J Intell Inf Syst (2012) 38:601–644
DOI 10.1007/s10844-011-0170-7

Modeling temporal dimensions of semistructured data

Carlo Combi · Barbara Oliboni · Elisa Quintarelli

Received: 24 June 2010 / Revised: 11 May 2011 / Accepted: 29 June 2011 /
Published online: 21 July 2011

Abstract In this paper we propose a graph-based generic model able to uniformly represent semistructured data and their temporal aspects. In particular, we start from a generic and expressive model proposed in the database literature and consider in a formal and systematic way both valid time and transaction time, together with the set of temporal constraints needed to correctly manage the semantics of the represented time dimension. We then propose operations, which allow the incremental management of the proposed model satisfying the introduced temporal constraints. Moreover, we also deal with the possibility of managing together the two classical time dimensions of valid and transaction times, and formalize the set of constraints needed to correctly handle these temporal aspects together. Some examples taken from a medical scenario will be used to describe the introduced concepts.

Keywords Temporal databases · Semistructured data · Valid time · Transaction time · Data modeling · Temporal constraints · Clinical databases

C. Combi (✉) · B. Oliboni
Dipartimento di Informatica, Università degli Studi di Verona, Ca' Vignal 2,
Strada le Grazie 15, 37134 Verona, Italy
e-mail: carlo.combi@univr.it

B. Oliboni
e-mail: barbara.oliboni@univr.it

E. Quintarelli
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Via Ponzio 34,
20133 Milano, Italy
e-mail: quintare@elet.polimi.it

1 Introduction

Often data are stored according to different formats, with different structures, or even with missing or implicit structures. It is the case, for example, of the different forms filled by physicians and nurses during their daily clinical routine for patient lab results, patient visits, patient history, and so on: the structure of the form is recorded with data. A further example from the medical domain is that of the discharge report, which is often written in natural language style, even though some common basic structures could be retrieved. Moreover, slightly different formats and different record structures could be used in different domains and organizations: it arises the need of integrating these heterogeneous data to be able to access them in a unified way. To this regard, in the past years the database research community has concentrated on the introduction of methods for representing and querying *semi-structured data*, i.e., neither raw data nor strictly typed data with an absolute schema fixed in advance (Abiteboul 1997). In several domains semistructured data are used to represent information having some temporal dimension (Dyreson and Grandi 2009). For example, in the medical domain semistructured data could represent the evolution of patient health status through time; on the other side, semistructured data should allow one to be aware of the clinical information known by a physician in different past situations.

A lot of work has been done in order to represent semistructured data by using data models based on directed labeled graphs without considering any temporal dimension (see, for example, OEM (Papakonstantinou et al. 1995), UnQL (Buneman et al. 1996), STRUDEL (Fernandez et al. 1997), GraphLog (Consens and Mendelzon 1990), G-Log (Paredaens et al. 1995) and XML-GL (Ceri et al. 1999)). All these proposals organize data in graphs where nodes denote either objects or values, and edges represent relationships between them. As for the classical database field, also in the semistructured data context it has been considered the possibility of extending previously proposed models in order to take into account the evolution of data through time (see for example Amagasa et al. 2000; Chawathe et al. 1999; Oliboni et al. 2001). Again, the development of methods for representing and querying *changes in semistructured data* have been often founded on graph-based data models (see, for example Amagasa et al. 2000; Chawathe et al. 1998, 1999; Dyreson et al. 1999; Oliboni et al. 2001; Rizzolo and Vaisman 2008). Even though the well-known concepts of valid and transaction times (Jensen et al. 1998) have been considered as a starting point, these proposals focus on different aspects and propose different strategies to deal with temporal dimensions of information.

To complete the sketched scenario, it is worth mentioning that semistructured data are often used and accessed through web sites: to this regard, it has been recognized and emphasized that time is an important aspect to consider in designing and modeling (data-intensive) web-sites (Atzeni 2002; Rizzolo and Vaisman 2008).

We feel that the temporal aspect of data is not an outdated topic, and semi-structured temporal data models could became a reference for the logical design of (data-intensive) web sites, when considering information that require an effective management of time-varying perspectives (as a matter of fact, a widely accepted model for the logical design of data-intensive web sites, such as the relational data model in the classical database field, has not yet been proposed). In particular, a very

interesting issue to tackle is the complete formalization of constraints related to the considered temporal dimensions. This aspect is important both for semistructured temporal data models, and for the temporal relational models (Jensen and Snodgrass 1999) considered in the classical temporal database area.

In this work we start from the proposal described in Dyreson et al. (1999), which, in our opinion represents an expressive and generic model suitable to represent various perspectives of semistructured data (not only of XML documents): in Dyreson et al. (1999), the authors focus on the definition of (i) a general semistructured graph-based data model and of (ii) suitable operators for managing queries on such graphs. By adopting and extending the approach proposed in Dyreson et al. (1999), we propose a graph-based generic model able to uniformly represent semistructured data and their temporal aspects: we thoroughly analyze the definition, formalization, and management of constraints specifying the semantics of either valid or transaction time temporal dimensions. More precisely, in the semistructured data context, our main contributions are:

- representation of valid time, transaction time, or both the two dimensions;
- specification of the main constraints needed to correctly manage the semantics of the considered time dimension;
- definition of suitable operations for a correct handling of each of these two temporal aspects.

Even if the data model proposed in Dyreson et al. (1999) is general enough for representing temporal aspects of semistructured data, the definition of the constraints for modeling either transaction or valid times is outside the main goal of that work. Moreover, we differ from our previous work (Combi et al. 2004) by proposing a formal and complete analysis of the possible temporal dimensions on semistructured data.

The structure of the paper is as follows: in Section 2 we give some basic notions and definitions, and introduce the clinical domain we will use as a motivating example for our proposal, while in Section 3 we briefly report related work. In Section 4 we introduce the Property-based Semistructured Temporal Data Model (PSTDM) by describing and discussing constraints when dealing with transaction time and valid time, separately. Sections 5 and 6 discuss about operations and related algorithms for an incremental construction of valid time and transaction time PSTDM graphs, respectively. Section 7 deals with managing transaction and valid times together. In Section 8, we discuss pros and cons of PSTDM, while in Section 9 we sketch some conclusions and possible lines for future work.

2 Background and motivation

In this section we provide some basic notions useful for the comprehension of the proposal. Moreover, we introduce the motivation of the work, the issues we deal with, and the example we use as case study.

In the past years, in the classical database context, the representation of time-varying information was studied and extensions of data models considering the

given (valid/transaction) time dimension were proposed in the literature (Jensen et al. 1998; Jensen and Snodgrass 1999). We recall here that the *valid time* (VT), which is user-provided, represents the time when a fact is true in the considered domain, while *transaction time* (TT), which is system-generated, represents the time when a fact is current in the database and may be retrieved (Jensen et al. 1998; Jensen and Snodgrass 1999). In the literature different temporal data types have been considered. They are *time instants*, *time intervals*, and *temporal elements*. A *time instant* is a particular point on the time line. A *time interval* $[t_s, t_e]$ is the time period between the two instants t_s and t_e (with $t_s \leq t_e$): TT is usually represented through time intervals. A *temporal element* $\{[t_1, t_2], [t_3, t_4], ..., [t_{k-1}, t_k]\}$ is a finite set of disjoint time intervals. In this work, we represent the valid time dimension by means of *temporal elements*, because they allow more expressiveness, for example in describing the valid time of entities and relations in consecutive snapshots.

In the relational database context, there was a natural extension of temporal data models to bitemporal ones, since valid and transaction times are fundamental dimensions that, if taken into account in the same data representation, allow one to recover (and query on) past states of both the domain and the database (Jensen and Snodgrass 1999). In order to give an exhaustive treatment of issues deriving from the bitemporal representation of data, we study time-related semantics of semistructured data by proposing constraints and operations related to the representation and manipulation of semistructured information considering their valid and transaction time dimensions together.

At the best of our knowledge, the issue related to the definition of the set of constraints needed to manage together and in a correct way both transaction and valid times has not been yet extensively considered in the literature, and the few papers dealing with the consistency of temporal XML documents (Campo and Vaisman 2006; Rizzolo and Vaisman 2008) focused on transaction time.

The need of considering constraints on transaction and valid times of semistructured databases stems from two different requirements: from a first point of view, we want that a temporal semistructured database is consistent with respect to the real world history it represents. An example of constraint considering this kind of consistency is that any property of a given entity cannot be valid when the entity itself is not valid. We call this first kind of consistency *real-world consistency*. From another point of view, we want that the temporal semistructured database is consistent with respect to the database history, it represents. This kind of consistency (*database consistency*) deals, for example, with the fact that, after any data insertion, any property of an entity cannot have at the same time two distinct values, if we are not allowing multivalued properties in our data model. It is worth noting that the same reasons motivated several research efforts in the temporal database community, focusing on temporal extensions both to the relational data model (Snodgrass 2000) and to the entity-relationship data model (Combi et al. 2008; Gregersen and Jensen 1999): as an example, we mention here the introduction of the concepts of temporal key and of the sequenced semantics both for valid and transaction times (Snodgrass 2000) for relational databases.

In the following, we will show that such needs related to real-world and database consistencies for semistructured databases generate, differently from what happens for structured data models (as the relational, the entity-relationship, and the object-oriented ones), not only constraints dealing with specific data values, but also

constraints related to the maintenance of some basic structural properties of the database. We then propose an approach to verify the consistency of semistructured data in an incremental way: more precisely, we propose some algorithms which guarantee that data insertion/modification/update on a consistent semistructured temporal database produce another consistent semistructured temporal database. Throughout the paper, we will use some real-world examples to describe and discuss the introduced concepts; in particular we will consider a medical scenario related to patients suffering from chronic stable angina, a cardiological pathology (Gibbons et al. 1999).

Stable angina corresponds to (predictable) chest pain or discomfort that typically occurs with activity/stress and is a type of chest discomfort caused by poor blood flow through the blood vessels (coronary vessels) of the heart muscle. The most common cause of angina is coronary artery disease (CAD). Chronic stable angina is caused by a chronic narrowing of coronary arteries. The narrowing may be usually observed through angiography. As coronary arteries are narrowed, the myocardial tissue perfused by the considered arteries will not receive adequate blood flow because of the limited flow capacity. This results in an occurrence of relative ischemia when the oxygen demand increases, leading to anginal pain, for example during physical exertion, large meal or emotional stress. CAD has different levels of severity according to the number of significantly diseased vessels and/or to the degree of obstruction of arteries.

Chronic stable angina is most commonly treated with drugs that reduce oxygen demand. Drugs include beta-blockers, calcium-channel blockers, nitrodilators. They act by decreasing heart rate, contractility, afterload and preload. The most commonly used drug to treat angina is nitroglycerin: it is a vasodilator, thus making more oxygen available to the heart muscle. Beta-blockers and calcium channel blockers act to decrease the heart's workload, and thus its requirement for oxygen.

Non-pharmaceutical treatments are balloon angioplasty and coronary bypass surgery: as for angioplasty, a balloon is inserted at the end of a catheter and inflated to widen the arterial lumen. Stents are often used to maintain the arterial widening. On the other hand, coronary bypass surgery consists in bypassing constricted arteries with venous grafts. This is much more invasive than angioplasty.

The main goals of treatment in angina pectoris are relief of symptoms, slowing progression of the disease, and reduction of future events, especially heart attacks and, of course, death.

According to this scenario, patients with chronic stable angina undergo a follow-up regimen, made of pharmaceutical therapies, angiographies, rest and exercise electrocardiograms, chest x-rays, possible interventions: to assess the evolution of CAD and to suitably manage the patient's health status, (historical) data about the follow-up regimen and the patient's symptoms have to be properly acquired and analyzed by physicians. It is widely known in the literature that temporal representation and reasoning in medicine is a key issue to support medical decision making (Combi and Pozzi 2006). Indeed, different medical tasks are based on temporal clinical data processing; among the others, we mention here decision support for diagnosis, advice on therapy, clinical data summarization, intensive care unit (ICU) monitoring, and epidemiological studies (Combi et al. 2010). In the following, we will focus in particular on the semistructured nature of these temporal data, as they come from different sources, are often given through forms containing natural languages sentences, and may be characterized by different, coexisting, levels of detail.

3 Related work: temporal data models

Recently, some research contributions are concerned with temporal aspects in semistructured databases. While they share the common goal of representing time-varying information, they consider different temporal dimensions and adopt different data models and strategies to capture the main features of the considered notion of time (Ali and Pokorný 2006; Amagasa et al. 2001; Chawathe et al. 1998, 1999; Dyreson 2001; Dyreson and Grandi 2009; Dyreson et al. 2006; Grandi and Mandreoli 2000; Grandi et al. 2005; Li et al. 2010; Mandreoli et al. 2006; Mendelzon and Rizzolo 2004; Noh et al. 2008; Oliboni et al. 2001; Rizzolo and Vaisman 2008; Rosado et al. 2007; Wang and Zaniolo 2002; Wang et al. 2008; Wang and Zaniolo 2008).

The Delta Object Exchange Model (DOEM) proposed in Chawathe et al. (1998, 1999) is a temporal extension of the Object Exchange Model (OEM) (Papakonstantinou et al. 1995), a simple graph-based data model, with objects as nodes and object-subobject relationships represented as labeled arcs. Change operations (i.e., node insertion, update of node values, addition and removal of labeled arcs) are represented in DOEM by using *annotations* on nodes and arcs of an OEM graph for representing the history. Intuitively, annotations are the representation of the history of nodes and edges as it is recorded in the database. This proposal takes into account the *transaction time* dimension of a graph-based representation of semistructured data. DOEM graphs (and OEM graphs as well) do not consider labeled relationships between two objects (actually, each arc is labeled with the name of the unique pointed node).

The Temporal Graphical Model (TGM) (Oliboni et al. 2001) is a graphical model for representing semistructured data dynamics. This model uses *temporal elements*, instead of simple intervals, to keep trace of different time intervals when an object exists in the reality. In Oliboni et al. (2001) the authors consider only some preliminary issues (e.g. admitted operations and queries) related to the *valid time* representation.

The Graphical sEmistructured teMporal data model (GEM), proposed in Combi et al. (2004), is a data model based on labeled graphs and allows one to represent in a uniform way semistructured data and their temporal aspects by considering transaction time. In the GEM data model, differently from other proposals (Chawathe et al. 1999; Dyreson et al. 1999), labels are associated both to edges and to nodes: this way, a more compact, graph-based representation of semistructured databases is possible. In Combi et al. (2004) a first formalization of the set of constraints needed for correctly managing transaction time is given.

Focusing more closely on contributions dealing with temporalities in XML data and queries, different approaches have been proposed in the last years trying to extend the XML data model, the XPath data model and query language, and several further XML-based methodologies and technologies. Some proposals start from XPath (World Wide Web Consortium 1999) and extend it to manage time dimensions (Amagasa et al. 2001; Dyreson 2001; Zhang and Dyreson 2002). The Temporal XPath Data Model (Amagasa et al. 2001) is an extension of the XPath Data Model (World Wide Web Consortium 1999) capable of representing history changes of XML documents. In particular, this approach introduces the *valid time* label only for edges in the XPath model. An extension of XPath for supporting

transaction time is presented in Dyreson (2001). The proposed extension allows the representation of the history of an XML document as a sequence of XML documents representing the versions of the considered XML document. According to the data model extension, the author extends the query language to query with respect to transaction time. At this aim several new axes, node tests, and temporal constructs are added. Another extension of XPath is proposed in Zhang and Dyreson (2002), where the authors extend the XPath data model and query language to include valid time. In particular, they extend the XPath data model by adding to each node a superset of disjoint intervals or instants representing valid time, and impose that the valid time of a node is constrained to be a subset of the valid time of the node parent. Moreover, a valid-time axis, with a suitable syntax, is added to the query language to retrieve nodes according to a valid time view. The valid-time axis of a node allows one to refer to the valid-time information of the node itself.

In Dyreson (2001), the authors propose to extend Web servers in order to deal with transaction time (i.e., modification time) of resources, such as XML documents. In Grandi and Mandreoli (2000), Wang and Zaniolo (2002) some proposals devoted to represent and query histories of XML documents are introduced. In Hunter (2002), Hunter proposes a framework for merging potentially inconsistent (semi) structured text using temporal knowledge. In Buneman et al. (2002), the authors propose an archiving tool for XML data which provides meaningful change descriptions and is able to support different basic functions concerning the evolution of data. For example, it is possible to retrieve a specific version of data from the archive and to query the temporal history of any element.

In Grandi et al. (2005), the authors present an XML-based temporal data model for the representation and management of versioned normative texts. The model supports multiple temporal dimensions, all involved in the law application lifecycle: publication time, validity time, efficacy time, and transaction time. Versioned texts are represented through XML documents, allowing one to manage different temporalities. Law texts are represented through a tree-like structure, where the *temporal pertinence* of a sub-tree may be contained in the temporal pertinence of the overall tree. The dynamics management and temporal querying of normative texts are then faced; in particular, a suitable temporal extension of XQuery is discussed. In Mandreoli et al. (2006), the authors propose a solutions for supporting temporal slicing when querying temporal XML documents, where XML documents are represented as trees made of timestamped nodes. No specific constraints are specified for the (generic) temporal dimensions on nodes.

In Dyreson et al. (2006), the issue of providing the XML data evolution by web servers is faced: in particular, the authors present a system to build a temporal data collection, which records the history of each published datum and not only its current state. A temporal schema mediates the interaction between the publisher and the reader in exchanging temporal data. The schema describes how to construct a temporal data collection by merging individual states into an integrated history. Temporalities of XML elements are provided through timestamping attributes and through different versions of the same element holding on different intervals. The considered scenario is that of NCBI databases managing data on genes and proteins in the biological domain.

In Noh and Gadia (2006), the authors face a different issue, namely that of managing parametric temporal data through an XML encoding. Parametric temporal

data are mainly temporal relations with attribute timestamping. Timestamping is obtained through temporal elements (i.e., sets of non overlapping intervals) rather than intervals, to provide a more compact and set-theoretic representation of the temporal dimension. Two different approaches are thus compared to this regard, based on XQuery and on an extension of SQL, respectively. Then, in Noh et al. (2008), a complete XML-based methodology is proposed for parametric temporal database model implementation: a temporal database system is built on top of an original XML storage system.

In Ali and Pokorný (2006) a number of temporal XML data models are considered and their comparison is provided according to (i) time dimension (valid time, transaction time), (ii) support of temporal elements and attributes, (iii) querying possibilities, (iv) association to XML Schema/DTD, and (v) influence on XML syntax. The authors conclude that the considered approaches to the management of temporal information using XML mostly do not require changes of current XML standards. No attention is paid to possible constraints among temporal dimensions of different XML elements, as the document hierarchy corresponds usually to a containment relationship of the temporal dimensions of the considered elements.

In Rosado et al. (2007), the authors deal with the issue of branching in versions for (versioned and temporal) XML documents. Suitable XML representations are provided both for versioned XML documents, their version history, and the history of changes to the original XML document. Versions of XML documents are considered also in Wang and Zaniolo (2008): efficient techniques for managing multi-version document histories and supporting powerful temporal queries on such documents are proposed. The authors show that the data definition and manipulation framework of XML and XQuery can effectively support temporal models and historical queries without requiring extensions to the current standards.

In Wang et al. (2008), the authors propose an XML-based approach for managing transaction time in temporal databases. In particular, they design an architecture using XML to support the representation of the database history, and XQuery to retrieve temporal information. In this proposal, XML is used to represent history by using a temporally grouped approach where the values of each attribute are related to a set of time intervals denoting their temporal dimension: for each value of the attribute, the corresponding transaction time interval is represented. This approach allows the user to express temporal queries in XQuery without requiring the introduction of new temporal constructs in the language.

Recently, the support of temporal queries on XML keyword search engines has been considered in Manica et al. (2010): the authors' proposal is based on identifying temporal constraints in a keyword query and intercepting the query processing, executed by a conventional XML search engine, in order to evaluate those constraints.

In Li et al. (2010), the authors introduce TempXTQ, a pattern-based temporal XML query language, with a Set-based Temporal XML (STX) data model which uses hierarchically-grouped data sets to uniformly represent both temporal information and common XML data. TempXTQ deploys various patterns equipped with certain pattern restructuring mechanisms to present requests on extracting and constructing temporal XML data. It is worth noting that in the STX data model, the usual temporal integrity constraint is assumed to hold for every temporal XML element: the temporal dimension of a node contains the (possibly different) temporal dimensions of its children nodes.

A further proposal for the management of valid time for XML document is described in Mendelzon and Rizzolo (2004), Vaisman et al. (2004): the authors consider several aspects of the topic, ranging from data modeling and query languages, to the implementation of indexing structures. In Vaisman et al. (2004), the authors introduce in an informal way some consistency conditions for graphs representing temporal XML documents. Consistency is mainly based on the fact that each *snapshot* graph (i.e., the graph composed by nodes valid at a given time) must be a rooted tree. The authors' approach for modeling valid time consistency of XML data is limited to manage a *sequenced semantics* (Snodgrass 2000) of XML data: a graph representing a temporal XML document is a compact representation of several snapshot graphs, each of them representing the XML document valid at a specific time point. In this direction, in Campo and Vaisman (2006), the authors deal with inconsistencies in temporal XML documents where transaction time is represented: more precisely, they consider how to manage inconsistencies arising from temporal XML documents, when transaction times of containment edges are not able to properly represent time varying (well formed) XML documents. In a recent work (Rizzolo and Vaisman 2008), the authors propose a data model for representing temporal information in XML documents and study the related temporal constraints. They also present algorithms for validating XML documents with respect to the defined constraints and methods for fixing inconsistent documents. In the proposed data model, they represent XML documents by means of labeled graphs with temporal annotated edges. The time dimension they consider is transaction time and the type of temporal data is the temporal element, even though in the paper they deal with some abstract representation of temporal XML documents where the difference between valid and transaction times is not adequately highlighted; moreover they actually work with single intervals, instead of temporal elements.

Few proposals explicitly consider bitemporal semistructured models (Amagasa et al. 2001; Dyreson et al. 1999; Wang and Zaniolo 2004). In Dyreson et al. (1999) the authors propose a graph-based model which uses labeled graphs to represent semistructured databases and the peculiarity of these graphs is that each edge label is composed by a set of descriptive properties, i.e., meta-data (e.g. name, transaction time, valid time, security properties of relationships). Edges may have different properties: a property may be present in an edge and missing in another one. This proposal is very general and extensible: any property may be used and added to adapt the model to a specific context. In particular, the model allows one to represent temporal aspects and to consider only a temporal dimension or multiple temporal dimensions: to this regard, some examples of constraints which need to be suitably managed to correctly support semantics of the time-related properties are provided, both for querying and for manipulating graphs. We can state that in principle this proposal can be used as a bitemporal model, although some other constraints must be specified in order to guarantee the consistency of the specific concepts included in the set of labels of the model itself. This last issue has not been completely addressed by the authors and is outside the main goal of that work; in this perspective our work can be considered complementary to that in Dyreson et al. (1999). A work proposing a solution to the problem of representing valid and transaction times in XML document is Amagasa et al. (2001): the Bitemporal XML Data Model is an extension of the XPath Data Model capable of representing the history of both the domain and the database of XML documents. In particular, this approach introduces the pair of *valid time, transaction time* labels only for edges in the XPath model. In this

work, no formalization of the set of constraints required to guarantee the consistency of bitemporal documents is provided. Finally, in Wang and Zaniolo (2004), the XML-based bitemporal data model XBiT is proposed: the authors show that valid-time, transaction-time, and bitemporal databases can be naturally viewed in XML using temporally-grouped data models. Similarly to the approach proposed in Noh and Gadia (2006), here the focus in on the representation of temporally-grouped data: in these models tuples are composed by non-atomic multivalued timestamped attributes but all the attributes of a tuple have the same overall timespan. Then, authors show that complex historical queries, that would be very difficult to express in SQL on relational tables, may be easily expressed in standard XQuery on such XML-based representations.

4 The property-based semistructured temporal data model

In this section we introduce a graph-based data model (PSTDM: Property-based Semistructured Temporal Data Model), which is inspired by the one described in Dyreson et al. (1999), with the aim of representing in a uniform way semistructured information by considering also their time dimensions.

In Dyreson et al. (1999) the authors proposed an extensible data model general enough to represent multiple aspects of semistructured data, such as security and temporal properties. Such aspects are represented in labeled graphs by means of sets of descriptive properties, where each property is a kind of meta-data denoted as `property_name`: *property_value*.

As defined in Dyreson et al. (1999), a semistructured database $DB = (V, E, \&root, \Gamma)$ (hereinafter DBJ), is a graph where V is a set of nodes, E is a set of labeled directed edges, $\&root$ is a single root node, and Γ is a collection of property operations that determine the semantics of properties, when querying data. An edge in E, from a node v to a node w, has a label $\mathcal{L}$, composed by a set of m pairs $\{(p_1 : x_1), \ldots, (p_m : x_m)\}$, where each p_i is the name of a property and x_i a value in the property domain. Each property name is unique; moreover, a *required property* p_i is denoted as $(!p_i : x_i)$ property. Property operations exist in Γ for each p_i: i.e., for each property p_i, operations *collapse*, *match*, *coalesce*, and *slice* on property values should be specified in Γ. These operations are needed to accommodate properties in queries.

In this paper, we focus only on properties that represent temporal aspects of semistructured data, by considering the classical notions of valid and transaction times studied in the past years in the context of temporal databases (Jensen et al. 1998; Jensen and Snodgrass 1999); these notions will be represented as required properties. However, the model and algorithms we propose are general enough to be applied also in scenarios when other properties are considered. Moreover, we explicitly formalize all the temporal operations and constraints needed to guarantee the consistency of the model; as we do not focus on queries on semistructured databases, in the proposed data model we will not consider the set of operations Γ.

To motivate our proposal, we point out some limitations of the DBJ model and our extensions to overcome them:

- DBJ graphs have property labels only on edges, because in the authors' opinion nodes are completely described by the paths that lead to them. This limitation does not allow one to represent general relationships (different from the containment) between nodes, unless introducing ad-hoc nodes representing the semantics of the relationships. For example, for expressing validity of relationships between people, as it happens in ontological representations, both nodes and edges may be labeled with the valid time property; in this case with edges it is possible to represent different relationships, such as *parent_of*, *friends_of*, and so on. Our choice is to use a higher-level graph-based model with (required) properties both on nodes and edges.
- Valid time both for nodes and edges is not entirely redundant: indeed, introducing valid time only on edges, and computing the validity of a node as the union of the lifetime of the incoming edges, does not allow one to represent the situation when a node exists at times other than times of its relationships. For example, if we consider to represent facts related to a conference organization, the organization activity, modeled by a relation *Organizes* can be naturally performed and completed before the beginning of the conference. This situation is modeled in a labeled graph, by introducing two nodes representing the organization committee and the considered conference and the *Organizes* edge between them with a validity that is not included in the valid time interval of the *Conference* node (the valid time of the edge is before the one of the *Conference* node w.r.t. the Allen's relationships (Allen 1983)).
- We use as domain of the valid time property the set of temporal elements. We recall that a *temporal element* $\{[t_1, t_2], [t_3, t_4], ..., [t_{k-1}, t_k]\}$ is a finite set of disjoint time intervals, that is closed under the set theoretic operations of union, difference and intersection, as defined in Garani (2006).
- Our model is coalesced, a term coined in Böhlen et al. (1996) for relational databases, to indicate that all pairs of distinct tuples from a relation r are either not value-equivalent, or, if they are value-equivalent, then they must be non-adjacent and non-overlapping. We will see that our operations explicitly enforce coalescing on update and insertion of paths.

Formally, a *PSTDM graph* is a rooted, acyclic, connected, directed, and labeled graph $\langle N, E, r \rangle$, where:

1. N is a (finite) set of labeled nodes (actually, N is the set of node identifiers n_i). In VT PSTDM graphs (i.e. PSTDM graphs representing the Valid Time dimension), each node $n_i \in N$ has as label a set composed by three required descriptive properties, of the form $\{(!\texttt{name} : node\_name_i), (!\texttt{content} : c_i), (!\texttt{vt} : t_i)\}$, where the value $node\_name_i$ is the *name* of the node, c_i its *content*, and t_i the value for the temporal dimension `vt`, respectively. Nodes of TT PSTDM graphs (i.e. PSTDM graphs representing the Transaction Time dimension) have labels composed by the properties $\{(!\texttt{name} : node\_name_i), (!\texttt{content} : c_i), (!\texttt{tt} : t_i)\}$ representing the *name* of the node, its *content*, and the value for the temporal dimension `tt`, respectively.

The domains for the properties are: $domain(\texttt{name}) = \mathcal{S}$ (where $\mathcal{S}$ is the set of *strings*), $domain(\texttt{content}) = \mathcal{S} \cup \{\perp\}$ (where $\perp$ represents the undefined value), $domain(\texttt{vt}) = \mathcal{TE}$ (where $\mathcal{TE}$ is the set of temporal elements), and $domain(\texttt{tt}) = \mathcal{CI}$ (where $\mathcal{CI}$ is the set of closed intervals).
To make the formalization more readable, we introduce the node labeling function ℓ that, given a node, returns its set of descriptive properties. ℓ can be seen as the composition of three single-valued functions $\ell_{\mathcal{N}} : N \longrightarrow \mathcal{S}$, $\ell_{\mathcal{C}} : N \longrightarrow \mathcal{S} \cup \{\perp\}$ and $\ell_{\mathcal{T}_{vt}} : N \longrightarrow \mathcal{TE}$ (or $\ell_{\mathcal{T}_{tt}} : N \longrightarrow \mathcal{CI}$), which return, for a given node, the value of the `name`, `content`, and `vt` (`tt`) property, respectively.

2. E is a set of labeled edges. Each edge $e_j \in E$ has as label a set of two required properties of the form $\{(!\texttt{name} : edge\_name_j), (!\texttt{vt} : t_j)\}$ and $\{(!\texttt{name} : edge\_name_j), (!\texttt{tt} : t_j)\}$ respectively for VT and TT PSTDM graphs; $edge\_name_j$ is the *name* representing the semantics of the edge, and t_j the value for the represented temporal dimension: $domain(\texttt{name}) = \mathcal{S}$, $domain(\texttt{vt}) = \mathcal{TE}$, and $domain(\texttt{tt}) = \mathcal{CI}$. Edges have no identifiers: an edge can be identified by its label and by the two connected nodes, because between two nodes we suppose to have only one edge with a particular name and a particular value for the represented temporal dimension.
3. $r \in N$ is the unique root of the graph and is introduced in order to guarantee that all the other nodes can be reached (starting from the root).

There are some general constraints for a PSTDM graph: indeed, we impose internal nodes to have a not defined value as content (the $\perp$ value is admissible for the `content` property); on the other hand, a (primitive) content can be only in leaves. Thus, for example, a VT PSTDM graph must satisfy the following constraints:

1. $\forall n \in N((\exists e \in E(e = \langle (n, m), \{(!\texttt{name} : name_e), (!\texttt{vt} : time_e)\}\rangle)) \rightarrow \ell_{\mathcal{C}}(n) = \perp)$.
2. $\forall n \in N((\ell_{\mathcal{C}}(n) \neq \perp) \rightarrow \neg\exists e \in E(e = \langle (n, m), \{(!\texttt{name} : name_e), (!\texttt{vt} : time_e)\}\rangle))$.

Hereinafter, we will call *complex nodes* those representing complex objects without a value for the `content` property, and *simple nodes* the other ones, i.e. leaves representing atomic features.

In the following sections we will focus on constraints we have to introduce on PSTDM graphs, to suitably represent the different semantics related to the VT dimension, to the TT dimension, and VT and TT dimensions together. We preferred to adopt a semistructured data model, instead of dealing directly with XML: this way, as mentioned in the introduction and according to other proposals (Chawathe et al. 1999; Dyreson et al. 1999; Oliboni et al. 2001), the proposed solution can be considered as a logical model for temporal semistructured data, which can be translated into different XML-based languages/technologies. Moreover, we do not focus on data structures and algorithms to store and manage (PSTDM) graphs in an efficient way: our main aim is the definition of the semistructured temporal data model, the needed constraints to manage time in a consistent way, and the operations required to (correctly) modify PSTDM graphs.

5 Managing valid time with PSTDM

In this section we consider the issues related to the management of VT in the semistructured data context. In this case, the considered temporal dimension is given

by the user, being valid time related to the description of the considered real world. Thus, a VT PSTDM graph represents complex objects and their valid times through complex nodes; temporal relations between objects are represented through suitable labeled edges between complex nodes. Simple (temporal) properties of complex objects are represented through labeled edges between complex nodes and leaf nodes, which contain the property values. Valid times of properties are represented through the valid time of the corresponding edges, while valid times of simple (leaf) nodes are derived from all the valid times of their ingoing edges.

Thus, constraints and operations must be able to guarantee that the history of the given application domain is consistent.

5.1 Constraints for valid time

The following constraints on a PSTDM graph allow us to explicitly consider the features of semistructured data timestamped by the valid time. When considering the valid time dimension, we define the validity of a node (edge) with respect to the time, in the represented domain. Thus, a node (edge) is considered as *valid* in the time period represented by its related time intervals (the ones composing its temporal element).

1. The temporal element of an edge between a complex node and a simple node must be related to the temporal element of the complex node.

$$\begin{aligned}\forall e_j \in E \ \Big(& \big(e_j = \langle (n_h, n_k), \{(!\texttt{name} : edge\_name_j), (!\texttt{vt} : T_j)\}\rangle \\ & \wedge \ \ell_{\mathcal{T}_{vt}}(n_h) = T_{n_h} \wedge \ell_C(n_k) \neq \perp\big) \\ & \rightarrow \forall [t_{js}, t_{je}] \in T_j \, \exists [t_{n_{hs}}, t_{n_{he}}] \in T_{n_h} \big(t_{js} \geq t_{n_{hs}} \wedge t_{je} \leq t_{n_{he}}\big)\Big)\end{aligned}$$

   We suppose that a complex node is connected to its properties (simple nodes) during its temporal elements, i.e. when it is valid. Thus, the valid time of the edge between a complex node and a simple node must be contained in the valid time of the complex node. This means that each time interval of the edge cannot start before and cannot end after a valid time interval of the temporal element related to the complex node.
2. The valid time of each simple node is related to the valid times of all its ingoing edges:

$$\begin{aligned}\forall n_h \in N \Big(& \big(\ell_C(n_h) \neq \perp \wedge \ell_{\mathcal{T}_{vt}}(n_h) = T_{n_h}\big) \rightarrow T_{n_h} = \bigcup \{T_{e_j} | \exists e_j \in E \\ & \big(e_j = \langle (n_k, n_h), \{(!\texttt{name} : e\_name_{e_j}), (!\texttt{vt} : T_{e_j})\}\rangle\big)\}\Big)\end{aligned}$$

   T_{n_h} is the union of temporal elements (Garani 2006) of the ingoing edges for the considered node n_h, which are closed w.r.t. the union operation, thus, T_{n_h} is itself a temporal element. This is due to the fact that simple nodes represent properties of complex nodes, and thus the valid time of a simple node must be related to the valid times of all its ingoing edges. This represents the fact that in the reality, a property can be present only if it is related to an entity.
   In the case of VT this constraint is specified, and thus holds, only for simple nodes: indeed, while the valid time of a simple node is constrained by the fact that the given node represents a simple property of some complex node, the

valid time of a complex node is not related to the VT of its incoming edges, as it represents the intervals of time over which the represented entity holds. More complex valid time semantics for complex nodes and related edges can be specified, as discussed in the following.

3. When representing valid time we impose to have unique simple node labels:

$$\forall n \in N \left(\ell_C(n) \neq \perp \rightarrow \neg \exists n' \in N \left(n' \neq n \wedge \ell_N\left(n'\right) = \ell_N(n) \wedge \ell_C\left(n'\right) = \ell_C(n)\right)\right)$$

4. When representing valid time we impose to have unique edge labels between each pair of nodes:

$$\begin{aligned} \forall e \in E \ \ & \left(e = \langle (n, m), \{(!\texttt{name} : name_e), (!\texttt{vt} : T_e)\}\rangle \rightarrow \neg \exists e' \right. \\ & \left. \in E \left(e' = \langle (n, m), \{(!\texttt{name} : name_e), (!\texttt{vt} : T_{e'})\}\rangle \wedge T_e \neq T_{e'}\right)\right) \end{aligned}$$

5. Other possible optional constraints can be introduced, according to the modeled domain. Let us consider the constraints for imposing restrictions on the temporal element of an edge connecting two complex nodes. These constraints are strictly related to the semantics of represented objects and relationships. For example, when considering the relation *Is_descendant* between two *Person* nodes, we can assume that the relation cannot be established before that both nodes are valid in the considered domain, but it can be maintained after the end of one (or both) of the connected nodes. The relation *Is_descendant* represents a family tie which is still valid after the end (death) of the considered people. In this case, the following formula holds:[1]

$$\begin{aligned} \forall e_j \in E \ \ & \left(\left(e_j = \left\langle (n_h, n_k), \left\{(!\texttt{name} : Is\_descendant), \left(!\texttt{vt} : \left\{\left[t_{js}, \_\right]\right\}\right)\right\}\right\rangle \right. \right. \\ & \wedge \ \ell_N(n_h) = \text{Person} \wedge \ell_{\mathcal{T}_{vt}}(n_h) = \{[t_{hs}, \_]\} \\ & \left. \left. \wedge \ell_N(n_k) = \text{Person} \wedge \ell_{\mathcal{T}_{vt}}(n_k) = \{[t_{ks}, \_]\}\right) \rightarrow t_{js} = \max(t_{hs}, t_{ks})\right) \end{aligned}$$

On the other hand, if we consider to represent future events, as, for example, the information about a conference organization, we could consider a constraint imposing that the relation *Organizes* between a *Person* node and a *Conference* node cannot be established before that the *Person* node is valid in the considered domain, and cannot continue after the end of the *Person* node (in this example the validity of the edge is not related to the validity of the *Conference* node, and the validity of the *Person* node is related to its life).

[1] Hereinafter, in the formulae we will use a prolog-like notation: each part of a node/edge label which could have any value in the formula is represented by the "do not care" symbol "_" in the corresponding position in the label.

If we assume that the organization activity of a conference ends at most at the beginning of the conference, then the following formula must hold:

$$\begin{aligned}\forall e_j \in E \ \big(\big(e_j = \langle (n_h, n_k), \{(\texttt{!name} : \mathit{Organizes}), (\texttt{!vt} : \{[t_{js}, t_{je}]\})\}\rangle \\ \wedge\ \ell_N(n_h) = \text{Person} \wedge \ell_{\mathcal{T}_{vt}}(n_h) = \{[t_{hs}, t_{he}]\} \\ \wedge\ \ell_N(n_k) = \text{Conference} \wedge \ell_{\mathcal{T}_{vt}}(n_k) = \{[t_{ks}, \_]\}\big) \\ \rightarrow\ \big(t_{js} \geq t_{hs} \wedge t_{je} \leq \min(t_{he}, t_{ks})\big)\big)\end{aligned}$$

Otherwise, if we assume that the organization activity may go on also during the conference time and ends at most at the end of the conference, then it must hold:

$$\begin{aligned}\forall e_j \in E \ \big(\big(e_j = \langle (n_h, n_k), \{(\texttt{!name} : \mathit{Organizes}), (\texttt{!vt} : \{[t_{js}, t_{je}]\})\}\rangle \\ \wedge\ \ell_N(n_h) = \text{Person} \wedge \ell_{\mathcal{T}_{vt}}(n_h) = \{[t_{hs}, t_{he}]\} \\ \wedge\ \ell_N(n_k) = \text{Conference} \wedge \ell_{\mathcal{T}_{vt}}(n_k) = \{[t_{ks}, t_{ke}]\}\big) \\ \rightarrow\ \big(t_{js} \geq t_{hs} \wedge t_{je} \leq \min(t_{he}, t_{ke})\big)\big)\end{aligned}$$

Otherwise, if we assume that the organization activity may go on also after the end of the conference, then it must old:

$$\begin{aligned}\forall e_j \in E \ \big(\big(e_j = \langle (n_h, n_k), \{(\texttt{!name} : \mathit{Organizes}), (\texttt{!vt} : \{[t_{js}, t_{je}]\})\}\rangle \\ \wedge\ \ell_N(n_h) = \text{Person} \wedge \ell_{\mathcal{T}_{vt}}(n_h) = \{[t_{hs}, t_{he}]\} \\ \wedge\ \ell_N(n_k) = \text{Conference} \wedge \ell_{\mathcal{T}_{vt}}(n_k) = \{[t_{ks}, t_{ke}]\}\big) \\ \rightarrow\ \big(t_{js} \geq t_{hs} \wedge t_{je} \leq t_{he}\big)\big)\end{aligned}$$

Another (general) option is that we could choose to adopt an approach similar, in some sense, to the *sequenced semantics* of relational temporal databases (Snodgrass 2000). In this semantics, the history of the represented world is perceived (and managed through queries, updates, and so on) as a temporal sequence of states, which are in our case (disjoint) atemporal graphs. According to this view, we have to impose that the relations between complex nodes may be valid only in the intersection of the time intervals of the nodes. Thus, the following formula will hold:

$$\begin{aligned}\forall e_j \in E \ \big(\big(e_j = \langle (n_h, n_k), \{(\texttt{!name} : e\_name), (\texttt{!vt} : TE_j)\}\rangle \\ \wedge\ \ell_{\mathcal{T}_{vt}}(n_h) = TE_{n_h} \wedge \ell_C(n_k) = \perp \wedge \ell_{\mathcal{T}_{vt}}(n_k) = TE_{n_k}\big) \\ \rightarrow \forall [t_{js}, t_{je}] \in TE_j \exists [t_{hs}, t_{he}] \in TE_{n_h} \exists [t_{ks}, t_{ke}] \\ \in TE_{n_k}\big(t_{js} \geq \max(t_{hs}, t_{ks}) \wedge t_{je} \leq \min(t_{he}, t_{ke})\big)\big)\end{aligned}$$

In this perspective, user-defined constraints for VT PSTDM graphs could allow one to represent different kinds of temporal information, as those proposed, for example, in Terenziani and Snodgrass (2004), where the distinction of point-based vs interval-based semantics when associating a fact to a temporal dimension is discussed within the context of temporal databases. A more refined classification of temporal propositions according to their temporal features has been proposed in Shoham (1987) and widely considered in the AI area. In Combi (2000), the author proposes

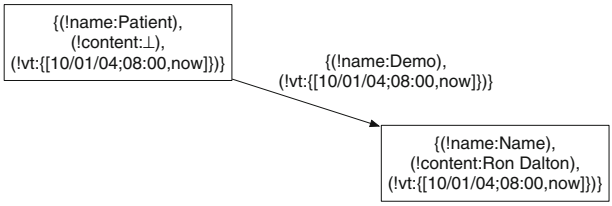

Fig. 1 A PSTDM database dealing with valid time

different semantics for temporal facts associated to multimedia data in the context of multimedia object-oriented temporal databases.

5.2 Example: managing VT with PSTDM

On January 10, 2004, at 8:00 a.m., the physician visits for the first time Ron Dalton who becomes, from this moment, his patient. Thus, we have to insert into the PSTDM database the node *Patient* with the related node *Name* having as content "Ron Dalton". The temporal element of this portion of database is {[10/01/04;08:00, *now*]}.[2] This portion of PSTDM database is represented in Fig. 1 and it does not matter when data insertion happened.

From February 1, 2004, at 22:00 p.m. to February 2, 2004, at 02:00 a.m. Ron Dalton suffered from chest pain and the physician diagnosed this symptom as Angina. Thus, we have to insert into the PSTDM database the node *Symptom*, with the related node *Description* having as content "Angina". Moreover, we have to insert the relations *P_Situation* between *Patient* and *Symptom* and *S_Name* between *Symptom* and *Description*. The temporal element of these nodes and edges are {[01/02/04;22:00, 02/02/04;02:00]}, i.e., the interval of the symptom. Figure 2 shows the database after the above operations.

For this situation the physician diagnoses the correct pathology named CAD (Coronary Artery Disease) with a low severity on February 2, 2004, at 08:00 a.m., specifying that this pathology is related to the reported symptom. Thus, we have to insert into the PSTDM database the node *Pathology* with the related nodes *Name* (having as content "CAD") and *Severity* (having as content "Low") with temporal element {[02/02/04;08:00, *now*]}. Moreover, we have to insert the relations *Diagnosis* between *Patient* and *Pathology*, *P_Name* between *Pathology* and *Name*, and *P_Severity* between *Pathology* and *Severity*, with the same temporal element. Then we have to insert the relation *Related_to* between *Pathology* and *Symptom*, with temporal element {[02/02/04;08:00, *now*]}. Managing valid time it is possible to insert an edge between a node having a past valid time and another one which has a valid time still current: in this example, the time interval of the edge *Related_to* starts after the end of the temporal element of *Symptom*, i.e., {[01/02/04;22:00, 02/02/04;02:00]}. Figure 3 shows the PSTDM graph, after adding the new information.

[2]In the following we will use the notation DD/MM/YY;HH:Mi (i.e., day, month, year, hour, and minute) for timestamps.

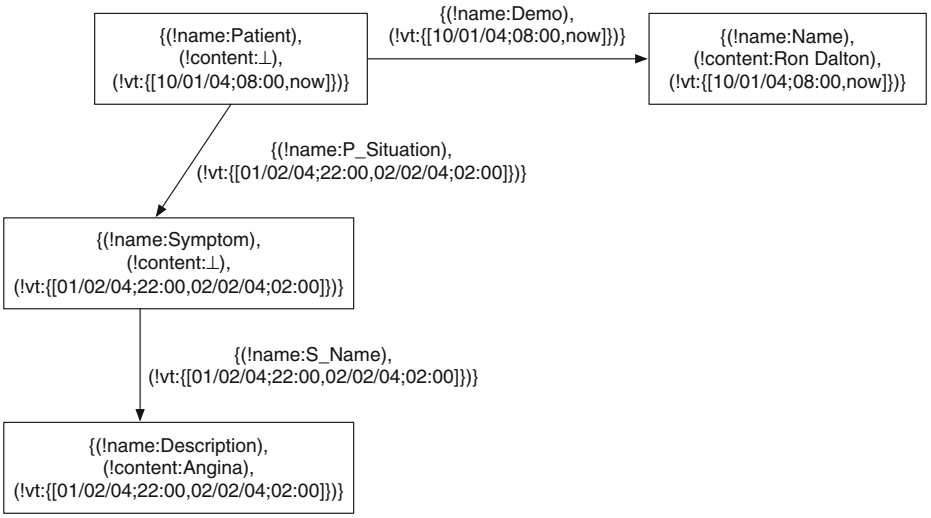

Fig. 2 Adding symptom information to the PSTDM database

On February 2, 2004, at 08:05 a.m., the physician prescribes the therapy to the patient advising to assume Nitroglycerin. Thus, we have to insert into the PSTDM database the node *Drug* and the related node *Name* with temporal element {[03/05/01; 08:00, *now*]}, which represents the time interval during which the drug is at disposal. Moreover, we have to insert the relation *Therapy* between *Patient* and *Drug*, with temporal element {[02/02/04;08:05, *now*]}, as in Fig. 4.

On February 10, 2004, at 11:00 a.m., the pathology of Ron Dalton gets worse and the severity changes from low to intermediate. Thus, we have to change the temporal element of the node *Severity*, from {[02/02/04;08:00, *now*]} to {[02/02/04; 08:00, 10/02/04;10:59]}, and the same for the temporal element of the edge *P_Severity*.

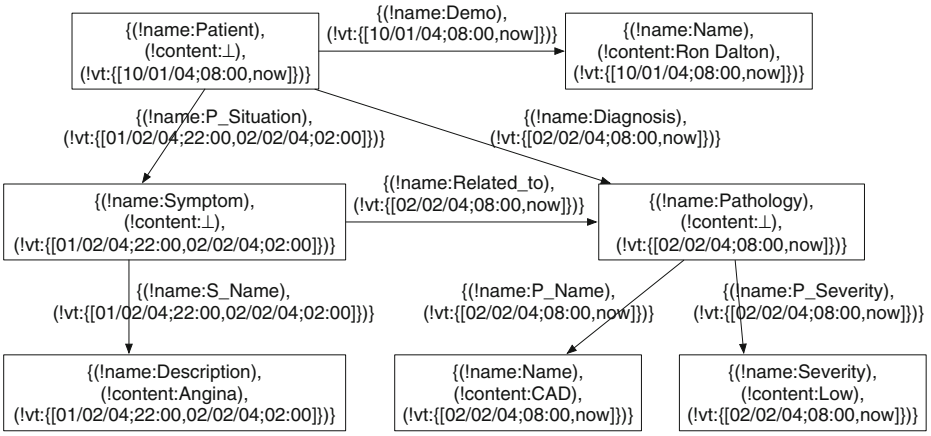

Fig. 3 Adding diagnosis information to the PSTDM database

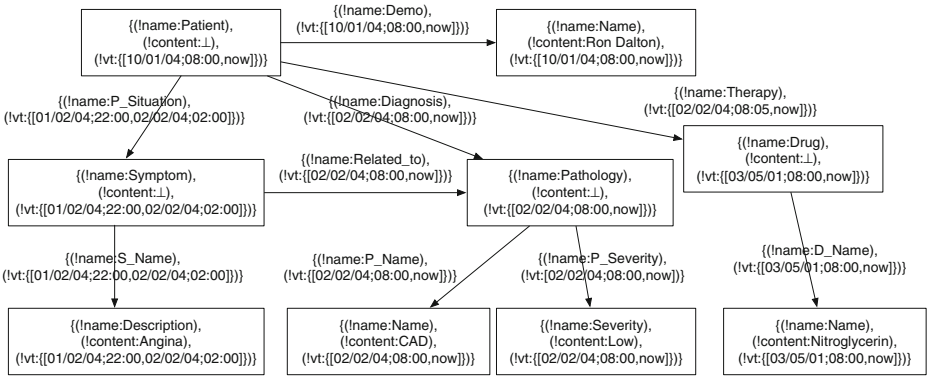

Fig. 4 Adding therapy information to the PSTDM database

Moreover, we have to insert into the PSTDM database the node *Severity* (having as content "Intermediate") and the relation *P_Severity* between *Pathology* and *Severity*. The new *Severity* and new *P_Severity* temporal elements are {[10/02/04;11:00, *now*]}. Figure 5 shows the PSTDM graph after these updates.

On February 10, 2004, at 11:30 a.m., the physician decides to interrupt the therapy with the Nitroglycerin for nearly two days, thus prescribes to the patient to suspend the drug from February 10, 2004, at 11:30 a.m. to February 12, 2004, at 08:00 a.m., and then to start it again. Thus we have to change the temporal element of the edge *Therapy*, from {[02/02/04;08:05, *now*]} to {[02/02/04;08:05,10/02/04;11:29], [12/02/04;08:00, *now*]}, Fig. 6 shows the final PSTDM graph after the last update.

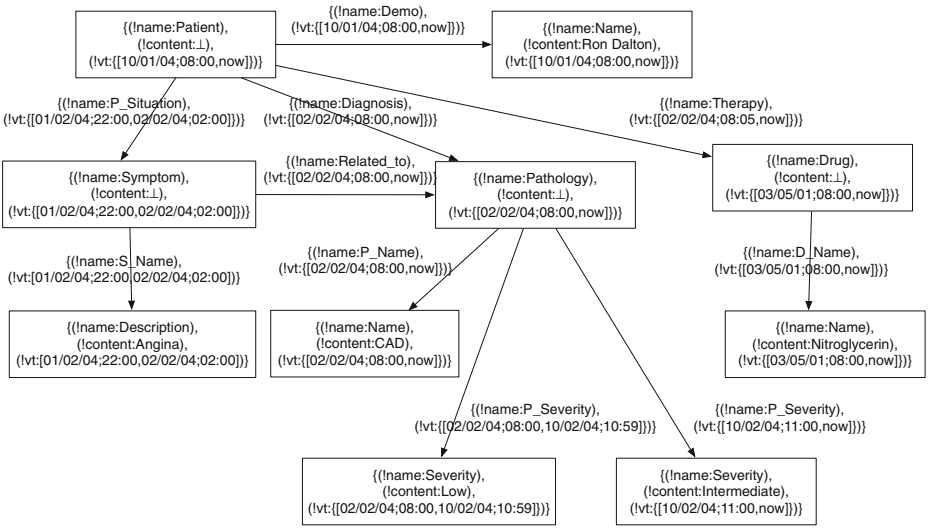

Fig. 5 Updating the pathology information of the PSTDM database

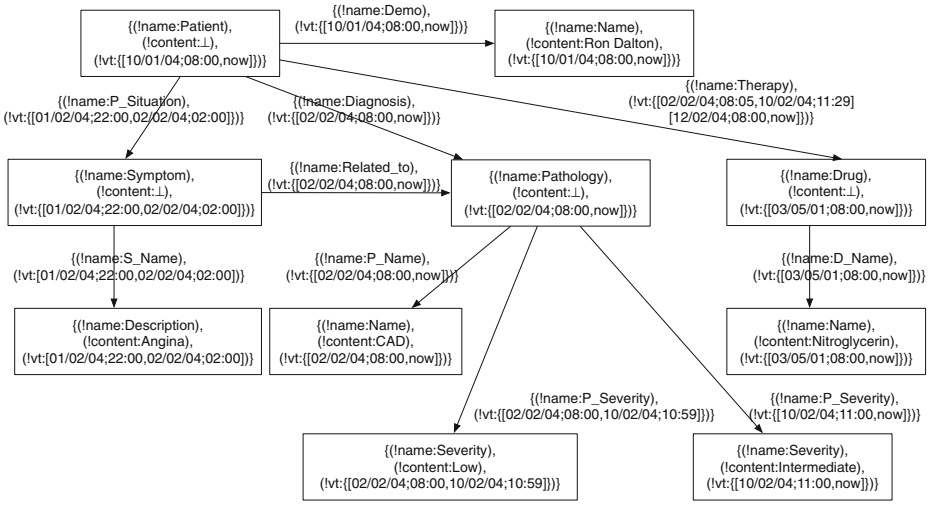

Fig. 6 Updating the validity of the therapy in the PSTDM database

5.3 Operations on PSTDM graphs for managing valid time

In this section we define the set of operations to build VT PSTDM graphs and manage valid time.

We define two groups of operations: *basic operations* and *complex operations*. Basic operations are applied by the system and are properly combined in order to implement complex operations in a correct way; they are not available to the user, while complex operations are, and allow her to modify incrementally VT PSTDM graphs. Without loss of generality, we may assume that complex operations are supported by the system and the final user (as, for example, the physician or a nurse in our case) may access it through an ad-hoc (graphical) interactive software application that uses complex operations as the basic way to interact with the semistructured temporal database.

In Section 5.1 we briefly introduced the constraints needed to manage in a correct way the valid time dimension; now we formally define the operations required to (correctly) change VT PSTDM graphs. In particular we adopt an incremental approach to build VT PSTDM graphs: we assume to start with a (correct) VT PSTDM graph and our goal is to be able to verify that any allowed operation on this graph produces another (correct) VT PSTDM graph. This incremental approach is obtained by defining ad-hoc algorithms which guarantee that any complex operation (i.e., a sequence of basic operations) produces only (correct) VT PSTDM graphs.

By considering operations producing correct effects on VT PSTDM graphs, i.e., the result of each operation is a VT PSTDM graph which satisfies the constraints defined in Section 5.1, we also guarantee that any sequence of the defined complex operations is consistent with respect to our model, thus we avoid the problem of having incorrect sequences of operations (Chawathe et al. 1999).

In the following, we will present basic operations and then we will introduce the complex ones.

Basic operations consist mainly in inserting/deleting nodes and edges.

1. **Insert a node**
   TE_insert-node$(G, Nname, Ncontent, TE_n, rootflag)$ inserts in a graph G a node with label {(!name : *Nname*), (!content : *Ncontent*), (!vt : TE_n)} and gives as result the identifier n_a assigned to the added node together with the resulting graph . If the boolean variable *rootflag* has value true, the added node is the root of the resulting graph.
2. **Insert an edge**
   TE_insert-edge$(G, n_{from}, n_{to}, Ename, TE_e)$ inserts the edge $e_j = \langle (n_{from}, n_{to}), \{(!\texttt{name} : Ename), (!\texttt{vt} : TE_e)\}\rangle$ between the nodes n_{from} and n_{to} in the graph G, and returns the resulting graph.
3. **Delete a node**
   TE_delete-node(G, n_d) removes the node n_d from the graph G.
4. **Delete an edge**
   TE_delete-edge$(G, n_{from}, n_{to}, Ename)$ removes from the graph G the edge between nodes n_{from} and n_{to}, labeled *Ename*. We remark that for valid time, the temporal element is not necessary to identify an edge between two nodes; indeed, for constraint 4 in Section 5.1, given two nodes, there is at most one edge between them with a specific label.

Let us now consider the set of available complex operations to modify a VT PSTDM graph. As we did not introduce any basic operation for node/edge updates, complex operations involving the update of some property of nodes return a graph where the modified nodes have new identifiers, as node identifiers are considered as internal features of a PSTDM graph used to manage in a proper way the structure of the graph itself. In the following, each introduced complex operation on VT PSTDM graphs will be identified by the prefix *CVT-* (Complex Valid Time).

1. **Add the root node**
   CVT-add-root-node$(Nname, TE_r)$ creates a new graph G with a root node having label {(!name : *Nname*), (!content :⊥), (!vt : TE_r)} by using the basic operation *TE_insert-node*, and gives as result the node identifier n_r and the new graph G. We suppose the user chooses this operation as the first one to build a new VT PSTDM graph; any node added to an existing VT PSTDM graph in a second moment by means of a *CVT-add-node* operation will be directly, or indirectly, reachable from the root node.
2. **Add a node**
   CVT-add-node$(G, Nname, Ncontent, TE_n, n_p, Ename, TE_e)$ adds a suitable node to the graph G, connected through a given edge to an existing node n_p, if it is possible.
   This operation is implemented by the algorithm reported in Fig. 7: the first **if** construct verifies whether the node to be added is complex. If the node is complex, the algorithm has to verify that the (optional) user-defined constraints would be satisfied by the modified graph; if it is the case, the new node is added by the suitable basic operations (lines 5 and 6 in Fig. 7).
   More complex checks have to be done when the node to be added is a simple one. In this case, before the insertion of the new node, the algorithm has to verify that valid intervals of the new node and edge are the same (end of line 14) and that the valid temporal element of the new edge is contained in the valid

Input: A PSTDM graph $G = \langle N, E, r\rangle$, the label components of the node to add, the node identifier of its parent node, and the edge connecting it to the parent node.
Output: The updated PSTDM graph $G = \langle N, E, r\rangle$, and the node identifier n_a of the added node.
Procedure
CVT-add-node (G, *Nname*, *Ncontent*, TE_n, n_p, *Ename*, TE_e)
{
1 **if** (*Ncontent* $=\perp$)
2 **then**
3 **if** (*user-defined semantic constraints are satisfied*)
4 **then**
5 $\langle G', n_a\rangle \leftarrow$ *TE_insert-node*(G, *Nname*,*Ncontent*, TE_n, false)
6 $G \leftarrow$ *TE_insert-edge*(G', n_p, n_a, *Ename*, TE_e)
7 **output**($\langle G, n_a\rangle$);
8 **else**
9 **output**($\langle G, \perp\rangle$);
10 **endif**
11 **endif**
12 **if** (*Ncontent* $\neq\perp$)
13 **then**
14 **if** ($\ell_{\mathcal{T}_{vt}}(n_p) = TE_{n_p} \wedge$ (*Constraint 1 is satisfied*) $\wedge TE_e = TE_n$)
15 **then**
16 **if** ($\exists n_h \in N(\ell_{\mathcal{N}}(n_h) =$ *Nname* $\wedge\, \ell_{\mathcal{C}}(n_h) =$ *Ncontent*))
17 **then**
18 $G' \leftarrow$ *TE_insert-edge*(G, n_p, n_h, *Ename*, TE_e)
19 $\ell_{\mathcal{T}_{vt}}(n_h) = \ell_{\mathcal{T}_{vt}}(n_h) \cup TE_e$
20 **output**($\langle G', \perp\rangle$);
21 **else**
22 $\langle G_1, n_a\rangle \leftarrow$ *TE_insert-node*(G, *Nname*, *Ncontent*, TE_n, false)
23 $G \leftarrow$ *TE_insert-edge*(G_1, n_p, n_a, *Ename*, TE_e)
24 **output**($\langle G, n_a\rangle$);
25 **endif**
26 **else**
27 **output**($\langle G, \perp\rangle$);
28 **endif**
29 **endif**
}

Fig. 7 The algorithm for adding a node in a valid time PSTDM graph

temporal element of the specified parent node n_p (line 14), to guarantee that the added simple node properly represents a property (with respect to the temporal dimension) of the parent node n_p (see also constraints 1 and 2 for VT PSTDM graphs). Otherwise, the operation fails and the graph G is returned without any change (line 27). In case the new node is suitable for the addition to graph G, the algorithm has to verify the existence of another simple node n_h with the same name *Nname* and the same content *Ncontent* in the graph G: in the positive case the operation reduces to adding an edge from n_p to this node n_h, and modifying (if needed) its temporal element according to constraint 2 for VT. Otherwise, the algorithm simply adds the new simple node and edge (lines 22 and 23).

3. **Add an edge**
CVT-add-edge(G, n_p, n_h, *Ename*, TE_e) checks that there are no other edges between nodes n_p and n_h, with the same *Ename* (see the constraint 4 of VT); furthermore, it checks the validity of the constraint (if any) with respect to the user-defined semantics of the represented relation; if any constraint is violated

the operation fails. Otherwise, if n_h is a complex node the operation is realized by calling the basic operation *TE_insert-edge*$(G, n_p, n_h, Ename, TE_e)$. If n_h is a simple node, before adding the required edge, the operation has to verify that the valid time of the parent node n_p contains the valid time of the considered edge; if it is the case, the algorithm adds the new edge by the basic operation *TE_insert-edge*, otherwise it fails. The algorithm has to suitably modify the valid time of the simple node n_h, according to Constraint 2 for Valid Time.

4. **Modify the temporal element of a complex node**
   CVT-modify-complex-time$(G, n_c, NewTE)$ checks (i) that the new valid time of the complex node is still consistent with valid times of relations with its simple nodes (i.e., valid times of edges to simple nodes must be contained in or equal to the new valid time of the complex node), and (ii) that optional user-defined constraints are satisfied by the new valid time. If all these constraints are satisfied, the operation modifies the valid time of the node n_c to *NewTE*. Basic operations are used to create a new complex node with the suitable label, to "move" all the edges pointing to/from the node to be modified to the new one (see lines 21 and 22 of the algorithm in Fig. 7 for a similar situation), and to delete the node with the old label. This operation is used even when an update of the temporal element of the node is needed: indeed, in this case $NewTE = oldTE \cup [t_s, t_e]$, where *oldTE* is the temporal element before the update and $[t_s, t_e]$ is the interval to add to *oldTE*.
5. **Modify the temporal element of an edge**
   CVT-modify-edge-time$(G, n_p, n_h, Ename, NewTE)$ modifies the valid time of an edge, only if the new valid time does not introduce a violation of the VT constraints. More specifically, if the edge to be modified is connecting a complex node to a simple one, the new valid time temporal element has to be contained in the temporal element of the connected complex node n_p (see constraint 1 for VT PSTDM graphs). The algorithm reported in Fig. 8 implements this operation. As for the previous operation, *CVT-modify-edge-time* is also used for updating the temporal element of an edge.
6. **Modify the content of a node**
   CVT-modify-value$(G, n_h, NewContent)$ modifies the old content of n_h to the new content *NewContent*. If a simple node n_k with label *NewContent* exists in G, then the operation is implemented by "moving" all the incoming edges of n_h to n_k and by modifying, if it is necessary, the temporal element of n_k. Otherwise, basic operations are used to create a new node with the suitable content, to "move" all the edges pointing to the node to be modified to the new one, and to delete the node with the old content.
7. **Remove a node**
   CVT-remove-node(G, n_r) removes the node n_r, if it exists. First, the operation removes all the ingoing edges of the node n_r through the basic operation *delete-edge*. Then, it removes also all the outgoing edges (if any) by calling the complex operation *CVT-remove-edge*, detailed in the following. Finally, it removes the node n_r by calling the basic operation *delete-node*(n_r).
8. **Remove an edge**
   CVT-remove-edge$(G, n_h, n_k, Ename)$ removes the edge labeled *Ename* between nodes n_h and n_k, if it exists. In order to avoid that the output VT PSTDM graph contains nodes that are not reachable from the root, the operation first removes the node n_k, if it is linked to the root only through paths containing the edge

Input: A PSTDM graph $G = \langle N, E, r \rangle$, the components of the edge to add, and the new valid time.
Output: The updated PSTDM graph $G = \langle N, E, r \rangle$.
Procedure
CVT-modify-edge-time($G, n_p, n_h,$ *Ename*, $NewTE$)
{
if (*user-defined semantic constraints are satisfied*)
then
if ($\ell_{\mathcal{C}}(n_h) = \perp$)
then
$G' \leftarrow$ *TE_delete-edge*$(G, n_p, n_h,$ *Ename*$)$
$G \leftarrow$ *TE_insert-edge*$(G', n_p, n_h,$ *Ename*, $NewTE$)
output(G);
endif
let $NodeTE$ be $\ell_{\mathcal{T}_{vt}}(n_p)$
if ($\ell_{\mathcal{C}}(n_h) \neq \perp$
$\wedge$ *Constraint 1 is not satisfied between NewTE and NodeTE*)
then
output(G);
else
$G' \leftarrow$ *TE_delete-edge*$(G, n_p, n_h,$ *Ename*$)$
$G \leftarrow$ *TE_insert-edge*$(G', n_p, n_h,$ *Ename*, $NewTE$)
In graph G set the valid interval of simple node n_h to be the minimal one containing all VTs of its ingoing edges
output(G);
endif
endif
endif
}

Fig. 8 The algorithm for modifying the valid time of an edge

to remove, by calling the operation *CVT-remove-node*(G, n_k). Finally, the basic operation *TE_delete-edge* is called. If n_k is a simple node and it has not to be deleted, it could be the case that the valid time of n_k must be properly modified, to be the minimal temporal element containing all the valid times of its ingoing edges (see constraint 2 for VT PSTDM graphs in Section 5.1).
It is worth noting that these two last operations, which are recursively intertwined, avoid that the output VT PSTDM graph contains nodes that are not reachable from the root, by recursively removing all the nodes and edges, which were linked to the root only through paths containing the node or the edge to be removed, respectively.

6 Managing transaction time with the PSTDM data model

Transaction time (TT) allows us to maintain the graph evolutions due to operations, such as insertion, deletion, and update, on nodes and edges. From this point of view, a TT PSTDM graph represents the changes of a (atemporal) graph, i.e., it represents in a compact way a sequence of several *atemporal graphs*, each of them obtained as result of some operations on the previous one. In this context, the *current* graph

represents the current facts in the semistructured database, and is composed by nodes and edges which have the transaction time interval ending with the special value *uc* (until changed).

Our idea is that operations on nodes and edges of a TT PSTDM graph must have as result a rooted, connected PSTDM graph. Thus, the *current* graph, composed by current nodes and edges, must be a rooted connected graph. Changes in a TT PSTDM graph are timestamped by TT in order to represent the graph history which is composed by the sequence of intermediate graphs resulting from the operations. Each operation on the graph corresponds to the suitable management of temporal labels of (possibly) several nodes and edges on the PSTDM graph.

6.1 Constraints for transaction time

The following constraints on a PSTDM graph $\langle N, E, r\rangle$ allow us to explicitly consider the append-only feature of semistructured data timestamped by an interval representing transaction time.

1. The transaction time interval of a generic edge connecting two nodes must be related to their time intervals. Intuitively, a relation between two nodes can be established and maintained only in the time interval in which both the related nodes are present in the graph.

$$\begin{aligned}\forall e_j \in E \ \Big(\big(e_j = \big\langle (n_h, n_k)\,, \{\_, [t_{js}, t_{je}]\}\big\rangle \\ \wedge\, \ell_{\mathcal{T}_{tt}}(n_h) = [t_{hs}, t_{he}] \wedge \ell_{\mathcal{T}_{tt}}(n_k) = [t_{ks}, t_{ke}]\big) \\ \rightarrow \big(t_{js} \geq \max(t_{hs}, t_{ks}) \wedge t_{je} \leq \min(t_{he}, t_{ke})\big)\Big)\end{aligned}$$

2. The time interval of each node is related to the time interval of all its ingoing edges. More particularly, for a given node but the root, the union of time intervals of all ingoing edges must be a (convex) interval. Indeed, a gap in the union of these intervals would mean that there are time points where the considered node is not reachable (which is not allowed).
   Let us assume the following function $In(n)$ where $n \in N$ is a generic node:

$$In(n) = \{[t_s, t_e] \mid \exists e \in E(e = \langle(\_, n),\ \{(\texttt{!name} : \mathit{Ename}), (\texttt{!tt} : [t_s, t_e])\}\rangle)\}$$

   The constraints can be expressed by the following formula:

$$\forall n_k \in N\left((\ell_{\mathcal{T}_{tt}}(n_k) = [t_{ks}, t_{ke}]) \rightarrow \bigcup_{[t_{js}, t_{je}] \in In(n_k)} [t_{js}, t_{je}] = [t_{ks}, t_{ke}]\right)$$

   Intuitively, a node (different from the root) can be maintained in the graph only if it is connected to at least one current complex node by means of an edge.
3. At a specific time instant, between two nodes it cannot exist more than one edge with the same name *Ename*.
   Let be

$$\begin{aligned}&eCnctd(n_1, n_2, \mathit{Ename}) \\ &\quad = \{[t_s, t_e] \mid e \in E(e = \langle(n_1, n_2), \{(\texttt{!name} : \mathit{Ename}), (\texttt{!tt} : [t_s, t_e])\}\rangle)\}\end{aligned}$$

   Note that all the considered edges have the same name *Ename*.

The constraint is represented through the following formula:

$$\forall En \in \mathcal{S}\ \ \forall n_h \in N\, \forall n_k \in N \left(\bigcap_{[t_{js}, t_{je}] \in eCnctd(n_h, n_k, En)} [t_{js}, t_{je}] = \emptyset \right)$$

This is due to the fact that an edge represents a relationship between two nodes, thus it makes no sense representing with two edges the same relationship.

6.2 Example: managing TT with PSTDM

On March 12, 2004, at 08:00 a.m., the physician stores into the database some information about the patient Ron Dalton and his symptom of chest pain. Thus, on March 12, 2004, at 08:00 a.m. the complex node *Patient* with its simple node *Name* (having as content "Ron Dalton"), and the complex node *Symptom*, with its simple node *Description* (having as content "Angina") are inserted into the PSTDM database. Moreover, the edges *Demo* between *Patient* and *Name*, *P_Situation* between *Patient* and *Symptom* and *S_Name* between *Symptom* and *Description* are inserted. It is worth noting that the time intervals of all these nodes and edges are [12/03/04;08:00, *uc*]. Figure 9 shows the PSTDM graph after these insertions.

On March 12, 2004, at 08:05 a.m., the physician stores into the database also the diagnosis of low coronary artery disease (CAD). Thus, on March 12, 2004, at 08:05 a.m., the complex node *Pathology* with its simple nodes *Name* (having as content "CAD") and *Severity* (having as content "Low") are inserted into the PSTDM database. Moreover, the relations *Diagnosis* between *Patient* and *Pathology*, *P_Name* between *Pathology* and *Name* and *P_Severity* between *Pathology* and *Severity* are inserted. The time intervals of these nodes and edges are [12/03/04;08:05, *uc*]. Figure 10 shows the PSTDM graph after these insertions.

On March 12, 2004, at 08:30 a.m., the physician stores in the database the drug prescription. Thus, on March 12, 2004, at 08:30 a.m., the complex node *Drug* with its simple node *Name* (having as content "Nitroglycerin") are inserted into the PSTDM database. Moreover, the relation *Therapy* between *Patient* and *Drug*, *D_Name*

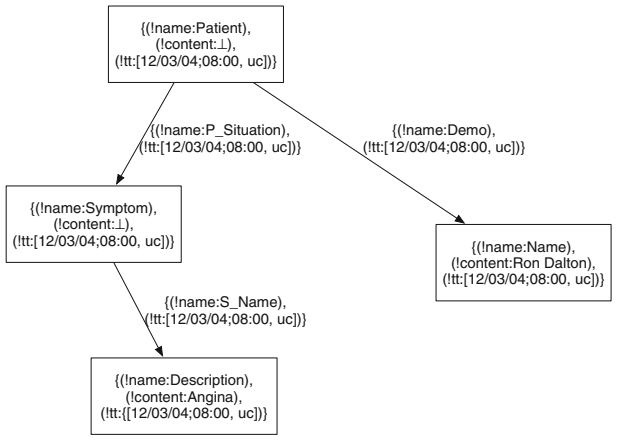

Fig. 9 The PSTDM database after data insertions on March 12, 2004 at 08:00 a.m

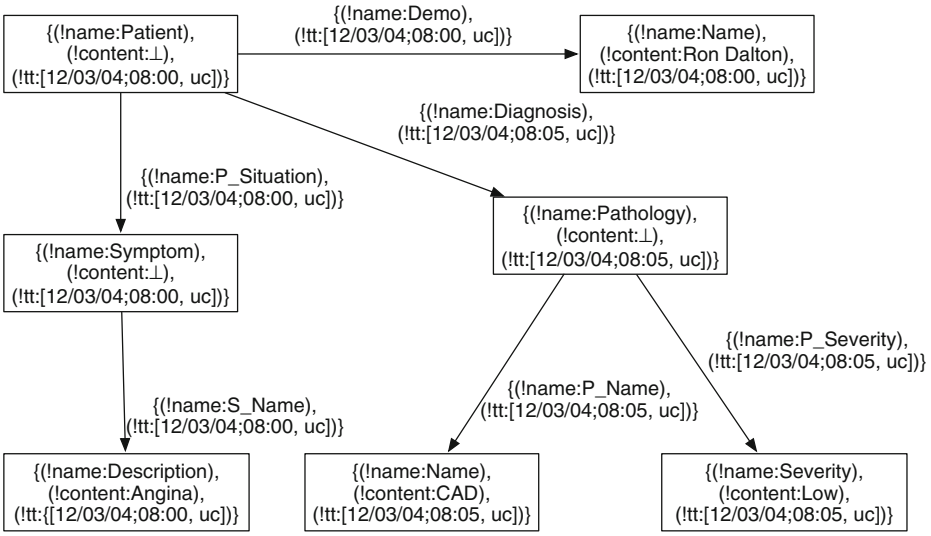

Fig. 10 The PSTDM database after data insertions on March 12, 2004 at 08:05 a.m

between *Drug* and *Name* are inserted. The time intervals of these nodes and edges are [12/03/04;08:30, *uc*]. Figure 11 shows the PSTDM graph after these insertions.

On March 21, 2004, at 10:00 a.m., the physician corrects the insertion mistake about the pathology severity, which is actually "Intermediate". Thus, on March 21, 2004, at 10:00 a.m., the simple node *Severity* is removed from the graph: its time interval changes from [12/03/04;08:05, *uc*] to [12/03/04;08:05, 21/03/04;09:59] in the PSTDM graph, and the same is for the time interval of the edge *P_Severity*.

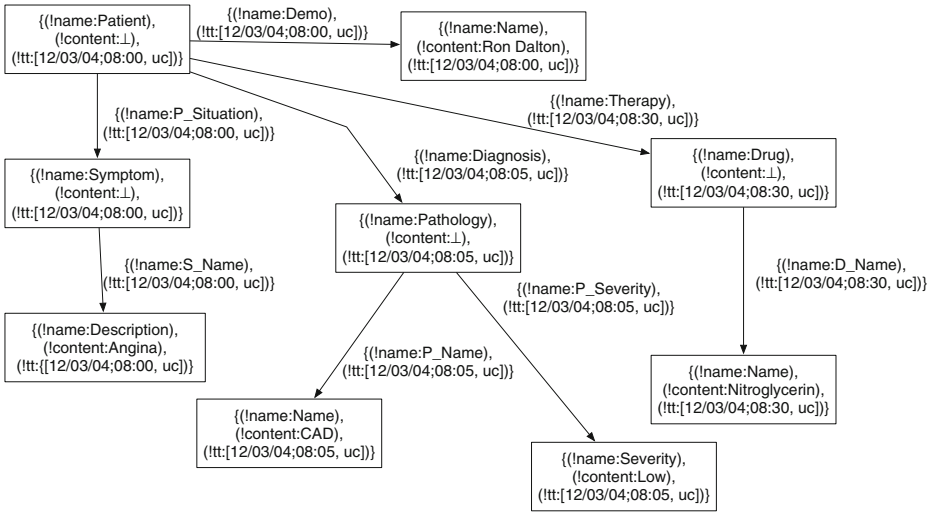

Fig. 11 The PSTDM database after data insertions on March 12, 2004 at 08:30 a.m

Moreover, the simple node *Severity* (having as content "Intermediate") and the relation *P_Severity* between *Pathology* and *Severity* are inserted into the PSTDM database. The new *Severity* and *P_Severity* time intervals are [21/03/04;10:00, *uc*]; from this moment the new simple node *Severity*, with content "Intermediate" is current. The physician does not modify the therapy. Figure 12 represents the PSTDM database resulting from these last operations.

6.3 Operations on PSTDM graphs for managing transaction time

Now we introduce basic and complex operations to deal with transaction time. In particular, we point out the main differences with the operations introduced in Section 5.3 for the valid time dimension.

When managing transaction time, the time interval of nodes and edges of a PSTDM graph is system-generated. In the following, each introduced complex operation on TT PSTDM graphs will be identified by the prefix *CTT-* (Complex Transaction Time). Moreover, basic operations differs from those introduced for VT only for the domain of the temporal property (i.e. `!tt`), that is the set of time intervals; thus, we explicitly introduce only the operation for inserting a node, for the others we remind their name. Since the domain of the temporal dimension is the set of time intervals, basic operations will be identified by the prefix *I_*.

Let us start with the basic operations, directly managed by the system.

1. **Insert a node**
   I_insert-node(*G*, *Nname*, *Ncontent*, $[t_s, t_e]$, *rootflag*) inserts in a graph *G* a node with label {(`!name` : *Nname*), (`!content` : *Ncontent*), (`!tt` : $[t_s, t_e]$)} and gives as result the identifier n_a assigned to the added node together with the resulting graph, as shown by the pseudocode in Fig. 13. If the boolean variable *rootflag* has value true, the added node is the root of the resulting graph.

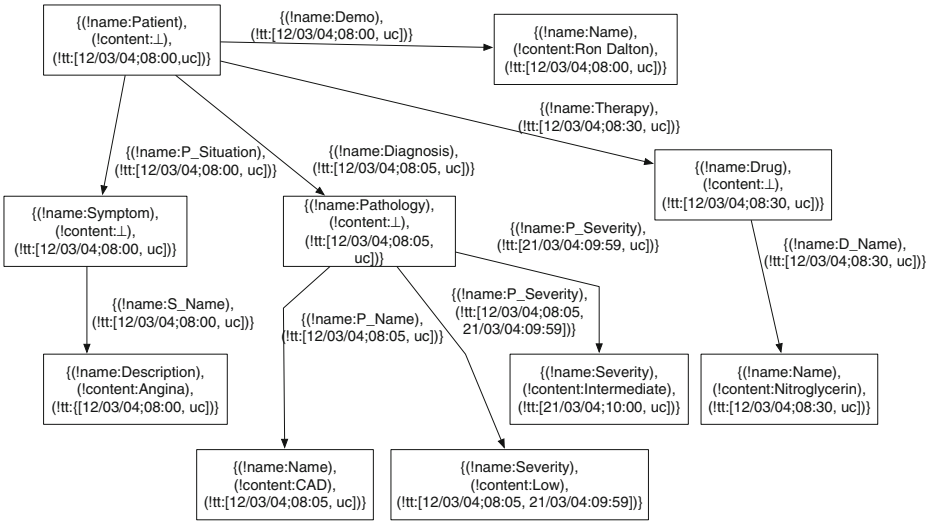

Fig. 12 The PSTDM database after the mistake correction on March 21, 2004 at 10:00 a.m

2. **Insert an edge**
   I_insert-edge$(G, n_{from}, n_{to}, Ename, [t_s, t_e])$ inserts the edge $e_j = \langle (n_{from}, n_{to}), \{(!\texttt{name} : Ename), (!\texttt{tt} : [t_s, t_e])\}\rangle$ between the nodes n_{from} and n_{to} in the graph G, and returns the resulting graph.
3. **Delete a node**
   I_delete-node(G, n_d) removes the node n_d from the graph G.
4. **Delete an edge**
   I_delete-edge$(G, n_{from}, n_{to}, Ename, [t_s, t_e])$ removes from the graph G the edge between nodes n_{from} and n_{to}, labeled *Ename* and having as value for the temporal property the interval $[t_s, t_e]$.

Let us now consider complex operations, built on top of the basic ones.

1. **Add the root node**
   CTT-add-root-node$(Nname, Ncontent)$ at time t_a creates a new empty graph $G = \langle \emptyset, \emptyset, \perp \rangle$ and calls the basic operation *I_insert-node*$(G, Nname, Ncontent, [t_a, uc]$, true). This operation returns the new graph G and the node identifier of the added root.
   We suppose the user chooses this operation as the first one to build a new PSTDM graph; any node added to an existing PSTDM graph in a second moment by means of a *CTT-add-node* operation will be directly, or indirectly, reachable from the root node.
2. **Add a node**
   CTT-add-node$(G, Nname, Ncontent, n_p, Ename)$ at time t_a adds to the graph G a new node having the node n_p as parent and connected to it by an edge named *Ename*. The operation is implemented by means of the algorithm reported in Fig. 14. The first **if** construct verifies whether the parent node n_p is current (line 1). If the parent node is not current, the algorithm does not modify the input graph, otherwise two main cases are considered.

Input: A PSTDM graph $G = \langle N, E, r\rangle$,
the components of a node label *Nname*, *Ncontent*, $[t_s, t_e]$,
and the value of a boolean flag *rootflag*.
Output: A graph G' and the identifier n_a of the added node.
Procedure I_insert-node (G, *Nname*, *Ncontent*, $[t_s, t_e]$, *rootflag*)
{
generate a new node identifier n_a *such that* $n_a \notin N$
if (the flag *rootflag* is set to false)
then
$r' \leftarrow r$
else
$r' \leftarrow n_a$
endif
$G' \leftarrow \langle N \cup \{n_a \mid \ell(n_a) = \{(!\texttt{name} : Nname), (!\texttt{content} : Ncontent), (!\texttt{tt} : [t_s, t_e])\}\}, E, r'\rangle$
output$(\langle G', n_a\rangle)$
}

Fig. 13 The algorithm for the operation *I_insert-node*

Input: A PSTDM graph $G = \langle N, E, r\rangle$, the labels of the node to add, the node identifier of its parent node, and the edge connecting it to the parent node.
Output: The updated PSTDM graph $G = \langle N, E, r\rangle$, and the node identifier n_a of the added node.
Procedure CTT-add-node (G, *Nname*, *Ncontent*, n_p, *Ename*)
{
1 **if** (the node n_p is not current)
2 **then**
3 **output**($\langle G, \perp\rangle$);
4 **else**
5 **if** (*Ncontent* $=\perp$)
6 **then**
7 $\langle G', n_a\rangle \leftarrow$ *I_insert-node*(G, *Nname*,*Ncontent*, $[t_a, uc]$, false)
8 $G \leftarrow$ *I_insert-edge*(G', n_p, n_a, *Ename*, $[t_a, uc]$)
9 **output**($\langle G, n_a\rangle$);
10 **endif**
11 **if** (*Ncontent* $\neq\perp$)
12 **then**
13 $G_{last} \leftarrow G$
14 **if** ($\exists n_h \in N(\ell(n_h) =$ {(!name : *Nname*), (!content : $Ncontent_h$),
15 (!tt : $[t_{hs}, uc]$)}$\wedge$
16 $\exists\langle(n_p, n_h)$, {(!name : *Ename*), (!tt : $[t_{es}, uc]$)}$\rangle \in E\wedge$
17 $Ncontent_h \neq$ *Ncontent*)
18 **then**
19 **if** ({$\langle(n_i, n_h)$, {(!name : *Ename*), (!tt : $[t_{si}, uc]$)}$\rangle$ |
20 $(n_i \neq n_p) \vee (n_i = n_p \wedge Ename_i \neq Ename)$} $\neq \emptyset$)
21 **then**
22 $G_1 \leftarrow$ *I_insert-edge*(G, n_p, n_h, *Ename*, $[t_{es}, t_a - 1]$)
23 $G_{last} \leftarrow$ *I_delete-edge*(G_1, n_p, n_h, *Ename*, $[t_{es}, uc]$)
24 **else**
25 $\langle G_1, n'_h\rangle \leftarrow$ *I_insert-node*(G, *Nname*, $Ncontent_h$, $[t_{hs}, t_a - 1]$, false)
26 $G_2 \leftarrow$ *I_insert-edge*(G_1, n_p, n'_h, *Ename*, $[t_{es}, t_a - 1]$)
27 $G_3 \leftarrow$ *I_delete-edge*(G_2, n_p, n_h, *Ename*, $[t_{es}, uc]$)
28 $G_{last} \leftarrow$ *I_delete-node*(G_3, n_h)
29 **endif**
30 **else**
31 **if** $Ncontent_h =$ *Ncontent*
32 **then**
33 **output**($\langle G, \perp\rangle$);
34 **endif**
35 **endif**
36 $\langle G_r, n_a\rangle \leftarrow$ *I_insert-node*(G_{last}, *Nname*, *Ncontent*, $[t_a, uc]$, false)
37 $G \leftarrow$ *I_insert-edge*(G_r, n_p, n_a, *Ename*, $[t_a, uc]$)
38 **output**($\langle G, n_a\rangle$);
39 **endif**
40 **endif**
}

Fig. 14 The algorithm for adding a node at time t_a in a transaction time PSTDM graph

(a) For adding a complex node, the two basic operations, *I_insert-node* and *I_insert- edge* are called (lines 7 and 8). This complex operation does not violate any constraint; in particular, constraint 1 of TT imposing that the time interval of an edge is related to the time intervals of the two connected nodes is satisfied, because the added edge has the same time interval of the added node, and the time interval of this node starts after the time interval of its parent n_p. For the same reason, constraint 2 of TT imposing that the

time interval of each node is related to the time interval of all its ingoing edges is satisfied.

(b) For adding a simple node, the operation first checks whether there is another current simple node n_h with the same name and the same parent n_p, and a different content. In this case there are two possible situations: the node n_p is not the unique complex node pointing to n_h (line 19), thus the algorithm changes the time interval of the edge connecting n_p to n_h (lines 22 and 23). Otherwise, if the node n_p is the unique complex node pointing to n_h (lines 18 and 19), the algorithm changes the time interval of n_h (lines from 25 to 28), in order to avoid the possibility of storing for a given node two properties with the same label at the same time (as specified by constraint 3 of TT). As it is shown in the pseudocode of the algorithm, updating time intervals of nodes and edges consists of inserting and deleting suitable nodes and edges. As as example, in Fig. 14 lines 25 and 28 substitute a current node with a corresponding non-current one, having the time interval ending at the insertion time $t_a - 1$. At the end, the complex operation calls the basic ones *I_insert-node* and *I_insert-edge* to add the new simple node to the PSTDM graph.

3. **Add an edge**
CTT-add-edge$(G, n_{from}, n_{to}, EName)$ at time t_a first checks whether the nodes n_{from} and n_{to} to be connected, are current. When at least one of the two nodes is not current, the operation fails and returns the boolean value false. Moreover, when there is already a current edge *Ename* between the two considered nodes, then this operation fails and returns the boolean value false, as it is not meaningful to have two nodes connected by two edges with the same name at the same time (see also constraint 4 for transaction time PSTDM graphs). In the other cases, *CTT-add-edge* calls the basic operation *I_insert-edge*$(G, n_{from}, n_{to}, Ename, [t_a, uc])$.
4. **Remove a node**
CTT-remove-node(G, n_r) at time t_r is implemented according to the following criteria: if n_r is not current, the operation fails and returns the boolean value false; otherwise, it logically removes the node n_r, its ingoing current edges, and its outgoing current edges. Removing logically nodes and ingoing edges is performed by basic operations *I_insert-node* and *I_delete-node*, *I_insert-edge* and *I_delete-edge*, respectively. Outgoing current edges are removed by suitable calls of the operation *CTT-remove-edge* detailed the following. As for the non-current ingoing and outgoing edges of the removed node, they modified to point to/from the new non-current node inserted instead of n_r.
5. **Remove an edge**
CTT-remove-edge$(G, n_h, n_k, Ename)$ at time t_r checks whether there is a current edge labeled *Ename* between the nodes n_h and n_k. If not, the operation fails, otherwise it calls the basic operations *I_insert-edge*$(G, n_h, n_k, Ename, [t_{es}, t_r - 1])$ and *I_delete-edge*$(G, n_h, n_k, Ename, [t_{es}, uc])$. At the end, if the node n_k has no ingoing current edges, the complex operation *CTT-remove-node*(n_k) is called. The intertwined and recursive use of the operations *CTT-remove-node* and *CTT-remove-edge* guarantee that the obtained PSTDM graph is a rooted, connected graph, the operation *CTT-remove-node* is recursively applied on each

current child of the removed node n_r, which has no current ingoing edges (see also constraints 1 and 2 for TT PSTDM graphs).

6. **Modify the name of a node**
   CTT-modify-name-node(G, n_h, *NewName*) is used to modify the name of a node at time t_m, if it is current, otherwise the operation fails and returns false. This operation is realized by calling the basic operation *insert-node*, which adds to G at time t_m the new (complex or simple) node n_k with name *NewName*, without any ingoing edge, with the same content and type of the modified node n_h. For each ingoing and outgoing (if any) current edge connected to n_h, a corresponding edge is added at time t_m, connected to n_k instead of to n_h. For example, for each current edge $\langle(n_i, n_h), \{(!\texttt{name} : \mathit{Ename}), (!\texttt{tt} : [t_{es}, uc])\}\rangle$, the corresponding edge $\langle(n_i, n_k), \{(!\texttt{name} : \mathit{Ename}), (!\texttt{tt} : [t_m, uc])\}\rangle$ is added.
   At the end, the operation calls *CTT-remove-node*(n_h) which removes the modified node and its ingoing and outgoing edges. The operation does not remove the nodes originally connected to the node n_h because they are now related to the new (added) node. This operation is composed by a set of basic and complex operations and thus we suppose that the transaction time is the same for all the composing operations, and that if there is at least one violation (in one of the composing operations) the operation *CTT-modify-name-node* fails and does not change the graph G.
   The user employs this operation to modify the name of a node; actually the result of the operation is a graph containing the old node (which becomes *non current*) and the new node (with the new name). The new node is not related to the old one, because we do not represent their relationship and do not maintain the relation between their identifiers.
7. **Modify the content of a node**
   CTT-modify-content-node(G, n_m, *NewContent*) modifies the content of a node at time t_m, if it is current, otherwise the operation fails. The behavior of this operation is similar to that of the previous *CTT-modify-name-node*, but it works on the content, instead of the name of the node.
8. **Modify the name of an edge**
   CTT-modify-name-edge(G, n_h, n_k, *OldEname*, *NewEname*) modifies the name of an edge at time t_m, if the edge is current, otherwise the operation fails. This operation first checks if there is another current edge labeled *NewEname* between n_h and n_k and in this case fails, to avoid the presence of two distinct current edges with the same name (i.e., *NewEname*) between two nodes (see constraint 3 for TT PSTDM graphs). In the other cases, it calls the basic operation *I_insert-edge*(G, n_h, n_k, *NewEName*, $[t_m, uc]$), which inserts at time t_m the modified edge, and the complex operation *CTT-remove-edge*(n_h, n_k, *OldEname*), which removes at time t_m the old edge $\langle(n_h, n_k), \{(!\texttt{name} : \mathit{OldEname}), (!\texttt{tt} : [t_{es}, uc])\}\rangle$.

We assume to represent PSTDM graphs by means of unordered sets of nodes and edges, and thus the algorithms we propose to define the operations are not the most efficient ones. For example, when dealing with transaction time, the rough complexity of the algorithm of Fig. 14 for adding a node to a PSTDM graph is $O(n_N \times n_E)$ (with n_N the number of nodes and n_E the number of edges in the graph) because the procedure has to consider each node in the set and for each node it has

to evaluate each edge of the graph (see lines 14 and 15). The algorithm complexities can be reduced by using ad-hoc data structures and algorithms (Cormen et al. 2001).

7 Extending PSTDM to manage valid and transaction times together

Bitemporal models are more expressive than temporal ones because they consider the capability of representing when the facts were valid in the considered reality, as well as when the facts were current in the database. Even though valid and transaction times are two orthogonal time dimensions (Jensen and Snodgrass 1999), i.e., they can be assigned independently one from each other, the presence of both the temporal dimensions for stored data involves more complex (and intertwined) constraints with respect to the simple "merge" of temporal constraints related to valid and transaction times, separately. In other words, the temporal consistency of information stored into the semistructured database can be verified only by considering together valid and transaction times: indeed, we are representing into the same semistructured database a collection of real-world histories, each of them identified by a given transaction time. The overall consistency of this collection of histories can be guaranteed only if we take into account also transaction time, when checking the real-world consistency of the collection and, on the other side, only if we explicitly consider the valid time dimension, when checking the database consistency of the collection.

In order to manage valid time and transaction times together, we need to extend a PSTDM graph, which becomes *bitemporal*, by representing two temporal properties; indeed a node n_i will have a label $\ell(n_i) = \{(!\texttt{name} : Nname_i), (!\texttt{content} : Ncontent_i), (!\texttt{vt} : Nvtime_i), (!\texttt{tt} : Nttime_i)\}$, and an edge $e_j = \langle (n_h, n_k), \{(!\texttt{name} : Ename_j), (!\texttt{vt} : Evtime_j), (!\texttt{tt} : Ettime_j)\}\rangle$. Thus, in a *bitemporal* (BT) PSTDM graph the two temporal dimensions are explicitly reported both on node and edge labels and are described by means of a temporal element and a closed interval, respectively.

In this section we define the set of constraints needed to support in a correct way the semantics of the considered time dimensions together, and then the operations to manipulate the bitemporal PSTDM graphs. For each operation we highlight the constraints that have to be checked in order to guarantee the time-related correctness of the operation itself.

In order to apply operations on bitemporal graphs we introduce the notion of *snapshot* for a bitemporal PSTDM graph, with respect to the transaction time.

Definition 1 Given a time instant t and a bitemporal PSTDM graph $G = \langle N, E, r\rangle$, the *snapshot* of G at time t (named S_t) is the bitemporal PSTDM graph $G' = \langle N', E', r'\rangle$ such that, $\forall n_i \in N'(t \in \ell_{\mathcal{T}_{tt}}(n_i))$ and $\forall e_j = \langle (n_h, n_k), \{(!\texttt{name} : Ename_j), (!\texttt{vt} : Evtime_j), (!\texttt{tt} : Ettime_j)\}\rangle \in E'(t \in Ettime_j)$.

Informally, given a time instant t, the snapshot S_t is the bitemporal PSTDM subgraph G' of G with all the nodes and edges containing t in the transaction time interval.

7.1 Constraints for valid and transaction times together

In the previous subsections temporal dimensions were not taken into account together and thus the set of constraints was related to the single modeled temporal dimension. In this subsection we deal with constraints, which must consider both valid and transaction times together. As for *database consistency*, we have to guarantee that each snapshot of a bitemporal PSTDM graph is still a bitemporal PSTDM graph, i.e., a connected, rooted graph, as outlined in the Definition 1. The first and second constraints defined for the PSTDM model with respect to the transaction time dimension must hold in the bitemporal context as well:

1. the TT time interval of an edge cannot start before neither end after the TT intervals of the connected nodes, and
2. the TT interval of each node is related to the TT intervals of all its ingoing edges. However, when considering the last two constraints with respect to the transaction time, we have to properly manage also the *real-world consistency*, by modifying the constraints as in the following.
3. At a given transaction time, for a complex node it is not possible to simultaneously store more than one valid property, i.e., more edges, connected to simple nodes, with the same name label and valid time intervals having an intersection.

$$\begin{aligned}
&\forall Ename\ \forall Nname\ \forall n_h \in N\ \forall n_k \in N\ \forall n_l \in N\ \forall e_j \in E\ \forall e_i \\
&\in E\ ((e_j = \langle (n_h, n_k), \{(!\texttt{name} : Ename), (!\texttt{vt} : Evtime_j), \\
&(!\texttt{tt} : [tt_{js}, tt_{je}])\}\rangle \\
&\wedge e_i = \langle (n_h, n_l), \{(!\texttt{name} : Ename), (!\texttt{vt} : Evtime_i), \\
&(!\texttt{tt} : [tt_{is}, tt_{ie}])\}\rangle \\
&\wedge n_k \neq n_l \wedge \ell_{\mathcal{N}}(n_k) = Nname \wedge \ell_{\mathcal{N}}(n_l) = Nname) \\
&\rightarrow ((Evtime_j \cap Evtime_i \neq \emptyset) \rightarrow ([tt_{js}, tt_{je}] \cap [tt_{is}, tt_{ie}] = \emptyset)))
\end{aligned}$$

4. At a given transaction time, between two nodes it cannot exist more than one edge with the same name *Ename*, valid at a time instant.

$$\begin{aligned}
&\forall Ename,\ \forall n_h \in N\ \forall n_k \in N\ \forall e_j \in E\ \forall e_i \\
&\in E\ ((e_j = \langle (n_h, n_k), \{(!\texttt{name} : Ename), (!\texttt{vt} : Evtime_j), \\
&(!\texttt{tt} : [tt_{js}, tt_{je}])\}\rangle \\
&\wedge\ e_i = \langle (n_h, n_l), \{(!\texttt{name} : Ename), (!\texttt{vt} : Evtime_i), \\
&(!\texttt{tt} : [tt_{is}, tt_{ie}])\}\rangle \\
&\rightarrow\ ((Evtime_j \cap Evtime_i \neq \emptyset) \rightarrow ([tt_{js}, tt_{je}] \cap [tt_{is}, tt_{ie}] = \emptyset)))
\end{aligned}$$

In general, for the *real-world consistency* we have also to guarantee that each real-world history is consistent with the data modeling choices adopted in describing real-world concepts and objects: for all time instants t, the snapshot S_t must satisfy

the general constraints defined for the PSTDM model with respect to valid time dimension (i.e., the non optional constraints from 1 to 4 defined in Section 5.1). Moreover, as discussed in the previous section, it is possible to define other optional constraints for imposing restrictions related to the (domain dependent) semantics of the represented information.

7.2 Example: managing VT and TT together with PSTDM

On January 10, 2004, at 8:00 a.m., the physician visits for the first time Ron Dalton who becomes, from this moment, his patient and on March 12, 2004, at 08:00 a.m., the physician stores into the database his information (but reporting the surname as "Dallton") and the symptom the patient reports. In Fig. 15 we show the bitemporal PSTDM graph representing the stored data.

On March 12, 2004, at 08:02 a.m., the physician immediately corrects the mistake in the patient surname. In Fig. 16 we report the mistake correction: the transaction time interval of the old name property change from [12/03/04;08:00, *uc*] to [12/03/04;08:00,12/03/04;08:01], while the new name property has the transaction time interval equal to [12/03/04;08:02, *uc*]. Note that the valid time interval is the same for the two nodes (but their transaction time intervals are disjoint).

On February 2, 2004, at 08:00 a.m., the physician diagnoses the correct pathology named CAD (Coronary Artery Disease) with a low severity. On March 12, 2004, at 08:05 a.m., the physician stores into the database this new information. Figure 17 shows the bitemporal PSTDM graph after these insertions.

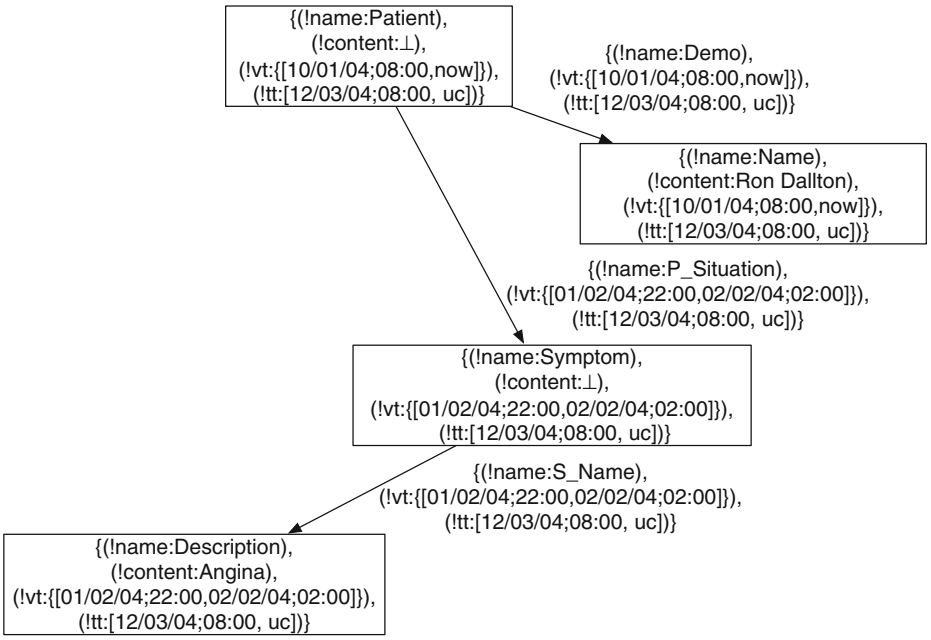

Fig. 15 The bitemporal PSTDM database after data insertions on March 12, 2004 at 08:00 a.m

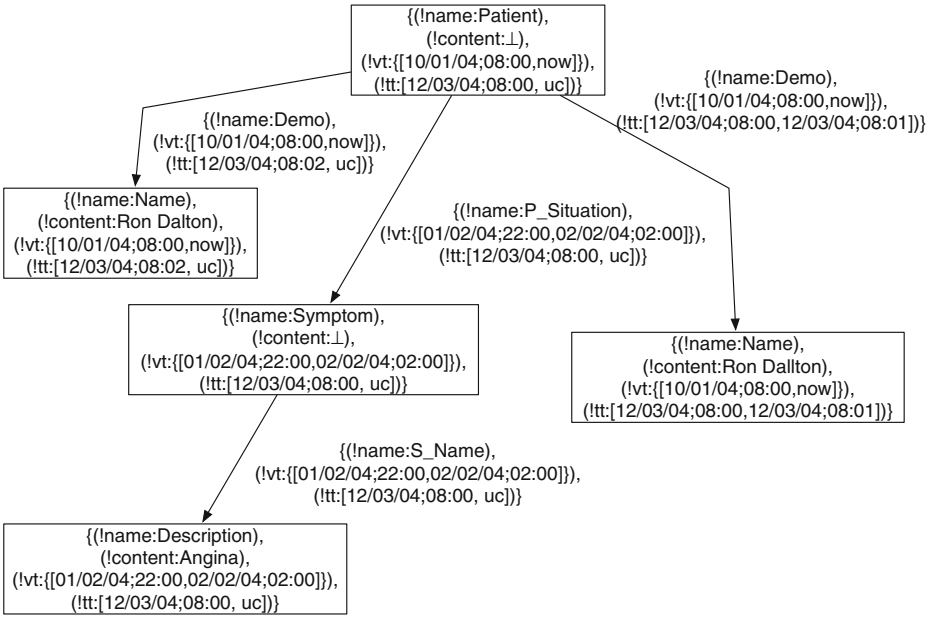

Fig. 16 The bitemporal PSTDM database after the correction of a mistake

On February 2, 2004, at 08:05 a.m., the physician prescribes the therapy to the patient advising to have Nitroglycerin, which we assume is at disposal since May 3, 2001 at 08:00 a.m. On March 12, 2004, at 08:30 a.m., the physician stores in the database the drug prescription. On February 10, 2004, at 11:00 a.m., the pathology of Ron Dalton

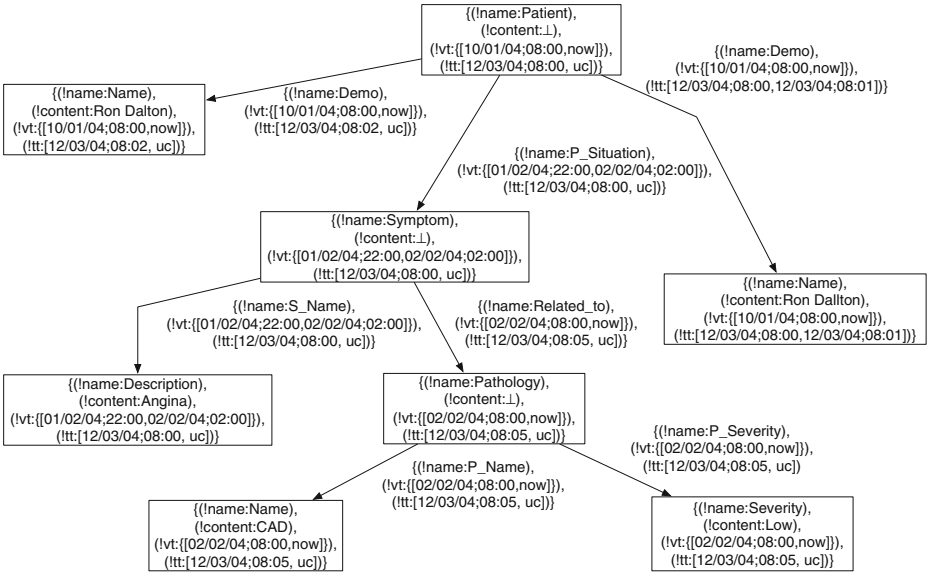

Fig. 17 The bitemporal PSTDM database after new data insertions on March 12, 2004 at 08:05 a.m

gets worse and the severity changes from low to intermediate. On March 21, 2004, at 10:00 a.m., the physician stores the evolution about the pathology severity, which is currently "Intermediate". In Fig. 18 we represent the bitemporal PSTDM database resulting from these last operations: the severity property with "Low" value has been removed (its transaction time interval ends at [21/03/04;09:59]), the other severity property with "Low" value has been added to highlight that this information was actually valid in the time interval [02/02/04;08:00,10/02/04;10:59]. The current severity property with "Intermediate" value has been inserted. Moreover, the complex node *Drug*, with valid time [03/05/01;08:00, *now*] and the edge *Therapy* with valid time interval [02/02/04;08:05, *now*] and transaction time interval [12/03/04;08:30, *uc*] have been inserted.

7.3 An incremental approach to build bitemporal PSTDM graphs

As we have discussed for transaction and valid times, managing bitemporal PSTDM graphs also requires the definition of operations to insert nodes (either complex or simple), insert edges, remove nodes, and so on, and each operation includes the checks of the previously defined constraints, needed to efficiently guarantee the consistency of bitemporal PSTDM graphs after each update operation. In this context the user provides only the temporal element for the valid time property, while the transaction time interval is system-generated.

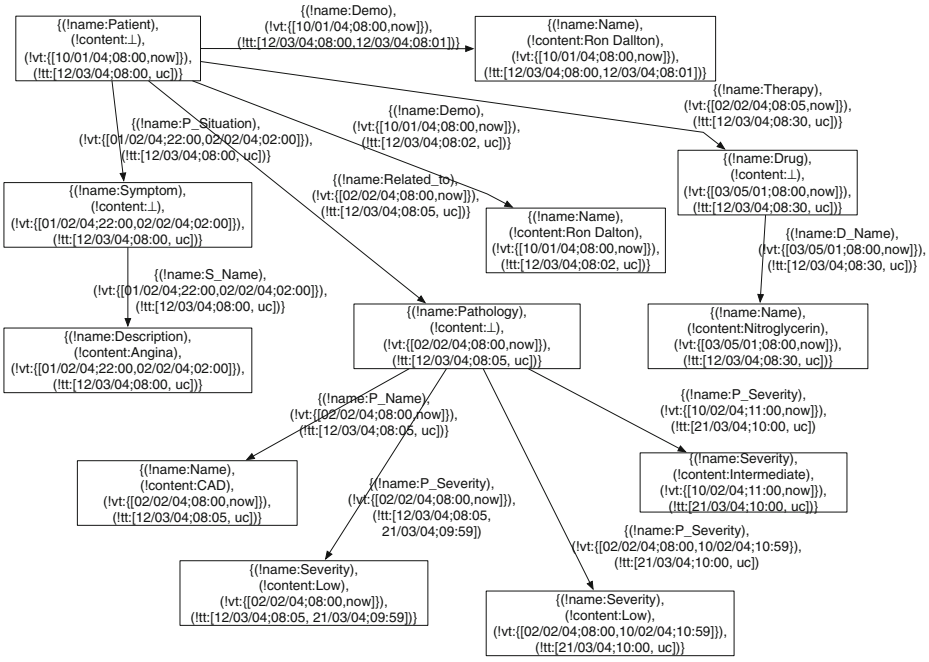

Fig. 18 The resulting bitemporal PSTDM database

As an example, in this section we report only the procedural definition and the algorithm of the complex operation required to remove a node, using in their definition a slightly extended version of basic operations (having names starting with "*B_*") to manage two temporal dimensions. Indeed, PSTDM graphs managing both temporal dimensions have the *vt* and *tt* property labels. The other complex operations are similar to the ones defined for the other time dimensions.

1. **Remove a node**
   CBT-remove-node(G, n_r) removes at time t_r the node n_r, if it is current and if the user-defined, VT-related constraints will be satisfied in the resulting BT PSTDM graph; otherwise the operation fails.
   Figure 19 shows the algorithm realizing this operation. This operation has mainly to modify the transaction time of the node to be removed, through the basic operations that we will call *B_insert-node* and *B_delete-node*; afterwards, the operation has to properly manage all the ingoing and outgoing edges of the given nodes: all the current edges have to be removed, by calling the operation *CBT-remove-edge* for all the outgoing edges (see line 24 in Fig. 19) and the basic operations *B_insert-edge* and *B_delete-edge* for the ingoing current edges (see lines 8 and 9 in Fig. 19). As for the non current edges, they have only to be updated to connect the modified (i.e., no more current) node (see lines 11, 12 and 26, 27 in Fig. 19).
2. **Remove an edge**
   CBT-remove-edge$(n_h, n_k, Ename, VTime)$ removes at time t_r the given edge, if it is current and the user-defined VT-related constraints will be satisfied in the resulting BT PSTDM graph; otherwise the operation fails. In order to avoid that the output PSTDM graph contains nodes that at some (transaction) times are not reachable from the root, the operation first removes the node n_k, if it is currently linked to the root only through paths containing the edge to remove, by calling the operation *CBT-remove-node*. Finally, the basic operations *B_insert-edge* and *B_delete-edge* are called, to properly modify the transaction time of the considered edge. If n_k is a simple node and it has not to be removed, it could be the case that the valid time of n_k must be properly modified, to be the minimal interval containing all the valid times of its ingoing current edges (see constraint 2 for VT PSTDM graphs in Section 5.1): in case of bitemporal PSTDM graphs, it means that the node n_k is first copied in another node n'_k, which is the same of n_k but the transaction time, ending at time $t_r - 1$. Another copy n''_k is then created which is the same of n_k but the transaction time, which is $[t_r, uc]$ and the valid time, which is the minimal one containing the valid time of all the current edges connected to n_k; before deleting n_k, all the remaining current edges connected to n_k are modified and copied to represent connections to n'_k up to (transaction) time $t_r - 1$ and connections to the node n''_k since time t_r. It is worth noting that these two operations, which are recursively intertwined, avoid that the output PSTDM graph contains at any transaction time nodes that are not reachable from the root, by recursively removing all the nodes and edges, which were linked to the root only through paths containing the node or the edge to be removed, respectively. Moreover, these operations guarantee that the valid time of simple nodes are always consistent.

Input: A Bitemporal PSTDM graph $G = \langle N, E, r\rangle$, the node identifier n_r
Output: The updated PSTDM graph $G = \langle N, E, r\rangle$.
Procedure BT-remove-node(G, n_r)
{
1 **if** ($\ell(n_r) = \{$(!name : *Nname*), (!content : *Ncontent*), (!vt : *Nvtime*), (!tt : $[tt_s, uc)$)$\}\wedge$
user-defined semantic constraints are satisfied
2 **then**
3 $\langle G_{temp}, n'_r\rangle \leftarrow$ *B_insert-node*(G, *Nname*, *Ncontent*, *Nvtime*, $[tt_s, t_r - 1]$)
4 **for each** *ingoing edge* $\langle(n_i, n_r)$, {(!name : *Ename*), (!vt : *Evtime*), (!tt : $[tt_{se}, tt_{ee}]$)}$\rangle \in G$
5 **do**
6 **if** ($tt_{ee} = uc$)
7 **then**
8 $G' \leftarrow$ *B_insert-edge*(G_{temp}, n_i, n'_r, *Ename*, *Evtime*, $[tt_{se}, t_r - 1]$)
9 $G_{temp} \leftarrow$ *B_delete-edge*(G', n_i, n_r, *Ename*, *Evtime*, $[tt_{se}, uc]$)
10 **else**
11 $G' \leftarrow$ *B_insert-edge*(G_{temp}, n_i, n'_r, *Ename*, *Evtime*, $[tt_{se}, tt_{ee}]$)
12 $G_{temp} \leftarrow$ *B_delete-edge*(G', n_i, n_r, *Ename*, *Evtime*, $[tt_{se}, tt_{ee}]$)
13 **endif**
14 **endfor**
15 **if** ($Ncontent \neq \perp$)
16 **then**
17 $G \leftarrow$ *B_delete-node*(G_{temp}, n_r)
18 **output**(G);
19 **else**
20 **for each** *outgoing edge* $\langle(n_r, n_j)$, {(!name : *Ename*), (!vt : *Evtime*), (!tt : $[tt_{se}, tt_{ee}]$)}$\rangle \in G$
21 **do**
22 **if** ($tt_{ee} = uc$)
23 **then**
24 $G_{temp} \leftarrow$ *BT-remove-edge*(G_{temp}, n_r, n_j, *Ename*, *Evtime*)
25 **else**
26 $G' \leftarrow$ *B_insert-edge*(G_{temp}, n'_r, n_j, *Ename*, *Evtime*, $[tt_{se}, tt_{ee}]$)
27 $G_{temp} \leftarrow$ *B_delete-edge*(G', n_r, n_j, *Ename*, *Evtime*, $[tt_{se}, tt_{ee}]$)
28 **endif**
29 **endfor**
30 $G \leftarrow$ *delete-node*(G_{temp}, n_r)
31 **output**(G);
32 **endif**
33 **else**
34 **output**(G);
35 **endif**
}

Fig. 19 The algorithm for removing at time t_r a node

8 Discussion

In our proposal, we use rooted, connected, directed, acyclic, and labeled graphs to represented semistructured temporal information. We avoid cyclic graphs to simplify the evaluation of constraints in the graphical matching. It means that relations between objects, which correspond to PSTDM edges, have to be suitably represented in some unidirectional way, i.e., without cycles.

In this work, we consider general semistructured data, such as, for example, data coming from heterogeneous data sources, instead of XML data; thus we do not deal with ordered and mixed nodes, that are a main feature of XML documents. Our approach is database-oriented, and thus we do not propose a model able to represent ordered nodes, but the defined node and edge labels can be easily extended to represent this property as well. At this aim, we could extend either the nodes labels, or the edges labels. In the former case we could represent absolute order between nodes with respect to their position in the graph, while in the latter case we could represent the relative order of nodes (children) with respect to the common node representing their parent.

To the best of our knowledge, the only other work which explicitly addresses the issue of time-related semantics for general semistructured data is Dyreson et al. (1999). As already discussed in the previous section, in Dyreson et al. (1999), the authors propose a framework for semistructured data, where graphs are composed by nodes and labeled edges representing different properties. The focus of that work is on the definition of suitable operators (i.e., collapse, match, coalesce, and slice), which allow one to determine the (different) semantics of properties for managing queries on such graphs. As for the temporal aspects, even though in Dyreson et al. (1999) some examples are provided about special semantics for update to accommodate the transaction time, a detailed and complete examination of all the constraints for modeling either transaction or valid times is missing and is outside the main goal of that work. Moreover, the authors claim that they "leave open the issue of how these constraints are enforced on update" (Dyreson et al. 1999). With respect to the proposal described in Dyreson et al. (1999), we thus explicitly focus on the semantics of temporal aspects and do not consider the semantics of other properties; this way, even though we are less general than the authors in Dyreson et al. (1999), we are able to provide a complete treatment of the constraints when representing either transaction or valid times, facing some important aspects which have not been completely considered in Dyreson et al. (1999), such as, for example, the problem of the presence of nodes/subgraphs which could become unreachable from the root of the graph after some updates. Moreover, another novel feature of our work is that we explicitly address the issue of providing users with powerful operators for building PSTDM graphs consistent with the given temporal semantics. Summarizing, with respect to Dyreson et al. (1999), the original aspects of our proposal are: the definition of a more compact data model for semistructured data, by labeling both nodes and edges and by using temporal elements for valid times; the constraint-based specification of the semantics of temporal semistructured databases supporting valid time, transaction time, and both valid and transaction times, respectively; the proposal of operations and of related algorithms allowing the user to build consistent temporal semistructured databases according to the supported temporal dimension(s).

Other proposals (Amagasa et al. 2000; Campo and Vaisman 2006; Rizzolo and Vaisman 2008; Wang and Zaniolo 2002, 2003) consider issues related to the time in the XML context. In particular, in Campo and Vaisman (2006), Rizzolo and Vaisman (2008) the authors also deal with the problem of tackling consistency in temporal XML documents. XML documents are semistructured in nature, but present different features to deal with. XML elements, in an XML document, are related by the containment relationship, thus temporal data models for XML documents do not allow one to represent general relationships (different from the containment)

between XML objects. Moreover, in their graphical representation they use trees instead of graphs. Even if we focus on general semistructured data instead of XML data, we compare our proposal with Amagasa et al. (2000) and Rizzolo and Vaisman (2008).

In Amagasa et al. (2000) the authors propose a logical data model for representing histories of XML documents. The proposed model extends the XPath data model, by enabling edges to have a label that represents their valid time. Moreover, the authors propose a set of operations to modify the represented XML documents, by extending the original DOM API (World Wide Web Consortium 2000). The model proposed in Amagasa et al. (2000) considers the valid time dimension, and is based on trees composed by four kinds of nodes and labeled edges. The kinds of nodes are: *root*, *element*, *text*, and *attribute* nodes. Edges between nodes represent the containment relationship, and their labels represent the valid time interval. To manage in a correct way the valid time dimension, the authors define the notion of *consistent temporal XML document*. At this aim, they impose the following two conditions: (i) the union of the valid time intervals of the children of a given node must be contained in the valid time interval of the node itself (parent), and (ii) the intersection of the valid time intervals of the children of the root node must be empty. These constraints do not cover issues possibly related to the semantics of representing objects and relationships, as those we describe for our model in Section 5.1. This is due to the fact the model described in Amagasa et al. (2000), being an XML data model, does not allow the representation of general relationships between objects. Considering valid time, which is a user-defined time dimension, operations must explicitly consider time values, to give the user the possibility to manage/modify the time dimension of the objects representing the considered reality. In Amagasa et al. (2000), time values are not explicitly considered as parameters of operations, and moreover operations to modify the valid time interval of an object are not introduced. In our proposal we provide a set of operations explicitly dealing with valid time (see Section 5.3).

The model proposed in Rizzolo and Vaisman (2008) considers the transaction time dimension, and is based on graphs with three types of nodes: *value*, *attribute*, and *element* nodes, more a distinguished node for the *root*, and two types of edges: *containment* and *reference* edges. Temporal labels are used only on edges for representing the time period of the containment relationship. This means that nodes do not have their own temporal information. Since this work deals with XML documents, i.e. documents where nodes are related only by means of a containment relationship, issues related to valid time could be addressed in an analogous way. In general, valid and transaction time dimensions are different in nature and thus are usually managed in different ways. Our proposal allows the representation of general relationships between nodes, and thus the management of either transaction or valid times (or both) requires different precautions, as we discuss in Sections 5, 6 and 7. In Rizzolo and Vaisman (2008), the authors define the set of conditions an XML document must satisfy to be a temporal XML document (with respect to transaction time). The condition stating that the temporal labels of the containment edges incoming to a node are consecutive implies the user do not use temporal elements and thus they cannot represent situations, related to non-consecutive facts, in a plain way. Aspects related to the consistency of temporal XML documents are faced in Rizzolo and Vaisman (2008) by studying algorithms for consistency

checking, while in our proposal we suppose to verify the temporal consistency of each update result.

The set of constraints that temporal labels must satisfy in order to guarantee that (i) a PSTDM graph, after each operation, is still a rooted connected graph and (ii) each atemporal graph composing the PSTDM graph is a rooted connected graph, was preliminarly introduced in Combi et al. (2004) for a generic graph-based model. In Combi et al. (2004), we defined a generic graph-based data model starting from scratch, while in this work we start from the data model proposed in Dyreson et al. (1999) and focus on the properties for representing temporal aspects of semistructured data. Moreover, in Combi et al. (2004) we considered only the transaction time dimension and defined the set of constraints needed for managing in a correct way that dimension by using the graphical formalism proposed in Damiani et al. (2003). Thus, in Combi et al. (2004), a constraint was composed by a graph, which was used to identify the subgraphs (i.e. the portions of a semistructured database) where the constraint was to be applied, and a set of formulae representing restrictions imposed on those subgraphs. In summary, with respect to the present work, in Combi et al. (2004) we (i) defined and used a different, simpler data model, (ii) focused only on the transaction time, and (iii) defined constraints by using a graphical formalism.

9 Conclusions

In this paper we have proposed a graph-based model (called PSTDM) for semistructured data that allows us to model in a homogeneous way (temporal) properties of data. More particularly we discussed in some detail constraints and operations on PSTDM graphs dealing with either transaction or valid times, the two well-known time dimensions of data (Jensen et al. 1998). We showed how a PSTDM graph can represent a sequence of timestamped atemporal graphs, when the transaction time is considered; on the other side, the overall PSTDM graph encodes the real-world history represented by valid time timestamped data.

We extended PSTDM to represent semistructured data by considering together the two classical time dimensions of valid and transaction times. We defined operations for manipulating bitemporal PSTDM graphs, and formalized the set of constraints needed to correctly handle these temporal aspects together.

As for future work, different approaches, such as the logic-based or the algebraic ones, will be considered and studied in order to provide the PSTDM data model with a language for querying, viewing, and "transforming" PSTDM graphs. Furthermore, as PSTDM may be considered as a logical data model for XML database design, it could be the basis for a sound methodology supporting the design of temporal XML databases. To this regard, next steps will be devoted to the design of an overall methodology and to the implementation of related software tools (for example, either based on XML-native database systems or based on XML-enabled extensions of relational database systems). To this regard, a suitable user-based evaluation of the efficacy of the proposed methodology/tools will be considered for both the expressiveness and the performances in representing and managing real world (clinical) temporal semistructured data.

References

Abiteboul, S. (1997). Querying semi-structured data. In *Proceedings of the international conference on database theory. Lecture notes in computer science* (Vol. 1186, pp. 262–275).

Ali, K. A., & Pokorný, J. (2006). A comparison of XML-based temporal models. In E. Damiani, K. Yétongnon, R. Chbeir, & A. Dipanda (Eds.), *SITIS. Lecture notes in computer science* (Vol. 4879, pp. 339–350). Springer.

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

Amagasa, T., Yoshikawa, M., & Uemura, S. (2000). A data model for temporal XML documents. In *Database and expert systems applications, 11th international conference, DEXA 2000. Lecture notes in computer science* (Vol. 1873, pp. 334–344). Berlin: Springer.

Amagasa, T., Yoshikawa, M., & Uemura, S. (2001). A bitemporal XML data model. In *IPSJ SIGNotes dataBase systems* (Vol. 125).

Amagasa, T., Yoshikawa, M., & Uemura, S. (2001). Realizing temporal XML repositories using temporal relational databases. In *Proceedings of the third international symposium on cooperative database systems and applications* (pp. 63–68). IEEE Computer Society.

Atzeni, P. (2002). Time: A coordinate for web site modelling. In *Advances in databases and information systems, 6th east European conference, ADBIS 2002. Lecture notes in computer science* (Vol. 2435, pp. 1–7). Berlin: Springer.

Böhlen, M. H., Snodgrass, R. T., & Soo, M. D. (1996). Coalescing in temporal databases. In *VLDB'96, proceedings of 22th international conference on very large data bases* (pp. 180–191), 3–6 September 1996, Mumbai (Bombay), India. Morgan Kaufmann.

Buneman, P., Davidson, S. B., Hillebrand, G. G., & Suciu, D. (1996). A query language and optimization techniques for unstructured data. In *Proceedings of the 1996 ACM SIGMOD international conference on management of data* (pp. 505–516). ACM Press.

Buneman, P., Khanna, S., Tajima, K., & Tan, W. C. (2002). Archiving scientific data. In *SIGMOD conference.*

Campo, M., & Vaisman, A. A. (2006). Consistency of temporal XML documents. In S. Amer-Yahia, Z. Bellahsene, E. Hunt, R. Unland, & J. X. Yu (Eds.), *XSym. Lecture notes in computer science* (Vol. 4156, pp. 31–45). Springer.

Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., & Tanca, L. (1999). XML-GL: A graphical language for querying and restructuring XML documents. *Computer Network, 31* (11–16), 1171–1187.

Chawathe, S. S., Abiteboul, S., & Widom, J. (1998). Representing and querying changes in semistructured data. In *Proceedings of the fourteenth international conference on data engineering* (pp. 4–13). IEEE Computer Society.

Chawathe, S. S., Abiteboul, S., & Widom, J. (1999). Managing historical semistructured data. *Theory and Practice of Object Systems, 5*(3), 143–162.

Combi, C. (2000). Modeling temporal aspects of visual and textual objects in multimedia databases. In *Proceedings of the seventh international symposium on temporal representation and reasoning (TIME-00)* (pp. 59–68).

Combi, C., Degani, S., & Jensen, C. S. (2008). Capturing temporal constraints in temporal ER models. In Q. Li, S. Spaccapietra, E. S. K. Yu, & A. Olivé (Eds.), *ER. Lecture notes in computer science* (Vol. 5231, pp. 397–411). Springer.

Combi, C., Keravnou-Papailiou, E., & Shahar, Y. (2010). *Temporal information systems in medicine* (1st ed.). Springer.

Combi, C., Oliboni, B., & Quintarelli, E. (2004). A graph-based data model to represent transaction time in semistructured data. In *Database and expert systems applications. Lecture notes in computer science* (Vol. 3180, pp. 559–568). Berlin: Springer.

Combi, C., & Pozzi, G. (2006). Temporal representation and reasoning in medicine. *Artificial Intelligence in Medicine, 38*(2), 97–100.

Consens, M. P., & Mendelzon, A. O. (1990). Graphlog: A visual formalism for real life recursion. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems* (pp. 404–416). ACM Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to algorithms (2nd ed.). MIT press.

Damiani, E., Oliboni, B., Quintarelli, E., & Tanca, L. (2003). Modeling semistructured data by using graph-based constraints. In *OTM workshops proceedings. Lecture notes in computer science* (pp. 20–21). Berlin: Springer.

Dyreson, C. (2001). Towards a temporal world-wide web: A transaction-time web server. In *Proceedings of the Australian Database Conference (ADC '01)* (pp. 169–175).

Dyreson, C. E. (2001). Observing transaction-time semantics with TTXPath. In *Proceedings of the 2nd international conference on Web Information Systems Engineering (WISE'01)* (pp. 193–202).

Dyreson, C. E., Böhlen, M. H., & Jensen, C. S. (1999). Capturing and querying multiple aspects of semistructured data. In *VLDB'99, proceedings of 25th international conference on very large data bases* (pp. 290–301). Morgan Kaufmann.

Dyreson, C. E., & Grandi, F. (2009). Temporal XML. In L. Liu & M. T. Özsu (Eds.), *Encyclopedia of database systems* (pp. 3032–3035). Springer US.

Dyreson, C. E., Snodgrass, R. T., Currim, F., & Currim, S. (2006). Schema-mediated exchange of temporal XML data. In: D. W. Embley, A. Olivé, & S. Ram (Eds.), *ER. Lecture notes in computer science* (Vol. 4215, pp. 212–227). Springer.

Fernandez, M., Florescu, D., Kang, J., Levy, A., & Suciu, D. (1997). STRUDEL: A web site management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD Record* (Vol. 26,2, pp. 549–552). ACM Press.

Garani, G. (2006). A generalised temporal algebra. *Data and Knowledge Engineering, 57*(3), 283–310.

Gibbons, R., et al. (1999). ACC/AHA/ACP-ASIM guidelines for the management of patients with chronic stable angina. *Journal of American College of Cardiology, 33*, 2092–2197.

Grandi, F., & Mandreoli, F. (2000). The Valid Web: An XML/XSL infrastructure for temporal management of web documents. In *Advances in information systems, first international conference, ADVIS 2000. Lecture notes in computer science* (Vol. 1909, pp. 294–303). Berlin: Springer.

Grandi, F., Mandreoli, F., & Tiberio, P. (2005). Temporal modelling and management of normative documents in XML format. *Data & Knowledge Engineering, 54*(3), 327–354.

Gregersen, H., & Jensen, C. S. (1999). Temporal entity-relationship models—a survey. *IEEE Transactions on Knowledge and Data Engineering, 11*(3), 464–497.

Hunter, A. (2002). Merging structured text using temporal knowledge. *Knowledge and Data Engineering, 41*(1), 29–66.

Jensen, C. S., Dyreson, C. E., et al. (1998). M. H. B.: The consensus glossary of temporal database concepts—february 1998 version. In *Temporal databases: Research and practice. The book grow out of a Dagstuhl seminar*, 23–27 June 1997. *Lecture notes in computer science* (Vol. 1399, pp. 367–405). Springer.

Jensen, C. S., & Snodgrass, R. (1999). Temporal data management. *IEEE Transactions on Knowledge and Data Engineering, 11*(1), 36–44.

Li, X., Liu, M., Ghafoor, A., & Sheu, P. C. Y. (2010). A pattern-based temporal XML query language. In L. Chen, P. Triantafillou, & T. Suel (Eds.), *WISE. Lecture notes in computer science* (Vol. 6488, pp. 428–441). Springer.

Mandreoli, F., Martoglia, R., & Ronchetti, E. (2006). Supporting temporal slicing in XML databases. In Y.E. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Böhm, et al. (Eds.), *EDBT. Lecture notes in computer science* (Vol. 3896, pp. 295–312). Springer.

Manica, E., Dorneles, C. F., & de Matos Galante, R. (2010). Supporting temporal queries on XML Keyword Search Engines. *JIDM, 1*(3), 471–486.

Mendelzon, A., & Rizzolo, F. (2004). A.V.: Indexing temporal XML documents. In *VLDB'04, proceedings of 30th international conference on very large data bases* (pp. 216–227). Morgan Kaufmann.

Noh, S. Y., & Gadia, S. K. (2006). A comparison of two approaches to utilizing XML in parametric databases for temporal data. *Information & Software Technology, 48*(9), 807–819.

Noh, S. Y., Gadia, S. K., & Ma, S. (2008). An XML-based methodology for parametric temporal database model implementation. *Journal of Systems and Software, 81*(6), 929–948.

Oliboni, B., Quintarelli, E., & Tanca, L. (2001). Temporal aspects of semistructured data. In *Proceedings of the eighth international symposium on temporal representation and reasoning (TIME-01)* (pp. 119–127). IEEE Computer Society.

Papakonstantinou, Y., Garcia-Molina, H., & Widom, J. (1995). Object exchange across heterogeneous information sources. In *Proceedings of the eleventh international conference on data engineering* (pp. 251–260). IEEE Computer Society.

Paredaens, J., Peelman, P., & Tanca, L. (1995). G–Log: A declarative graphical query language. *IEEE Transactions on Knowledge and Data Engineering, 7*(3), 436–453.

Rizzolo, F., & Vaisman, A. A. (2008). Temporal XML: Modeling, indexing, and query processing. *VLDB Journal, 17*(5), 1179–1212.

Rosado, L. A., Márquez, A. P., & Gil, J. M. (2007). Managing branch versioning in versioned/temporal XML documents. In D. Barbosa, A. Bonifati, Z. Bellahsene, E. Hunt, & R. Unland (Eds.), *XSym. Lecture notes in computer science* (Vol. 4704, pp. 107–121). Springer.

Shoham, Y. (1987). Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence, 33*(1), 89–104.

Snodgrass, R. T. (2000). Developing time-oriented database applications in SQL. *Series in data management systems*. Morgan Kaufmann.

Terenziani, P., & Snodgrass, R. T. (2004). Reconciling point-based and interval-based semantics in temporal relational databases: A treatment of the telic/atelic distinction. *IEEE Transactions on Knowledge and Data Engineering, 16*(5), 540–551.

Vaisman, A., Mendelzon, A. O., Molinari, E., & Tome, P. (2004). Temporal XML: Data model, query language and implementation. Technical Report. http://www.cs.toronto.edu/∼avaisman/papers.html.

World Wide Web Consortium (2000). Document Object Model (DOM) level 2 core specification. http://www.w3C.org/TR/DOM-Level-2-Core/.

World Wide Web Consortium (1999). XML path language (XPath) version 1.0. http://www.w3.org/TR/xpath.html. W3C Reccomendation 16 November 1999.

Wang, F., & Zaniolo, C. (2002). Preserving and querying histories of XML-published relational databases. In *Proceedings of the second international workshop on evolution and change in data management. Lecture notes in computer science* (Vol. 1909, pp. 26–38). Berlin: Springer.

Wang, F., & Zaniolo, C. (2003). Publishing and querying the histories of archived relational databases in XML. In *Proceedings of the 4th international conference on Web Information Systems Engineering (WISE 2003)* (Vol. 1909, pp. 93–102). IEEE Computer Society.

Wang, F., & Zaniolo, C. (2004). XBiT: An XML-based bitemporal data model. In P. Atzeni, W. W. Chu, H. Lu, S. Zhou, & T. W. Ling (Eds.), *ER. Lecture notes in computer science* (Vol. 3288, pp. 810–824). Springer.

Wang, F., & Zaniolo, C. (2008). Temporal queries and version management in XML-based document archives. *Data & Knowledge Engineering, 65*(2), 304–324.

Wang, F., Zaniolo, C., & Zhou, X. (2008). ArchIS: An XML-based approach to transaction-time temporal database systems. *The VLDB Journal, 7*(6), 1445–1463.

Zhang, S., & Dyreson, C. E. (2002). Adding valid time to XPath. In *Databases in networked information. Lecture notes in computer science* (Vol. 2544, pp. 29–42).