed into a set of non-temporal (printable or unprintable) class-nodes $\{n'_i\}$, containing all original instances without lifespans.
A new class-node \( s \) is created as an *aggregation* (see previous section) of the classes \( n_1', \ldots, n_k' \) with the class of time intervals \( i \). The new interpretation of the class-node \( i \) contains the *intersection* of the lifespans originally associated with the classes \( n_1, \ldots, n_k \) (note that the intersection can be empty).

A special label “all history” is included in the edge between the nodes \( r \) and \( i \), as shown in Figure 16 (a), corresponding to the query “Who were candidates and employees at the same period?”. Note that the state of the new role-node \( r \) is automatically switched to selected, in order to construct the result database.

The same procedure is used when the primitive is applied to a set of temporal role-nodes \( \{n_i\} \). The only difference is that the classes taking part to the aggregation are the classes \( AD(n_1) \cup \ldots \cup AD(n_k) \) plus the class-node \( i \), as shown in Figure 16 (b) (all figures in this section show only the subgraphs of interest), corresponding to the query “Retrieve the salary and levels of each employee in the last year”.

Note that the presented figures represent internal transformations of the TGMDB, and they are transparent to the end-user.

![Fig. 16: TGP Selection of node(s) shadow](image)

Table 3 shows the effects at *tuple level* [27] of the selection of the shadow of the temporal role-nodes \( \text{SALARY} \) and \( \text{LEVEL} \), so obtaining the non temporal role-node \( \text{SALARY/LEVEL-HISTORY} \).

<table>
<thead>
<tr>
<th>S</th>
<th>EMPLOYEE</th>
<th>SAL-VALUE</th>
<th>INTEGER</th>
<th>TIME-INTERVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J_1 )</td>
<td>( O_1 )</td>
<td>50000</td>
<td>2</td>
<td>( {1, 5} )</td>
</tr>
<tr>
<td>( J_2 )</td>
<td>( O_1 )</td>
<td>60000</td>
<td>2</td>
<td>( {6, 10} )</td>
</tr>
<tr>
<td>( J_3 )</td>
<td>( O_2 )</td>
<td>10000</td>
<td>7</td>
<td>( {3, 7} )</td>
</tr>
<tr>
<td>( J_4 )</td>
<td>( O_3 )</td>
<td>8000</td>
<td>3</td>
<td>( {2, 10} )</td>
</tr>
<tr>
<td>( J_5 )</td>
<td>( O_3 )</td>
<td>8000</td>
<td>4</td>
<td>( {11, \text{now}} )</td>
</tr>
</tbody>
</table>

Table 3: Interpretation of the role-node \( \text{SALARY/LEVEL-HISTORY} \)

More formally:

Let \( \{n_i\} \) be a set temporal unprintable class-nodes, \( \{n_i\} \subseteq D \), with \( 1 \leq i \leq k \), the **selection of the shadow** of \( \{n_i\} \) is a function: \( \varpi_n(D, \{n_i\}) = D' \) such that \( D' \) is the same as \( D \), except for the following conditions:

\[
\begin{align*}
N'_{c_u} &= N_{c_u} \cup \{n_i'\} \cup \{s\} \ (n_1', \ldots, n_k', s \text{ are new unprintable class-nodes}) \\
N'_{c_p} &= N_{c_p} \cup \{i\} \\
N'_r &= N_r \cup \{r\} \ (r \text{ is a new role-node}) \\
N'_e &= N_{e_u} \cap \{n_i\} \\
\text{l}_1 &= \text{l}_1 \cup \{\text{l}_r = \text{l}_1 \circ \ldots \circ \text{l}_k \circ \text{"history"} \} \text{ such that } \text{l}_1 \in f'_1(n_i) \} \\
\circ \text{represents concatenation among labels} \\
\text{l}_2 &= \text{l}_2 \cup \{\text{l}_e = \text{"all history"}\} \\
f'_1 &= f_1 \cup \{\text{l}_r, f'_1(i) = \text{"time-interval"}\} \\
\end{align*}
\]
The interpretation $m'$ of $D'$ is equal to $m$ except for $m'(s)$, $m'(n'_1)$, ..., $m'(n'_k)$, $m'(i)$ and $m'(r)$ ($m'(s)$ and $m'(r)$ are immediately derivable from $m'(n'_1)$, ..., $m'(n'_k)$, $m'(i)$ and the definition of $PARTOF$):

- $m'(s) = \{o_s\}$;
- $m'(n'_i) = \{o_{n'_i} | (o_{n_i}, \theta(o_{n'_i})) \in m(n_i)\}$;
- $m'(i) = \{I = \theta(o_1) \cap ... \cap \theta(o_k) \}$ such that $\exists o_i \in m(n_1), ..., o_k \in m(n_k)$.

A similar result is obtained whenever a subset of $\{n_i\}$ are temporal printable class-nodes. The only difference is that the nodes of this subset are new printable class-nodes.

Let $\{n_i\}$ be a set of temporal role-nodes, $\{n_i\} \subseteq D$, with $1 \leq i \leq k$, $AD(n_i) = \{n_{i_1}, ..., n_{i_k}\}$, with $ci = |AD(n_i)|$, the selection of the shadow of $\{n_i\}$ is a function $\pi_c(D, \{n_i\}) = D'$ such that $D'$ is the same as $D$, except for the following conditions:

- $\mathcal{N}'_{n_1} = N_{n_1} \cup \{s\} \text{ (s is a new unprintable class-node)}$;
- $\mathcal{N}'_{n_2} = N_{n_2} \cup \{i\} \text{ (i is a new printable class-node)}$;
- $\mathcal{N}'_{n_3} = N_{n_3} \cup \{r\} \text{ (r is a new role-node)}$;
- $\mathcal{N}'_{n_4} = N_{n_4} \setminus \{n_i\}$;
- $L_i = L_i \cup \{l_i = l_1 \circ ... \circ l_k \text{ "history" such that } l_i = f_i(n_i)\}$ ($\circ$ represents concatenation among labels);
- $f_I = f_i \cup \{f'_I(r) = l_r, f'_I(i) = "time-\text{interval}"\}$;
- $f^I = f_i \cup \{f'_I(i)\}$;
- $f'_I = f_i \cup \{f'_I(r) \}$;
- $\mathcal{C}' = \{c, AD(n_{i_1}), ..., AD(n_{i_k}), i\} \text{ PARTOF}(s, r)$;

The interpretation $m'$ of $D'$ is equal to $m$ except for $m'(s)$, $m'(i)$ ($m'(r)$ is immediately derivable from $m'(s)$, $m'(n_{i_1})$, ..., $m'(n_{i_k})$, $m'(i)$ and the definition of $PARTOF$):

- $m'(s) = \{o_s\}$;
- $m'(i) = \{I = \theta(x_1) \cap ... \cap \theta(x_k) \}$ such that $\exists x_i = \{f_i(n_{i_1}) : o_{i_1}, ..., f_i(n_{i_k}) : o_{i_k} \in m(n_i)\}$.

The queries described above involve several nodes $n_1, ..., n_k$ which share the same temporal information. In this case, the visual interaction is a sequence of user's clicking on nodes $n_1, ..., n_{k-1}$ plus a double clicking on node $n_k$.

On the other hand, there are queries in which the temporal information of each node $n_i$ is separately manipulated. In this case, the user double clicks on each node $n_i$, as shown in Figure 17, corresponding to the query "Retrieve the salary history and the level history of each employee". Note that two role-nodes resulting from that TGP are separately created.

Fig. 17: TGP Double clicking on nodes $SALARY$ and $LEVEL$.
5.2. Temporal extension of change of edge label

In order to visually express a temporal predicate, we add a further operation as temporal extension of change of edge label. It is denoted with $C_t(D', P, i, r)$, where the edge links the role-node $r$ and the class-node $i$, which results from the selection of node(s) shadow. $P$ is similar to the original formula $F_1 (F_1 = \alpha R \beta)$, with the difference that $P$ represents the temporal comparison operators among time intervals, defined in [8]. They are before($I$), meets($I$), during($I$), starts($I$), finishes($I$), overlaps($I$), and their respective symmetric counterparts after($I$), met-by($I$), during($I$), started-by($I$), finished-by($I$), overlapped-by($I$), and equal($I$), where $I$ is either an instant or a time interval, as shown in Figure 18, corresponding to the query “Retrieve all employees with salary greater than 5k during 1987-1988.”

As a consequence of this operation, the presence of labels different from the value “all history” will give rise to a restriction of the final interpretation.

More formally:

Let $r$ be a role-node resulting from the selection of node(s) shadow, $r \in D$, $AD(r) = \{n_i\} \cup \{i\}$ or $AD(r) = \{AD(n_1), \ldots, \cup AD(n_k)\} \cup \{i\}$, $AD(n_i) = \{n_{i_1}, \ldots, n_{i_m}\}$, with $ci = |AD(n_i)|$, and $P$ a temporal predicate, the temporal extension of change of edge label linking the role-node $r$ to the class-node $i$ is a function $C_t(D', P, r, i) = D''$ such that $D'$ is the same that $D'$, except for the following conditions:

$$m''(\langle r, i \rangle) = \{x \in m'(\langle r, i \rangle) \cup \{i\}$$

if $AD(r) = \{n_i\} \cup \{i\}$ then $m'' = m''_{n_i}$:

$$m''_{n_i}(r, i) = \{x = \langle f_1(s) : o_1, f_1(n_1') : o_1, \ldots, f_1(n_k') : o_k, f_1(i) : I' \rangle$$

$\exists x_j = \langle f_1(s) : o_1, f_1(n_1') : o_1, \ldots, f_1(n_k') : o_k, f_1(i) : I \rangle \in m'(r) \text{ such that }$

$$\prod_{f_1(n_{i_1}), \ldots, f_1(n_{i_m})} \{x_j\} \wedge I' = I \cap P;$$

if $AD(r) = \{AD(n_1), \ldots, \cup AD(n_k)\} \cup \{i\}$ then $m''(r) =

$$\{x = \langle f_1(s) : o_1, f_1(n_{i_1}) : o_1, \ldots, f_1(n_{i_m}) : o_k, f_1(i) : I' \rangle$$

$\exists x_j = \langle f_1(s) : o_1, f_1(n_{i_1}) : o_1, \ldots, f_1(n_{i_m}) : o_k, f_1(i) : I \rangle \in m'(r) \text{ such that }$

$$\prod_{f_1(n_{i_1}), \ldots, f_1(n_{i_m})} \{x_j\} \wedge I' = I \cap P;$$

$$m''(i) = \{I \mid I = \prod_{f_1(i) \in x_j} \{x \in m'(r) \}.$$

---

1We use Allen’s logics both for specifying temporal comparison operators and for inferring their relationships. Since the predicates of Allen are binary, the predicate before($I$), for example, actually means before($I(s), I$), indicating the objects whose lifespan satisfy the predicate.
It is worth noting that through the composition of the TGP selection of node shadow and the temporal extension of the operation change of edge label, we visually express a valid-time selection.

5.3. Change of node label

Regarding the role-node \( r \), whose related classes are either \( s, \{ n'_1 \} \) and \( i, \) or \( s, AD(n_1), \ldots, AD(n_k) \) and \( i (r \text{ comes from the selection of the shadow of temporal class- or role-node(s) respectively}), \) the user can select the first, second or last intervals from a lifespan, changing the label of the class-node \( i \) into a new label \( \mathcal{L} \), which represents an ordinal number of the form 1st-interval, 2nd-interval, nth-interval or last interval (as shown in Figure 19, corresponding to the query ‘When did the employees change their jobs for the first time?’).

Since the intervals of the lifespan of the objects are ordered, the tuples in \( r \) sharing the same values when projecting them on the classes \( n'_1, \ldots, n'_k \) or on the classes \( AD(n_1), \ldots, AD(n_k) \) are time ordered.

For example, considering the tuples of Table 4 (this table shows the effects of the selection of the shadow of the temporal role-node \( \text{JOB} \)), with the change of node label of Figure 19, only the tuple \( \langle J_2, O_1, T_1, P_1(8,10) \rangle \) satisfies the query above.

<table>
<thead>
<tr>
<th>S</th>
<th>EMPLOYEE</th>
<th>TASK</th>
<th>PROJECT</th>
<th>TIME-INTERVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>O1</td>
<td>T1</td>
<td>P1</td>
<td>(1,5)</td>
</tr>
<tr>
<td>J3</td>
<td>O1</td>
<td>T2</td>
<td>P2</td>
<td>(6,7)</td>
</tr>
<tr>
<td>J5</td>
<td>O2</td>
<td>T6</td>
<td>P3</td>
<td>(3,5)</td>
</tr>
<tr>
<td>J7</td>
<td>O2</td>
<td>T8</td>
<td>P3</td>
<td>(6,7)</td>
</tr>
<tr>
<td>J9</td>
<td>O3</td>
<td>T1</td>
<td>P1</td>
<td>(2,10)</td>
</tr>
<tr>
<td>J11</td>
<td>O3</td>
<td>T2</td>
<td>P2</td>
<td>(11,30)</td>
</tr>
<tr>
<td>J13</td>
<td>O3</td>
<td>T3</td>
<td>P1</td>
<td>(31,now)</td>
</tr>
</tbody>
</table>

Table 4: Interpretation of the role-node \( \text{JOB.HISTORY} \)

If the user is interested in retrieving the initial (final) instant of selected intervals, s/he may change the label to \( \text{begin} (\mathcal{L}) \) \( \text{(end} (\mathcal{L})) \) or simply \( \text{begin} \) \( \text{(end)} \), if the user does not specify an ordinal number. If the user is also interested in retrieving the duration of the selected intervals, s/he may change the label to duration \( \mathcal{L} \) or simply duration. The result is the set of durations \( D_1, \ldots, D_n \) of lifespans of the object with \( D_k = \text{end} I_k - \text{begin} I_k \), where \( I_k = \langle \text{begin} I_k, \text{end} I_k \rangle \) is an instance of the class-node \( i \). If the class-node \( i \) results from the application of the TGP selection of node label (defined in the following), the result is the set of durations \( D_1, \ldots, D_n \) with \( D_j = \sum_j (\text{end} I_j - \text{begin} I_j) \).

For example, the duration of the intervals associated with the tuple \( \langle O_1, T_1, P_1 \rangle \) in Table 4 is the value 6.
More formally:

Let \( r \) be a role-node resulting from the selection of node(s) shadow, \( r \in D' \); \( AD(r) = \{n'_1\} \cup \{i\} \) or \( AD(r) = \{AD(n_1) \cup \ldots \cup AD(n_k)\} \cup \{i\} \). \( AD(n_i) = \{n_{i1}, \ldots, n_{ik}\} \), with \( c_i = |AD(n_i)| \), and \( L \) be a parameter in \{nth-interval, last-interval\}, the change of label of a class-node \( i \) is a function \( C_i(D', L, i) = D'' \) such that \( D'' \) is the same as \( D' \), except for the following conditions:

\[
L'_i = L_i \cup \{l_i = L\};
\]

\[
f'_i = f_i \cup \{f'_i(i) = l_i\};
\]

\( m'' \) is equal to \( m' \) except for \( m''(r), m''(i) \):

if \( AD(r) = \{n'_1\} \cup \{i\} \)

\[m''(r) = \{x = \langle f_1(s) : o_1, f_1(n'_1) : o_1, \ldots, f_1(n'_k) : o_k, f_1(i) : l' \} | \exists \{x_j\} \in m'(r) \text{ such that } \prod_{f_1(n'_i) \ldots f_1(n'_k)}(x) = \prod_{f_1(n'_i) \ldots f_1(n'_k)}(x_j) \land l' = ord(l_j, L) \land l_j = \prod_{f_1(i)}(x_j)\};
\]

if \( AD(r) = \{AD(n_1) \cup \ldots \cup AD(n_k)\} \cup \{i\} \) then \( m''(r) = \{x = \langle f_1(s) : o_1, f_1(n_1) : o_1, \ldots, f_1(n_{k1}) : o_{k1}, \ldots, f_1(n_{k_{ck}}) : o_{k_{ck}}, f_1(i) : l' \} | \exists \{x_j\} \in m'(r) \text{ such that } \prod_{f_1(n_{1c}) \ldots f_1(n_{kc})}(x) = \prod_{f_1(n_{1c}) \ldots f_1(n_{kc})}(x_j) \land l = ord(l_j, L) \land l_j = \prod_{f_1(i)}(x_j)\};
\]

\( m''(i) = \{l | l = \prod_{f_1(i)}(x) \text{ with } x \in m''(r)\} \).

5.4. Selection of node label

In order to visually express temporal aggregations (e.g. the semester period of a student may represent an aggregate of course periods), we use the TGP selection of node label, which represents the visual specification of the coalesce operation [33].

The basic strategy is to exclude a class \( n_i \) from the set of classes \( n_1, \ldots, n_k \), which are related by the role-node \( r \). We can do this by selecting the role-node \( r \) and the class-node \( n_i \). As a consequence, the new interpretation of the class-node \( i \) contains the union of the lifespans originally associated with the tuples in \( r \), sharing the same values when projecting them on the classes \( n_1, \ldots, n_{i-1}, n_{i+1}, \ldots, n_k \).

For instance, we can derive the lifespan of any project by aggregating the lifespans of the associated tasks, that is, the user may select (project out) the label of the class-node TASK, as shown in Figure 20, corresponding to the query "For each employee, what was the average time to complete a project per department?", in order to gather the instances of the classes EMPLOYEE and PROJECT (e.g. \( \{O_2, P_3\} \) in Table 4).

![Fig. 20: TGP selection of node label](image_url)

More formally:
Let r be a role-node resulting from the selection of node(s) shadow, r \in D', AD(r) = \{n'_i\} \cup \{i\}
or AD(r) = \{AD(n_1)| |. . . | | AD(n_k)\} \cup \{i\}, AD(n_i) = \{n_{i_1}, . . . , n_{i_m}\}, with ci = |AD(n_i)|, the selection of label of both the role-node r and the class-node n_i is a function S(D', r, n_i) = D'
such that D' is the same as D', except for the following conditions:

if AD(r) = \{n'_i\} \cup \{i\} then m^r(r) =
\{x = \langle f_1(s) : a_0, f_i(n'_i) : a_1, . . . , f_i(n'_{i-1}) : a_{i-1}, f_i(n'_i) : a_{i+1}, . . . , f_i(n'_k) : a_k, f_i(i) : F'\rangle | \exists \{x_j\} \in m^r(r)\} such that
P' = \bigcup_j(I_j) \land I_j = \bigcup_{f_i(i)}(x_j)\};

if AD(r) = \{AD(n_1)| |. . . | | AD(n_k)\} \cup \{i\} then m^r(r) =
\{x = \langle f_1(s) : a_0, f_i(n_{i_1}) : a_1, . . . , f_i(n_{i_{i-1}}) : a_{i-1}, f_i(n_{i_k}) : a_{i+1}, . . . , f_i(n_{i_k}) : a_{i_k}, f_i(n_{i_{i_k}}) : a_{i_k}, f_i(i) : F'\rangle | \exists \{x_j\} \in m^r(r)\} such that
P' = \bigcup_j(I_j) \land I_j = \bigcup_{f_i(i)}(x_j)\};

Note that we visually express a valid-time projection, by using either the TGP change of node label, in order to display instants, time intervals and duration in the query result, or the TGP selection of node label, in order to display temporal aggregates in the query result.

5.5. Temporal Reference

We add a further operation as temporal extension of the GP drawing of an edge, in order to visually express queries with a temporal reference to another data. The temporal extension is denoted as a function \(S_D(D', P, n, q) = D''\), where n and q are role-nodes, and P represents one of the Allen temporal comparison operators. Considering the query “Which salaries did the employees earn when they changed their level for the first time?”, Figure 21 illustrates the temporal reference to the level history (see section 3 for the practical implementation of this primitive in the interface).

![Temporal reference to another data](image)

Fig. 21: Temporal reference to another data

Finally, we assume that several views of a database may be used during query formulation. In order to build such views, we recall the DUPPLICATE function introduced in [9]. The function DUPPLICATE\(^k\)(D), where D = (g, c, m) is a TGMDDB, results in a new TGMDDB\(D^k = (g^k, c^k, m^k)\) (the k-copy of D) which is equal to D except for the node labels, which represent a concatenation of k with the original labels. For example, in order to specify the query “Retrieve level history of all employees whose current level is greater than 5”, the DUPPLICATE function is applied to the Typed Graph so that a different role-node LEVEL is selected in each view, as shown in Figure 22

To conclude this section, Table 5 shows the graphical primitives and their corresponding visual mechanisms used in the TVQE interface (note that the abbreviation TE means temporal extension).
Formalizing Visual Interaction with Historical Databases

Fig. 22: Different views of a database

<table>
<thead>
<tr>
<th>Graphical Primitives</th>
<th>TVQE Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection of a node</td>
<td>Sucessive click on the index representing the node.</td>
</tr>
<tr>
<td>Drawing of an edge between the nodes n and q</td>
<td>Click on both the Where icon and the indices representing n and q.</td>
</tr>
<tr>
<td>Change of edge label between the nodes s and q</td>
<td>Click on both the Where icon and the index representing q.</td>
</tr>
<tr>
<td>Selection of node(s) shadow</td>
<td>Click on both the When icon and the indices (nodes).</td>
</tr>
<tr>
<td>TE of change of edge label between the node s and the role-node q</td>
<td>Click on both the When icon and the index representing q; Selection of either instant or period radio buttons; Specification of the temporal condition by using sliders.</td>
</tr>
<tr>
<td>Change of node label</td>
<td>Click on both the When icon and the index (node); Selection of the filter the retrieved time intervals check box; Selection of the corresponding interval time radio button.</td>
</tr>
<tr>
<td>Selection of node label</td>
<td>Click on the When icon and the index (temporal role-node); Selection of the temporal aggregation check box; Selection of the class(es) in the list to be collapsed.</td>
</tr>
<tr>
<td>TE of drawing of an edge between the nodes n and q</td>
<td>Click on both the When icon and the index representing n; Selection of both the temporal reference to... check box and the class representing q in the menu of temporal references; Selection of either instant or period radio buttons; Specification of the temporal condition by using sliders.</td>
</tr>
</tbody>
</table>

Table 5: TGPs and the corresponding actions in the TVQE interface

6. RESULT DATABASE

A new GMDB $D^s$ represents the result of the execution of a query expressed through the GMDB $D^r$, resulting from the application of GPs and TGPs (note that $D^s$ may be used as initial database (i.e. $D$) for further operations).

$D^s$ is a GMDB composed by a unique unprintable class node $q$ linked, by means of binary role-nodes $r_1, \ldots, r_k$, to a set of printable nodes, corresponding to the nodes set to displayed in $D^r$.

The interpretation of the above binary role-nodes is computed in two logical steps: in the first step all the selected role-nodes of $D^r$ are joined together giving the meaning of a fictitious n-ary role-node; in the second step such a meaning is suitably projected on the binary role-nodes of $D^s$, taking into account the restrictions specified in the labels of the edges drawn during the query formulation.

In order to specify $m^s$, which contains the result of the query, i.e., the interpretation of $q$, $r_1, \ldots, r_k$, some functions have to be introduced.

---

$[6]$ specifies the necessary and sufficient conditions on $D^r$, in order for $D^s$ to contain a non empty set of nodes. In this case, $D^r$ is said to be admissible. Roughly speaking, it must contain at least one displayed role-node and each displayed role-node must be linked to at least one displayed printable class-node and one unprintable (selected or displayed) class-node.
For a better understanding of the functions that are defined in the following, let us consider the query: "Give the name and emp-id of all employees living in NYC and who had levels with duration greater than the duration of their salaries. Give also these levels, salaries and their corresponding periods".

Expressing this query corresponds to applying on the database Typed Graph the following GPs and TGP, as shown in Figure 23: selection of nodes EMPLOYEE, LIVES, CITY, C-NAME; display of nodes NAME, STRING, EMP-ID, INTEGER, LEVEL, SALARY, SALARY-VALUE; and selection of shadow of the nodes LEVEL and SALARY.

![Figure 23: Selection of nodes and shadowed nodes](image)

Figure 24 shows the effects of the selection of nodes and shadows on the Typed Graph in Figure 23.

![Figure 24: Typed Graph of interest](image)

Let $HSN(x)$ be a function defined on the class-node $x$ and returning a true value if $x$ is the highest selected node in the ISA hierarchy it belongs to (i.e., is $x$ the highest selected node?): $HSN(x) = True$ iff $f_3(x) \in \{selected, displayed\}$ and there exists no $w \in N_e$ such that $xISA^*w$ and $w \in \{selected, displayed\}$. Considering the generalization hierarchy in the Typed Graph of Figure 23, then, $HSN(Employee) = True$. 

Let $TAD(y)$ (i.e., the set of true adjacents of $y$) be a function defined on a role-node $y$ and returning the subset of $AD(y)$ satisfying the $HSN$ condition: $TAD(y) = \{x \mid x \in AD(y) \text{ and } HSN(x)\}$. For example, considering the Typed Graph in Figure 23, then, $TAD(\text{Name}) = \{\text{Employee, String}\}$.

Let $RM(m(y))$ (i.e., restrict the meaning of $y$) be a function defined on the Interpretation of a role-node $y$. When $y$ is a non-temporal role node, $RM(m(y))$ results into an Interpretation of $y$, restricted according to the selected class-nodes in the ISA hierarchies which are in $TAD(y)$.

The new interpretation of the role-node $NAME$, for example, will be the set of tuples $\{(\text{Employee : } o_1, \text{String : Mary}), (\text{Employee : } o_2, \text{String : Peter}), (\text{Employee : } o_3, \text{String : John})\}$. Note that this function suitably changes the labels of the tuples in the Interpretation of $y$ as well.

When $y$ is a temporal role node, the result of $RM(m(y))$ is a set of tuples, where the lifespan of each tuple contains at least the current time interval. It means that if a temporal role-node $y$ exists in $D'$ (its shadow has not been selected), we consider only its current state in the interpretation of $D'$.

Hence, the function $RM(m(y))$ is defined as follows:

if $y \in N_{r_{\tau}}$ then $RM(m(y)) = \{\{l_1: o_1, \ldots, l_k: o_k\} \mid \{l_1: o_1, \ldots, l_k: o_k\} \in m(y)\}$

if $y \notin N_{r_{\tau}}$ then $RM(m(y)) = \{\{l_1: o_1, \ldots, l_k: o_k\} \mid \{x, \theta(x)\} \in m(y), \text{ where } x \text{ is the tuple of the form } \langle l_1: o_1, \ldots, l_k: o_k \rangle, \text{ and } 3l \in \theta(x) \text{ such that } I = \langle \text{begin}(x), \text{now} \rangle, \text{ and } l_1: o_1, \ldots, l_k: o_k \rangle \in m(f^{-1}(l_1)).$

In our example, there is not any temporal role-node $y$ in $D'$, as shown in Figure 24.

Let $RM'(m(y))$ be a function defined on the Interpretation of a role-node $y$, which, in order to compute the Interpretation of $D^s$, concatenates the label of $y$ to the labels of the tuples in the Interpretation of $y$:

$RM'(m(y)) = \{\{l_1: o_1, \ldots, l_k: o_k\} \mid \{l_1: o_1, \ldots, l_k: o_k\} \in RM(m(y)) \text{ and } l_1: o_1 = f_1(y) \circ l_1\}.$

For example, $RM'(m(\text{Name})) = \{\{\text{Name.Employee : } o_1, \text{Name.String : Mary}\}, \{\text{Name.Employee : } o_2, \text{Name.String : Peter}\}, \{\text{Name.Employee : } o_3, \text{Name.String : John}\}\}$.

We define the interpretation of $D^s$ as follows:

$m^s(q) = \{t_1, \ldots, t_s\},$

where each $t_i$ is a new invented unprintable value, and $s$ is the cardinality of a set $\text{RIS}$ that can be interpreted as the extensional part of the user query, and is defined as follows:

if $N_{r_{\tau}} = \emptyset$ then $\text{RIS} = \emptyset$;

if $\left| N_{r_{\tau}} \right| = 1 \land \{\{k \in N_{r_{\tau}} \mid f_2(k) \in \{\text{selected, displayed}\}\} = 1 \right.$ then $\text{RIS} = RM'(m(k));$

Otherwise, $N_{r_{\tau}} = \{r_1, \ldots, r_k\}$, with $k \geq 1$, and $\text{RIS} = \text{inst}(\text{eval}(n_1, \text{eval}(n_2, \ldots, \text{eval}(n_{k-1}, n_k)))$),

where $\{n_1, \ldots, n_k\} \equiv \{m \in N_{r_{\tau}} \mid f_2(m) \in \{\text{selected, displayed}\}\}$ and the function $\text{inst}$ extracts the set of instances of a fictitious node computed by $\text{eval}$, where $\text{eval}(n_1, n_2)$ returns a fictitious role-node $n$, whose adjacents are the union of the adjacents of $n_1$ and $n_2$, and whose Interpretation is a set of tuples as follows:

$\{\{l_1: o_1, \ldots, l_k: o_k, \text{"n"} \circ l_{k+1}: o_{k+1}, \ldots, \text{"n"} \circ l_n: o_n, l_{k+1}: o_{k+1}, \ldots, l_n: o_n\} \mid \text{such that } i = k + 1 \ldots, h, f^{-1}(l_i) \in N_{r_{\tau}} \text{, and } \{l_1: o_1, \ldots, l_k: o_k, f_1(n_1) \circ l_{k+1}: o_{k+1}, \ldots, f_1(n_2) \circ l_n: o_n\} \in \text{RM}'(m(n_1)) \text{, and } \{f_1(n_2) \circ l_{k+1}: o_{k+1}, \ldots, f_1(n_2) \circ l_n: o_n, l_{k+1}: o_{k+1}, \ldots, l_n: o_n\} \in \text{RM}'(m(n_2))\}$.

Note that, if the nodes $n_1$ and $n_2$ do not share tuple components, the function $\text{eval}$ returns the cartesian product of the interpretations of $n_1$ and $n_2$. 


Hence, the set \( RIS \) is defined as follows: \( RIS = \text{inst}(\text{eval}(\text{SALARY-HISTORY}, \text{eval}(\text{LEVEL-HISTORY}, \text{eval}(\text{EMP-ID}, \text{eval}(\text{NAME}, \text{eval}(\text{C-NAME}, \text{LIVES})))))) \).

We show the result of the \( \text{eval}(\text{LEVEL} - \text{HISTORY}, n_2) \) in Table 6 and \( \text{eval}(\text{SALARY} - \text{HISTORY}, n_3) \) in Table 7 (the column titles of this table were abbreviated due to space limitations). Note that \( n_2 \) and \( n_3 \) are the fictitious nodes generated by \( \text{eval}(\text{EMP-ID}, \text{eval}(\text{NAME}, \text{eval}(\text{C-NAME}, \text{LIVES})))) \) and \( \text{eval}(\text{LEVEL} - \text{HISTORY}, n_2) \) respectively.

<table>
<thead>
<tr>
<th>city</th>
<th>c-name.string</th>
<th>empid</th>
<th>emp_name</th>
<th>level.integer</th>
<th>level.interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>NYC</td>
<td>O_1</td>
<td>Mary</td>
<td>0001</td>
<td>2</td>
<td>(1, 10)</td>
</tr>
<tr>
<td>Boston</td>
<td>O_2</td>
<td>Peter</td>
<td>0010</td>
<td>7</td>
<td>(3, 7)</td>
</tr>
<tr>
<td>Boston</td>
<td>O_3</td>
<td>John</td>
<td>0007</td>
<td>3</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>Boston</td>
<td>O_4</td>
<td>John</td>
<td>0007</td>
<td>4</td>
<td>(11, now)</td>
</tr>
</tbody>
</table>

Table 6: \( \text{eval}(\text{LEVEL} - \text{HISTORY}, n_2) \)

<table>
<thead>
<tr>
<th>city</th>
<th>c-name.string</th>
<th>empid</th>
<th>emp_name</th>
<th>level-h.intg</th>
<th>level-h.intv</th>
<th>sal-h.av</th>
<th>sal-h.intv</th>
</tr>
</thead>
<tbody>
<tr>
<td>NYC</td>
<td>O_1</td>
<td>Mary</td>
<td>0001</td>
<td>2</td>
<td>(1, 10)</td>
<td>5000</td>
<td>(3, 7)</td>
</tr>
<tr>
<td>NYC</td>
<td>O_1</td>
<td>Mary</td>
<td>0001</td>
<td>2</td>
<td>(1, 10)</td>
<td>5000</td>
<td>(3, 7)</td>
</tr>
<tr>
<td>NYC</td>
<td>O_1</td>
<td>Mary</td>
<td>0001</td>
<td>2</td>
<td>(1, 10)</td>
<td>6000</td>
<td>(3, 7)</td>
</tr>
<tr>
<td>Boston</td>
<td>O_2</td>
<td>Peter</td>
<td>0010</td>
<td>7</td>
<td>(3, 7)</td>
<td>10000</td>
<td>(3, 7)</td>
</tr>
<tr>
<td>Boston</td>
<td>O_3</td>
<td>John</td>
<td>0007</td>
<td>3</td>
<td>(2, 10)</td>
<td>8000</td>
<td>(2, now)</td>
</tr>
<tr>
<td>Boston</td>
<td>O_4</td>
<td>John</td>
<td>0007</td>
<td>4</td>
<td>(11, now)</td>
<td>8000</td>
<td>(2, now)</td>
</tr>
</tbody>
</table>

Table 7: \( \text{eval}(\text{SALARY} - \text{HISTORY}, n_3) \)

Let us denote with \( RIS' \) the set of tuples of the form \( \langle h_1 : a_1, \ldots, h_y : a_y, f_1(q) : a_{y+1} \rangle \) such that \( a_{y+1} \in m^t(q), \langle f_1 : a_1, \ldots, f_n : a_n \rangle \in RIS \) and satisfies all boolean expressions \( \mathcal{F}_1, \ldots, \mathcal{F}_n \) and temporal predicates \( P_1, \ldots, P_m \), such that different tuples have different values in the \( y + 1 - th \) component. Returning to the example, the user perform another operations, as shown in Figure 25 (for the sake of simplicity, we show only the selected nodes).

![Fig. 25: Application of other GPs and TGP](image)

The only tuples which satisfy the conditions specified in the query are the first, second and third in Table 7.
Eventually, we obtain from $RIS'$ the interpretation of the role-nodes of $D^p$:

$$m(r_i) = \{ \langle l : o, f_1(q) : o_{g+1} \rangle | l : o_1, \ldots, f_1(r_i) \circ \ldots \circ l : o, \ldots, l_g : o_g, f_1(q) : o_{g+1} \rangle \in RIS' \}$$

The system evaluates the result database and displays the corresponding GMDB $D^p$, as shown in Figure 26, with the corresponding interpretation in Table 8. On the interface, the selection of either the Data Vis or the Time Vis icons forces the construction of the GMDB $D^p$.

![Result Typed Graph](image)

**Fig. 26: The Result Typed Graph**

$$m(RESULT) = \{o_{20}\};$$

$$m(String) = \{a, \ldots, 1, \ldots, 0\}^*;$$

$$m(Integer) = \{1000 \ldots 10000\};$$

$$m(Name) = \{Employee : o_{28}, String : Mary\};$$

$$m(Emp-id) = \{(Employee : o_{28}, Integer : 0001)\};$$

$$m(Job) = \{(Employee : o_{28}, Salary : 5000, \{1, 5\}, \{8, 10\})\},$$

$$m(Level) = \{(Employee : o_{28}, Level : 2, \{1, 10\})\}. $$

**Table 8: The Result Interpretation**

### 7. IMPLEMENTATION ISSUES

For portability, flexibility and efficiency issues, the TVQE prototype has been implemented in the JAVA object-oriented language, using the JDBC (Java Database Connectivity) API as interface to the DBMS. The overall system is actually under implementation, specifically we are currently implementing the Data Visualization Window, based on the dynamic query approach, which supports the display of the lifespan of a class or relationship at different granularities.

In this section, we describe the implementation issues of TVQE, considering the system architecture and the class diagrams in UML (Unified Modeling Language) [2].

#### 7.1. System Architecture

The system exhibits a client-server architecture, with a direct connection with the database server, as shown in Figure 27.

A TGM schema is created using a suitable tool, i.e. conceptual project tool (this module is not included into the TVQE architecture). The TGM schema is the input of the Mapping TGM-Relational module, which creates a DDL (Data Definition Language) for a specific DBMS. This module also creates a structure that associates the graphical schema and its corresponding relational structure (TGM-Relation structure). This structure is copied into the client module.
The user accesses the database through the *TVQE interface*. The system generates a subgraph of interest which will be converted into a query in *SQL* through the *translator* module (note that this module needs the *TGM-relational structure*). Through *JDBC*, the DBMS provides the user with the query result.

7.2. Class Diagrams of the TVQE environment

In order to detail the structure of TVQE we use the well-known UML Class diagrams [2]. We recall that in such diagrams *classes* are represented as rectangles. *Composition*, *aggregation* and *generalization* are represented as an edge with a filled diamond, an edge with a hollow diamond and an edge with a white arrow respectively. The filled diamond attached to a class indicates that it is a composite class. It means that if such class is removed, so are its component classes, otherwise, the diamond may be hollow to indicate an aggregation relationship.

Furthermore, the arrowhead from the class $A$ to the class $B$ indicates that the class $B$ is initialized from the class $A$.

The TVQE system comprises more than 90 classes. Figure 28 illustrates the main class of TVQE environment, i.e., the *Agenda* (notebook) class. In Figure 28 such a class is represented at “high level”, i.e., without attributes and methods. The Agenda class builds the main window with the panels *schema* and *interaction* windows. It also builds the internal structure of the conceptual schema which is selected by the user. Basically, the *Agenda* class is composed by the classes which visually represent the conceptual schema as a context tree, a graph, a subgraph and as a graphical “notebook”.

All classes of the system are composed by event classes (classes whose labels finish with the word *Listener*) associated with the widgets which represent the functionalities of the system. Starting from the event classes, other classes are initialized, for example, the *saveSubgraph* class, which generates the subgraph or the *nodeColor* class, which is responsible for the colors visualization of nodes and indices. The *filterAttribute* and *dialogWhen* classes construct the *Where* and *When* panels, respectively.

More details about the class diagrams of TVQE can be seen in [18].
7.3. Difficulties and Activities in Development

Generally speaking, one of the hard problems in database interface development is to achieve both high expressive power and ease-of-use. This also happened in the TVQE case.

Other more specific difficulties have been also encountered, such as:

- To find a visual property (color, shape, size, position) to effectively characterize visual components which compose a conceptual schema (classes, relationships, attributes, and hierarchical relationships);
- To explicitly model a class as aggregate of other classes in complex schemas;
- To include cardinality constraints in the graphical schema or notebook, without using notations like [1, *], which are designer-oriented, and to visually differentiate a monovalued (multivalued) attribute from a temporal monovalued (multivalued) attribute;
- To design either a single panel (When panel), for specifying a temporal condition or two panels (Instant and Period panels), for specifying separately an instantaneous and periodic conditions.

There are some ongoing activities in order to enhance the TVQE interface, such as:

- Inclusion of a brief description of each context or class on the notebook sheet;
- Inclusion of representative images on indices that represent attributes;
- Support of the duplication of indices in queries which involve cycles or references to the same attribute;
- Inclusion of the *equal* predicate within the panel which contain the Allen predicates [3];
- Inclusion of the *Undo* mechanism;
- Exporting of the TVQE environment on the Web.

Moreover, we are implementing the mapping of the subgraph of interest into a SQL query, as discussed in Section 7.1.
8. RELATED WORK

Similar proposals of non-temporal and temporal visual interfaces can be seen in [36], [7], [6], [35], [30] and [38].

8.1. Non-temporal Visual Interfaces

The visualization of a complex schema as a context tree is similar to the pre-tree structure described in [36], where an environment for the visualization of and navigation on documents in hypertext systems is presented. A pre-tree is an intermediate between a graph and a tree. It has a root, but differently from a real tree, all its descendants can be arbitrary graphs. Hence, the tree leaves (branches) compose a list of graphs and nodes of different branches can not be related by edges.

Considering the hierarchical representation of information, a pre-tree is similar to a context tree. Both present a root node, and each graph in a pre-tree may correspond to a context in the context tree. However, considering the hierarchical visualization of information, the two approaches are different. While leaves in a pre-tree are visually represented by square nodes, where each one contains a graph, such leaves in the context tree represent the classes which compose a context. The detailed graph of a context is visualized in the graphical schema. Hence, the pre-tree visualizes different hierarchical perspectives of a system in a simple structure, while the context tree and graphical schema are more adequate for the Location and Manipulation phases of the query.

The use of a graphical notebook in interfaces has arisen in other application domains such as hypertext systems [39], [40], [25], and interfaces which manage personal information in operational systems [7], [17].

TabWorks\textsuperscript{sm} [7] has been proposed as an enhancement of the desktop metaphor used in the Windows operating system, by considering the organization of documents and application programs. In TabWorks\textsuperscript{sm}, the indices represent directories. When an index is selected, the files of the directory represented by that index appear in a notebook page. More specifically, the notebook comprises a set of indices, each one containing one or more pages. Pages contain icons that represent documents or application programs. The user can create, remove, rename and organize indices, pages and icons. Hence, the metaphor visualizes the container hierarchy used in the file manager of Windows.

By adapting the approach of TabWorks\textsuperscript{sm} to the database context, our notebook metaphor could present the structure shown in Figure 29. Note that distinct visual components may correspond to the same concept (ex. indices and icons may correspond to the same class). The two approaches explore the same visual metaphor in distinct application domains, but the information navigation on the two approaches is different (while indices are fixed in TabWorks\textsuperscript{sm}, in TVQE the indices are dynamically modifiable).

The target class concept in TVQE is related to the notion of primary concept associated to a point of view, described in [6]. This work describes a prototype called VisTool for querying relational databases, which is based on the visual query language Visionary.

Visionary[5] adopts a hybrid visual representation (diagrammatic and iconic). Primary concepts are visually represented as icons, while the association among concepts are visually represented as directed edges. Ellipses and triangles visually represent the is-a and part-of relationships, respectively. The primary concept is connected to other concept through a path of associations. So, a query is interpreted by a point of view without explicitly formulating relational joins. Moreover, it has a mechanism which solve conflicts among different points of view.

8.2. Temporal Visual Interfaces

Notable proposals of VQSs for temporal databases are [35], ERT/vql [45], TVQL [30] and Lifelines [38].

\textsuperscript{1}TabWorks\textsuperscript{sm} is a trademark of the Xerox Corporation.
The visual query language of Kouramajan and Gertz [35] comprises some visual constructors that allow the formulation of temporal queries, based on a temporal extended E-R model [16]. The visual constructors are: temporal boolean expression, which is a conditional expression over attributes and relationships of an entity; true time of a boolean expression \( c \), which determines the value of a temporal element of each entity \( c \); temporal selection, which is used for selecting particular entities based on temporal conditions; and temporal projection, which is used to filter the displayed data of selected entities in specific periods.

The approach of Kouramajan and Gertz is different from our approach in the following aspects: it does not explicitly visualizes temporal entities and relationships.

The temporal projection constructor is applied over a completed E-R diagram (after the selection of entities of interest). The problem is that only a temporal projection is applied on a query. For example, the following query can not be specified: "Retrieve the name and salary at 1990, and rank during 1985-1988, of each current computer science instructor".

The temporal selection constructor is used for retrieving entities based on the temporal conditions over attributes. Such conditions involve the comparison of two temporal elements, by using the comparison operators \( =, \neq, \subseteq \) and \( \supseteq \). As a consequence, not all Allen operators have corresponding ones. The expressivity of filtering time intervals in [35] is superior to the one of our approach.

Queries which involve a temporal reference to another data are specified in a similar way in the two approaches. However, in [35] the database schema is represented as an E-R diagram, and such a model-based representation may be cumbersome to non-expert users.

A similar approach is followed by the visual language for the ERT model, \texttt{ERT/vql} [45]. It contains an interesting feature: a result visualization phase where advanced interactive visualization techniques are applied over the query result. However, during the query construction phase, the user needs to adopt a textual syntax and a model-based visual representation of the schema.

\texttt{TVQL} [30] and \textit{Lifelines} [38] are timeline-based systems. \texttt{TVQL} is mainly used for video data, more specifically, for identifying temporal trends in video data. \textit{Lifelines} is a visualization environment used in applications which involve a survey of personal histories (biographical data). It adopts the dynamic query approach, but has the limitation of displaying a small data set at a time.
9. CONCLUSION

This paper presents a proposal to put in an easy-to-use visual form the difficult task of formulating queries on historical databases. The different kinds of temporal queries are shown in a homogeneous way, inside a global visual environment. The approach of integrating the notebook metaphor with the visual access to the database facilitates the user in expressing the query, since s/he has not to learn neither the underlying data model, nor the syntax and semantics of a textual query language. Moreover, the user can incrementally formulate the query, also receiving immediate graphical feedback.

It is worth noting that the aim of this paper is not in defining a new temporal data model and a corresponding temporal query language, since the literature is overabundant of them. The basic idea is in giving relevance to the visual representation, which is one of the most important components in building effective interfaces for databases, including the historical ones.

The main contribution of this work is the formal definition and practical realization of a completely visual and easy-to-use environment for querying temporal databases.

The specific contributions of this work are:

- The use of a notebook metaphor to stimulate the user’s attention towards the system. Such a metaphor recall a known concept of the “real world” and the user performs a “conceptual navigation” instead of a “structural navigation” on the schema.
- The introduction of the context tree concept to allow the user to easily select a subpart of the overall database.
- The availability, in the query phase, of different data perspectives, which derive from the choice of different target classes.
- The availability of a domain editor to remove any ambiguity which may occur when specifying the domain of an attribute.
- Reusability and portability issues, since TVQE can be put on the top of any relational DBMS that supports ODBC (Open Database Connectivity).
- The application domain independence of TVQE.
- The definition of a complete mapping between the formal primitives and their corresponding visual mechanisms.

Limitations of the system are the absence of update facilities and the lack of a formal evaluation of the TGP’s expressive power.

Future work will aim at overcoming such limitations and extending TVQE with new features, such as:

- Application of visual interaction mechanisms to express complex queries;
- Enhancement of the domain editor, in order to support other attribute types, including the temporal ones;
- Extension to other visual representations (e.g., form-based) and accessing mechanisms;
- Extension of the temporal data model with features such as periodic times (for instance ‘each Monday’). In this case, the time of an object is given implicitly by a function and is no longer stored explicitly in the database;
- Storing of data about the history of interaction, to access and dynamically maintain them with the same mechanisms as the application data.
This last extension represents an effort towards clearly establishing a link between user interaction and modeling and data modeling and querying, also allowing an empirical study and evaluation of TVQE in terms of usability (preliminary results can be seen in [19], [21]).

We also plan to increase the expressive power of the TGM model and its graphical primitives, in particular by:

- Including a formal treatment of time granularities;
- Comparing the TGPVs with a temporal algebra [44];
- Allowing the specification of queries which involve negation;
- Allowing the explicit specification of quantifiers.

REFERENCES


