The Events Method for Temporal Integrity Constraint Handling in Bitemporal Deductive Databases

Carme Martín Escofet

Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Jordi Girona Salgado 1-3, 08034 Barcelona - Catalonia
e-mail:martin@lsi.upc.es

Abstract. A bitemporal deductive database is a deductive database that supports valid and transaction time. A temporal integrity constraint deals with only valid time, only transaction time or both times. A set of facts to be inserted and deleted in a bitemporal deductive database can be done in a past, present or future valid time and at current transaction time. The temporal integrity constraint handling in bitemporal deductive databases causes that the maintenance of consistency becomes more complex than another databases. The "events method" is based on applying transition and event rules, which explicitly define the insertions and deletions given by a database update. In the conceptual model, we augment the database with temporal transition and event rules and then standard SLDNF-resolution can be used to verify that a transaction does not violate any temporal integrity constraint. In the representational data model, we use time point-based intervals to store temporal information. In this paper, we adapt the "events method" for handling temporal integrity constraints. Finally, we present the interaction between the above-mentioned conceptual and representational data models.

1 Introduction

In temporal databases, two measures of time are distinguished [SA86]: valid time and transaction time. Valid time is the time when the fact is true in the modelled reality, while transaction time is the time when the fact is stored in the database. In a consensus glossary of temporal database concepts [JCG+92], a deductive database that supports valid time and transaction time is called a bitemporal deductive database and we denote it bt-ddb throughout the paper.

An integrity constraint is a condition that a database is required to satisfy at any time. A temporal integrity constraint is an integrity constraint that deals with only valid time, only transaction time or both times. A bt-ddb must be consistent, that is, when performing a past, present or future update, that happens at some valid time point and at current transaction time point, it is necessary to validate whether this update violates some temporal integrity constraint, and if so the update must be rejected. In bt-ddb and valid time deductive databases the possibility of past, present and future valid time updates causes that the maintenance of consistency becomes more complex than transaction time deductive databases, where only transaction time

updates are allowed. Moreover, transaction time deductive databases only deal with transaction time integrity constraints and valid time deductive databases only deal with valid time integrity constraints, whereas bt-ddb can have transaction time, valid time and bitemporal integrity constraints.

Integrity constraint checking is an essential issue, which has been widely studied in deductive databases (see for example [BMM90] for a comprehensive state of the art survey), but not in the field of temporal deductive databases. Although, the research work that have been developed in deductive databases could be the basis for integrity constraint checking methods in temporal deductive databases, as in our case. The simplest solution to check all temporal integrity constraints after each update is highly impractical, principally, for the high cost of reversing the update in the case that constraints were violated. Thus, all the methods assume the consistency of the database before an update. Events method [Oli91] is based on applying transition and event rules, which explicitly defines the insertions and deletions given by a database update and, of course, assume the consistency of the database before an update.

Temporal integrity constraint checking in bt-ddb is an unexplored issue, as you can see in the most recent bibliography of this research area [TK96].

Approaches, such as [Cho92], [Cho95] or [Wüt91], use temporal logic to formulate only transaction time integrity constraints.

[GL95] employs transition graphs to express only valid time integrity constraints.

[SK95], [Sri95] or [VDD95] utilise variants of Event Calculus to develop theirs studies in temporal knowledge representation and reasoning.

[Ple93] formulates integrity constraints in the assertional language provided by Telos [MBJ+90], using the thirteen temporal predicates proposed by [All83]. In addition, [Böh94a] uses the logic-based language ChronoLog. Both of them utilise time intervals to deal with temporal integrity constraints.

Our approach [MS96b] works with temporal integrity constraints in bt-ddb using a first order language and storing temporal information with a time point-based interval representation, in which intervals are just a set of points. We consider that the user are more comfortable working with time points than time intervals and the answers to his queries, using time points, are closer to the user expectations.

In this paper, we present our previous work [MS96b] evolution for handling temporal integrity constraints. We define new temporal transition rules and temporal event rules in order to express more complex integrity constraints. In our previous work integrity constraint definition only contain valid time references. From now on, integrity constraints are able to have valid time, transaction time or both times references. In the conceptual data model, temporal transition rules range over all the possible cases, in which an update could violate these new temporal integrity constraints. We augment the database with above-mentioned temporal transition rules and temporal event rules and then standard SLDNF-resolution can be used to check satisfaction of temporal integrity constraints as our previous work. In the representational data model, we use time point-based intervals to store temporal information. Lastly, we introduce the interaction between conceptual and representational data models.

The paper is organised as follows. Next section defines basic concepts of bt-ddbs.

Section 3 presents the concept of transaction. Section 4 describes the conceptual data model. Section 5 presents the temporal integrity constraint handling using the conceptual model of section 4. Section 6 shows the representational data model. Section 7 defines the relationship between conceptual and representational data models. Finally, section 8 gives the conclusions and points out future research.

2 Bitemporal Deductive Databases

A bt-ddb D consists of three finite sets: a set F of facts, a set R of deductive rules, and a set I of temporal integrity constraints. The set of facts is called the extensional database (EDB), and the set of deductive rules is called the intensional database (IDB). A base predicate appears only in the extensional database and possibly in the body of deductive rules. A derived predicate appears only in the intensional database. We assume that database predicates are either base or derived. Every bt-ddb can be defined in this form.

Facts, rules and temporal integrity constraints are formulated in a first order language. We will use names beginning with a lower case letter for predicate symbols and constants and a capital letter for variables.

We consider a temporal domain τ isomorphic to the set of natural numbers, over which is defined the linear (total) order $<_\tau$, where $t_i <_\tau t_j$ means t_i occurs before t_j. The set τ is used as the basis for incorporating the temporal dimensions into the deductive database.

In this paper, we deal with stratified databases [ABW88] and, as usual, we require the bt-ddb before and after any updates to be allowed [Llo87].

2.1 Deductive Rules

A deductive rule is a formula of the form:

$$p \leftarrow L_1 \wedge \ldots \wedge L_n \quad \text{with } n \geq 1,$$

where p is an atom, denoting the conclusion, $L_1 \wedge \ldots \wedge L_n$ are literals representing conditions. Any variables in p, $L_1 \wedge \ldots \wedge L_n$ are assumed to be universally quantified over the whole formula. A **derived predicate** p may be defined by means of one or more deductive rules, but for the sake of simplicity, we only show the first case in this paper.

Condition predicates may be ordinary or evaluable. The former is a base or derived predicate, while the latter is a built-in predicate that can be evaluated without accessing the database.

2.2 Temporal Integrity Constraints

An integrity constraint is a closed first order formula that the database is required to satisfy. We deal with constraints that have the form of a denial:

$$\leftarrow L_1 \wedge \ldots \wedge L_n \quad \text{with } n \geq 1,$$

where each L_i is a literal. Any variables in $L_1 \wedge \ldots \wedge L_n$ are assumed to be universally quantified over the whole formula.

For the sake of uniformity we associate with each integrity constraint, an inconsistency predicate *ic*, and thus it has the same form as a deductive rule. We call them integrity rules. Then, we rewrite the former denial as:

$$ic \leftarrow L_1 \wedge ... \wedge L_n \quad with\ n \geq 1.$$

Integrity constraints use the usual operators $=$, $>$, $<$, $\geq$, $\leq$ and $\neq$ to compare time points and to express temporal constraints.

In [Böh94b] one can found the following taxonomy of integrity constraints for temporal databases:

Michael Böhlen's Integrity Constraints (IC) Taxonomy:

- Nontemporal IC
- Temporal IC
  - Transaction Time IC
    - Static IC
    - Dynamic IC
      - Transition IC
  - Valid Time IC
    - Intrastate IC
    - Interstate IC
  - Bitemporal IC
    - Intraelement IC
    - Interelement IC

First classification of integrity constraints partitioned them between temporal and nontemporal. Nontemporal integrity constraints are the snapshot database constraints. Transaction time integrity constraints are the transaction time database constraints. Valid time integrity constraints are the valid time database constraints. Lastly, bitemporal integrity constraints are the bitemporal database constraints.

We propose in this section the idea that all the temporal integrity constraints of Michael Böhlen's taxonomy are interesting to consider for bitemporal databases in general, and bt-ddb in particular. Temporal integrity constraints in bt-ddbs deal with only valid time, only transaction time or both times, depending on:

- If the user would define integrity constraints for a definite valid time point and the emphasis is on different transaction time points. The user defines *transaction time integrity constraints*. Figure 2.1.a.

- If the user would define integrity constraints for a definite transaction time point and the emphasis is on different valid time points. The user defines *valid time integrity constraints.* Figure 2.1.b.

- If the user would define integrity constraints for different valid time points and different transaction time points. The user defines *bitemporal integrity constraints.* Figure 2.1.c.

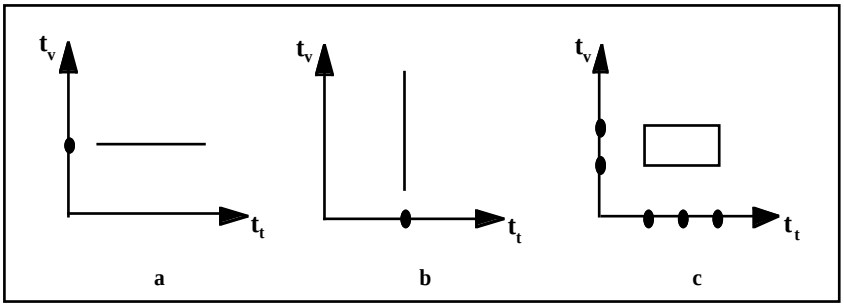

Fig. 2.1.

3 Transactions

The user can insert (ιp) and delete (δp) facts from an specific valid time t_V to an artificial time point that represent the end of time, denoted by *forever*, and only at transaction time *now*. $\iota p(x,t_V)$ and $\delta p(x,t_V)$ represent the external events given by the update. Therefore, we assume from now on that a transaction *U* consists of an unspecified set of insertion and deletion of external events.

Example.

Suppose the transaction *TR1* at transaction time 3 (now) on the bt-ddb as it is shown in figure 3.1.:

$\{\iota offered(databases,2), \iota takes(ton,databases,2), \delta takes(maria,logic,2)\}$

Where, in the figure, for the sake of clarity, *offered(C)* represents the base predicate $offered(C,T_V,T_t)$, which expresses that "course *C* is offered at valid time T_V and at transaction time T_t". And *takes(S,C)* represents the base predicate $takes(S,C,T_V,T_t)$, which expresses that "the student S is enrolled in course *C* at valid time T_V and at transaction time T_t".

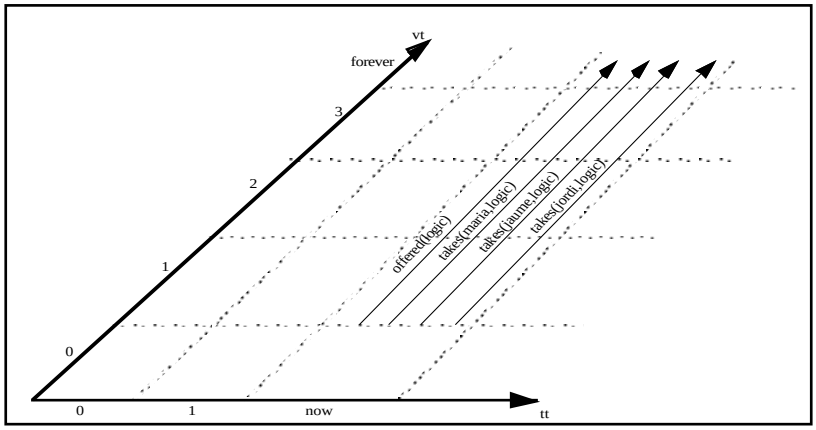

Fig. 3.1.

So this transaction does not violate any integrity constraint, at transaction time 3, we have the bt-ddb as it is shown in figure 3.2.

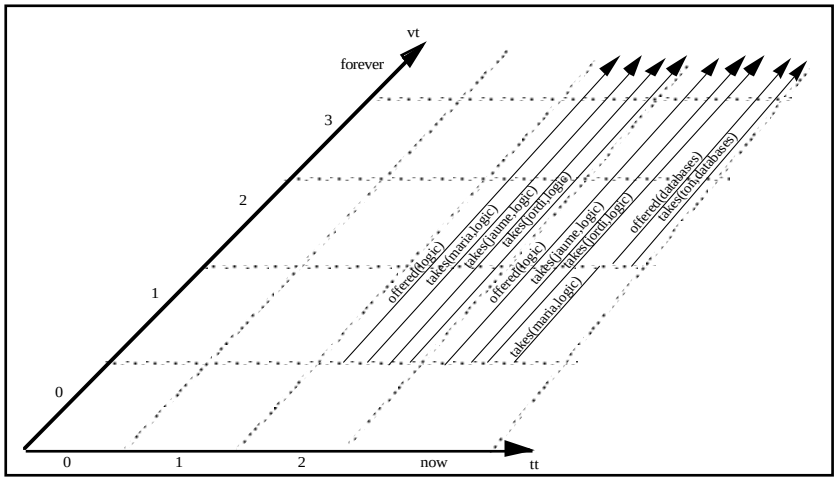

Fig. 3.2.

Fact insertions ($\iota p(x,t_V)$) call another predicate ($\iota p^*(x,t_V,now)$) that ensures the non-existence of $p(x,t_V,t_t)$ from valid time t_V to *forever* at transaction time *now*.

Algorithm:

$\iota p(X,T_V) \leftarrow \iota p^*(X,T_V,now)$.

$\iota p^*(X,T_V,T_t) \leftarrow$ *$p(x,t_V,t_t)$ not exists in the representational data model and the insertion is allowed.*

$\iota p^*(X,T_V,T_t) \leftarrow$ *$p(x,t_V,t_t)$ exists in the representational data model but not from a valid time start point lesser than or equal t_V to and a valid time point end greater than t_V and the insertion is allowed.*

$\iota p^*(X,T_V,T_t) \leftarrow$ *$p(x,t_V,t_t)$ exists in the representational data model in t_V, so it is known that this predicate is already offered.*

Fact deletions ($\delta p(x,tv)$) call another predicate ($\delta p^*(x,tv,now)$) that ensures the existence of $p(x,tv,tt)$ at valid time *tv* and at transaction time *now*.

Algorithm:

$\delta p(X,T_V) \leftarrow \delta p^*(X,T_V,now)$.

$\delta p^*(X,T_V,T_t) \leftarrow$ *$p(x,t_V,t_t)$ exits in the representational data model from a valid time start point lesser than t_V to forever* and the deletion is allowed.

$\delta p^*(X,T_V,T_t) \leftarrow$ *$p(x,t_V,t_t)$ exists in the representational data model but a valid time end point exists in the valid time interval that contains t_V, so it is known that this predicate is already finished.*

$\delta p^*(X,T_V,T_t) \leftarrow$ *$p(x,t_V,t_t)$ not exists in the representational data model, so it is known that this predicate is not offered.*

4 Conceptual Data Model

In [MS94b], we adapted the concepts of event, transition and event rules that were formalised in [Oli89] for the events model. The events model is an approach for the design of information systems from deductive conceptual models, and was applied in [SO94] to address database and transaction design decisions. In [Oli89] and [SO94] valid and transaction time are equivalent and the database can only be updated in the current state. The integrity constraints are transaction time integrity constraints. In our case, we explicitly distinguish between valid and transaction times and the updates can be done in a past, present or future valid time. Moreover, we deal with temporal integrity constraints whereas transaction time integrity constraints. In this section, we introduce temporal events, temporal transition and event rules and the augmented database necessary to define transaction, valid and bitemporal integrity constraints.

4.1 Temporal Events

Let D be a bt-ddb at transaction time point t_t *(now)* before an update U, and D' the updated bt-ddb at transaction time point t_t after the update U, as one can see in figure 4.1. We assume for the moment that U consists of an unspecified set of facts to be inserted and deleted and the bt-ddb can only be updated in the transaction time point *now*.

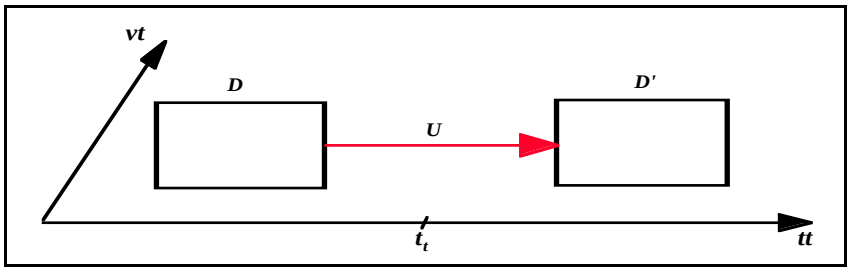

Fig. 4.1.

Let $p(x,t_V,t_t)$ be a fact in D and let $p'(x,t_V,t_t)$ denote the same fact evaluated in D'. Assuming that $p(x,t_V,t_t)$ holds in D, where x is a vector of constants, t_V is a valid time point, and t_t is the transaction time point *now*, two cases are possible:

$p'(x,t_V,t_t)$ also holds in D' (both $p(x,t_V,t_t)$ and $p'(x,t_V,t_t)$ are true). (1)

$p'(x,t_V,t_t)$ does not hold in D' ($p(x,t_V,t_t)$ is true, but $p'(x,t_V,t_t)$ is false). (2)

And assuming that $p'(x,t_V,t_t)$ holds in D', two cases are also possible:

$p(x,t_V,t_t)$ also holds in D (both $p(x,t_V,t_t)$ and $p'(x,t_V,t_t)$ are true). (3)

$p(x,t_V,t_t)$ does not hold in D ($p'(x,t_V,t_t)$ is true, but $p(x,t_V,t_t)$ is false). (4)

In case (2) we say that a **temporal deletion event** occurs in the transition at valid time point t_V, we denote it by $\delta p'(x,t_V,t_t)$ and we store it at transaction time point t_t as it is shown in figure 4.2.

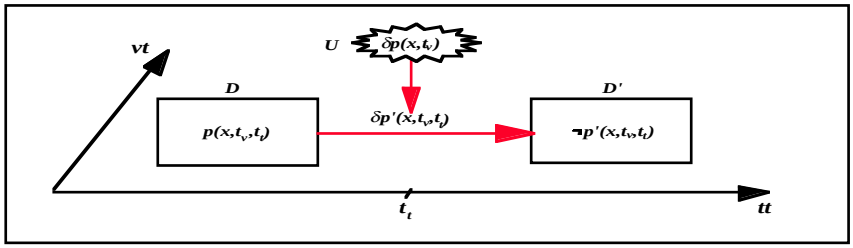

Fig. 4.2.

In case (4) we say that a **temporal insertion event** occurs in the transition at valid time point t_V, we denote it by $\iota p'(x,t_V,t_t)$ and we store it at transaction time point t_t as it is shown in figure 4.3.

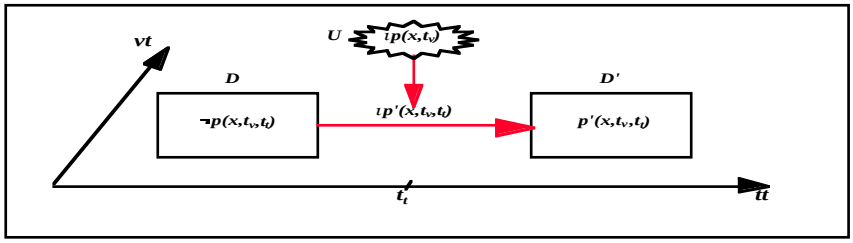

Fig. 4.3.

Formally, we associate a temporal insertion event predicate $\iota p'$ with each base, derived or inconsistency predicate p and a temporal deletion event predicate $\delta p'$ with each base or derived predicate, defined as:

$\forall X,T_V,T_t\,(\iota p'(X,T_V,T_t) \leftrightarrow p'(X,T_V,T_t) \wedge \neg p(X,T_V,T_t))$. (5)

$\forall X,T_V,T_t\,(\delta p'(X,T_V,T_t) \leftrightarrow p(X,T_V,T_t) \wedge \neg p'(X,T_V,T_t))$. (6)

where X is a vector of variables, T_V is a valid time point variable and T_t is a transaction time point variable.

From the above, we then have the equivalences:

$\forall X,T_V,T_t\,(p'(X,T_V,T_t) \leftrightarrow [p(X,T_V,T_t) \wedge \neg\delta p'(X,T_V,T_t)] \vee \iota p'(X,T_V,T_t))$. (7)

$\forall X,T_V,T_t\,(\neg p'(X,T_V,T_t) \leftrightarrow [\neg p(X,T_V,T_t) \wedge \neg\iota p'(X,T_V,T_t)] \vee \delta p'(X,T_V,T_t))$. (8)

Which relate the predicate p' at transaction time point t_t after the update to the predicate p at transaction time point t_t before the update and the events given by the transaction.

If p is a base predicate, then $\iota p'$ and $\delta p'$ represent temporal external insertions and deletions respectively.

If p is a derived predicate, then $\iota p'$ and $\delta p'$ represent temporal internal insertions and deletions respectively.

If p is an inconsistency predicate, then $\iota p'$ that occur during the transition will correspond to violations of its integrity constraint. For example, if a given transition induces $\iota ic'$ then it will mean that such transition leads to a violation of integrity constraint ic. Note that for inconsistency predicates $\delta p'$ cannot happen in any transition, since we assume that the bt-ddb is consistent before the update, and thus ic is always false.

4.2 Temporal Transition Rules

Let $p \leftarrow L_1, ..., L_i, ..., L_m$ be a deductive or inconsistency rule. When the rule is to be evaluated in the updated bt-ddb, its form is $p' \leftarrow L_1', ..., L_i', ..., L_m'$, where L_i' is obtained by replacing the predicate Q of L_i with Q'. Now if we rewrite each literal in the body by its equivalent definition, given in (7) or (8), we get a new rule called a **temporal transition rule**, which defines predicate p' in the updated bt-ddb in terms of transaction time point *now-1* of the predicates appearing in the body of the rule, and the events that occur at transaction time point *now*.

More precisely, if L_i' is an ordinary positive literal $Q_i'(X_i,T_{vi},T_{ti})$ we apply (7) and replace it with:

$$(Q_i(X_i,T_{vi},T_{ti}) \wedge \neg\delta Q_i'(X_i,T_{vi},T_{ti})) \vee \iota Q_i'(X_i,T_{vi},T_{ti})$$

and if L_i' is an ordinary negative literal $\neg Q_i'(X_i,T_{vi},T_{ti})$ we apply (8) and replace it with:

$$(\neg Q_i(X_i,T_{vi},T_{ti}) \wedge \neg\iota Q_i'(X_i,T_{vi},T_{ti})) \vee \delta Q_i'(X_i,T_{vi},T_{ti})$$

If L_i is an evaluable predicate, we just replace L_i'(positive or negative) by its current L_i.

It will be easier to refer to the resulting expressions if we denote by:

$O(L_i')$	$= (Q_i(X_i,T_{vi},T_{ti}) \wedge \neg\delta Q_i'(X_i,T_{vi},T_{ti}))$	*if* $L_i' = Q_i'(X_i,T_{vi},T_{ti})$
	$= (\neg Q_i(X_i,T_{vi},T_{ti}) \wedge \neg\iota Q_i'(X_i,T_{vi},T_{ti}))$	*if* $L_i' = \neg Q_i'(X_i,T_{vi},T_{ti})$
	$= L_i$	*if* L_i *is evaluable*
$N(L_i')$	$= \iota Q_i'(X_i,T_{vi},T_{ti})$	*if* $L_i' = Q_i'(X_i,T_{vi},T_{ti})$
	$= \delta Q_i'(X_i,T_{vi},T_{ti})$	*if* $L_i' = \neg Q_i'(X_i,T_{vi},T_{ti})$

Both $O(L_i')$ and $N(L_i')$ express conditions for which L_i' is true. $O(L_i')$ corresponds to the case that L_i' holds because L_i was already true in the **O**ld transaction time point *now-1* and has not been deleted, while $N(L_i')$ corresponds to the case that $N(L_i')$ holds because it is **N**ew, induced in the transition, and false before. Note that $O(L_i') \rightarrow L_i$ and $N(L_i') \rightarrow \neg L_i$.

With this notation, the equivalences (7) and (8) become:

$$\forall X,T_v,T_t\ (p'(X,T_v,T_t) \leftrightarrow O(p'(X,T_v,T_t)) \vee N(p'(X,T_v,T_t))). \quad (9)$$

$$\forall X,T_v,T_t\ (\neg p'(X,T_v,T_t) \leftrightarrow O(\neg p'(X,T_v,T_t)) \vee N(\neg p'(X,T_v,T_t))). \quad (10)$$

And applying them to each of the L_i' (i = 1...n) literals, we get:

$$p'(X,T_v,T_t) \leftarrow \bigwedge_{i=1}^{i=n} [(O(L_i') \vee N(L_i')) \vee O(L_i')] \quad (11)$$

where the first option is taken if L_i' is an ordinary literal, and the second one if L_i' is evaluable. After distributing $\wedge$ over $\vee$, we get an equivalent set of 2^k transition rules, each of them with the general form:

$$p_j'(X,T_v,T_t) \leftarrow \bigwedge_{i=1}^{i=n} [O(L_i') \vee N(L_i')] \qquad \text{with } j = 1, ..., 2^k \quad (12)$$

where k is the number of ordinary literals in the $p'(X,T_v,T_t)$ rule, and

$$p'(X,T_v,T_t) \leftarrow p_j'(X,T_v,T_t) \qquad with\ j = 1, ..., 2^k. \tag{13}$$

We are conscious of the resulting amount of transition rules and we presented in [MS96b] some simplifications to drastically reduce or eliminate them.

Note that in the case of temporal integrity constraints, $\neg ici_1'$ always holds because the bt-ddb is assumed to be consistent at transaction time point *now-1*.

4.3 Temporal Insertion Event Rules

Let p be a derived or inconsistency predicate. Temporal insertion events of p were defined in (5) as:

$$\forall X,T_v,T_t\ (\iota p'(X,T_v,T_t) \leftrightarrow p'(X,T_v,T_t) \wedge \neg p(X,T_v,T_t)).$$

And replacing $p'(X,T_v,T_t)$ by its equivalent definition given in (13) we get:

$$\iota p'(X,T_v,T_t) \leftarrow p_i'(X,T_v,T_t) \wedge \neg p(X,T_v,T_t) \qquad with\ \mathrm{i} = 1, ..., 2^k. \tag{14}$$

By replacing $p_i'(X,T_v,T_t)$ with its equivalent definition given in (12), we get a set of **temporal insertion events rules**. They allow us to deduce which $\iota p'$ happen in a transition. If p is an inconsistency predicate, $\iota p'$ correspond to a violation of the integrity constraint. Note that in the case of integrity constraints, $\neg ici$ always holds because the bt-ddb was consistent at transaction time point *now-1*, and we can eliminate this literal:

$$\iota ici' \leftarrow ici_i' \quad i=2, ..., 2^k \tag{15}$$

4.4 Temporal Deletion Event Rules

Let p be a derived predicate. Temporal deletion events of p were defined in (6) as:

$$\forall X,T_v,T_t\ (\delta p'(X,T_v,T_t) \leftrightarrow p(X,T_v,T_t) \wedge \neg p'(X,T_v,T_t)).$$

And replacing $p'(X,T_v,T_t)$ by its equivalent definition given in (13) we get:

$$\delta p'(X,T_v,T_t) \leftarrow p(X,T_v,T_t) \wedge \neg p_i'(X,T_v,T_t)\ ) \qquad with\ \mathrm{i} = 1, ..., 2^k. \tag{16}$$

By replacing $p_i'(X,T_v,T_t)$ with its equivalent definition given in (12), we get a set of **temporal deletion events rules**. They allow us to deduce which $\delta p'$ happen in a transition. Note that in the case of integrity constraints, $\delta icn'$ cannot happen in any transition, since we assume that the bt-ddb is consistent before the update, and thus *icn* is always false.

4.5 The Augmented Database

Let D be a bt-ddb. We denote the augmented bt-ddb by A(D), based in the concept of augmented deductive database defined in [Oli91], to the bt-ddb consisting of D, its temporal transition rules and its temporal event rules.

If SLDNF-resolution is complete for D, then it will also be complete for A(D). [Oli91].

The augmented bt-ddb described in the previous section can be used directly to check if a transaction produces or not inconsistencies.

Let *D* be a bt-ddb, *A(D)* the augmented bt-ddb, and *TR* a transaction consisting of a set of events at valid time point T_V and at transaction time T_t. If *TR* leads to an inconsistency then some of the *ιicn* will hold in the transition. Using SLDNF-proof procedure, *TR* violates integrity constraint *icn* if the goal *←ιicn'* succeeds from input set *A(D)* ∪ *TR*. If every branch of the SLDNF-search space for *A(D)* ∪ *TR* ∪ *{←ιicn'}* is a failure branch, then *TR* does not violate *icn*, as show in figure 4.5.

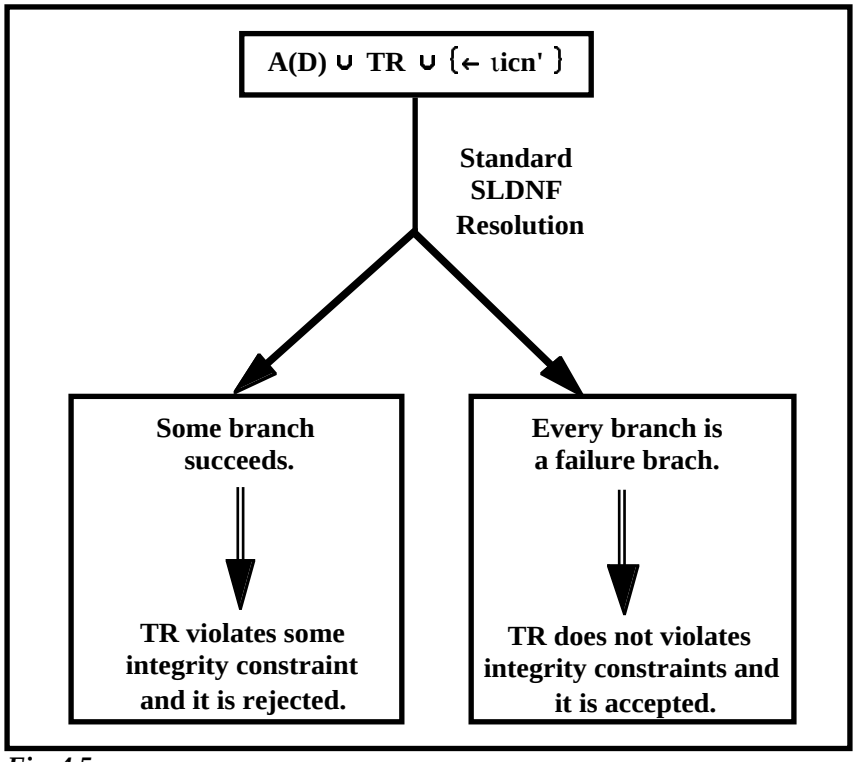

Fig. 4.5.

More details and examples about our integrity constraint checking method in bt-ddb for valid time integrity constraints can be found in [MS96b], that incorporates transaction time to our previous work (see [MS94] and [MS96a]).

5 Temporal Integrity Constraint Handling

In this section, we present the idea of how these new temporal transition rules deal with different temporal integrity constraints in bt-ddbs, using the Michael Böhlen's taxonomy notation.

Transaction Time Integrity Constraints.

Transaction time integrity constraints could be static or dynamic. Static integrity constraints must hold at every transaction time point. Dynamic integrity constraints relate and restrict arbitrary transaction time points of the transaction time axis. A particular case of dynamic integrity constraints is transition integrity constraint that deals with two succeeding transaction time points. Transaction time integrity constraints only can relate past or present transaction time points. If the user expresses some transaction time integrity constraint in terms of future time points, then it will be converted in terms of past transaction time points. The idea of this kind of integrity constrains is shown in figure 5.1.

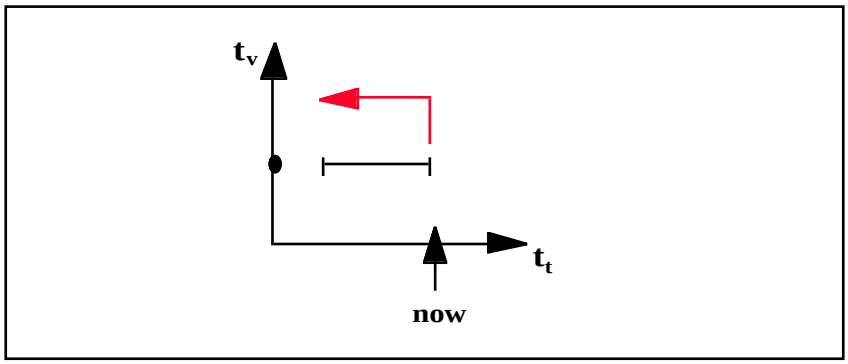

Figure 5.1.

A transaction time integrity constraint could be violated if an update at transaction time *now*, insert or delete some fact that produces an external event or induces an internal event that violate it. In transaction time integrity constraints, (7) and (8) will be interpreted as:

$\forall X,T_t\ (p'(X,t_v,T_t) \leftrightarrow [p(X,t_v,T_t) \wedge \neg \delta p'(X,t_v,T_t)] \vee \iota p'(X,t_v,T_t))$.

$\forall X,T_t\ (\neg p'(X,t_v,T_t) \leftrightarrow [\neg p(X,t_v,T_t) \wedge \neg \iota p'(X,t_v,T_t)] \vee \delta p'(X,t_v,T_t))$.

Note that, all the possible cases that could violate a transaction time integrity constraint are expressed in (7) and (8).

For example, $ic1 \leftarrow takes(S,C,T_v,T_t) \wedge \neg offered(C,T_v,T_t)$, that is a static integrity constraints, enforces the property that "a student *S* can only be enrolled in course *C* if course *C* is offered now". We could violate it if we delete a course with students enrolled, or if we insert a student in a course that does not exist or if we insert a student but delete the course where the student has been enrolled. These are all the cases that violate *ic1*, and all them are expressed by the temporal transition rules.

Valid Time Integrity Constraints.

Valid time integrity constraints could be intrastate or interstate. Intrastate integrity constraints must hold at every valid time point. Interstate integrity constraints relate and restrict arbitrary valid time points of the valid time axis. Valid time integrity constraints can relate past, present or future valid time points. The idea of this kind of integrity constrains is showing in figure 5.2.

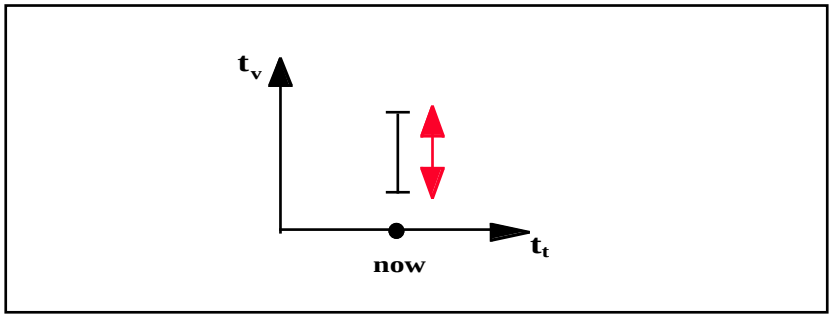

Figure 5.2.

A valid time integrity constraint in a bt-ddb could be violated if an update at transaction time *now*, insert or delete some fact that produces an external event or induces an internal event that violate it. In valid time integrity constraints, (7) and (8) will be interpreted as:

$\forall X,T_V\ (p'(X,T_V,t_t) \leftrightarrow [p(X,T_V,t_t) \wedge \neg\delta p'(X,T_V,t_t)] \vee \iota p'(X,T_V,t_t))$.

$\forall X,T_V\ (\neg p'(X,T_V,t_t) \leftrightarrow [\neg p(X,T_V,t_t) \wedge \neg\iota p'(X,T_V,t_t)] \vee \delta p'(X,T_V,t_t))$.

Note that, all the possible cases that could violate a valid time integrity constraint are expressed in (7) and (8).

For example, $ic2 \leftarrow takes(S,software\ engineering,T_V,T_t) \wedge takes(S,information\ systems,T1_V,T_t) \wedge T1_V \leq T_V$, that is a interstate integrity constraint, enforces the property that "if a student *S* is enrolled in the course *software engineering*, this student cannot be enrolled or cannot have been enrolled in the course *information systems*". We could violate it if we enrol a student in the course *software engineering* and he/she is enrolled or has been enrolled in the course *information systems*, or if we enrol a student in the course *information systems* and he/she is enrolled or will be enrolled in the course *software engineering*, or if we enrol at the same time a student in both courses with a valid time for *information systems* previous than or equal to the valid time for *software engineering.* These are all the cases that violate *ic2*, and all them are expressed by the temporal transition rules.

Bitemporal Integrity Constraints.

Bitemporal integrity constraints are the most complex type of temporal integrity constraints. They could be intraelement or interelement. Intraelement integrity constraints only are a generalisation of intrastate respectively static integrity constraints. Interelement integrity constraints relate and restrict arbitrary time points

of the valid time and transaction time axis. If the user expresses some bitemporal integrity constraint in terms of future transaction time points, it will be converted in terms of past transaction time points. The idea of this kind of integrity constrains is showing in figure 5.3.

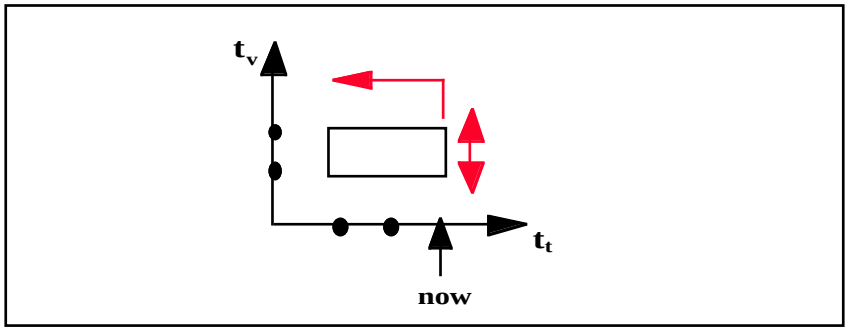

Figure 5.3.

A bitemporal integrity constraint in a bt-ddb could be violated if an update at transaction time *now*, insert or delete some fact that produces an external event or induces an internal event that violate it. Bitemporal integrity constraints, use all the power of (7) and (8):

$\forall X,T_V,T_t\ (p'(X,T_V,T_t) \leftrightarrow [p(X,T_V,T_t) \wedge \neg \delta p'(X,T_V,T_t)] \vee \iota p'(X,T_V,T_t))$.

$\forall X,T_V,T_t\ (\neg p'(X,T_V,T_t) \leftrightarrow [\neg p(X,T_V,T_t) \wedge \neg \iota p'(X,T_V,T_t)] \vee \delta p'(X,T_V,T_t))$.

Note that, all the possible cases that could violate a transaction time integrity constraint are expressed in (7) and (8).

For example, $ic3 \leftarrow many\_students(C,T_V,T_t) \wedge \neg many\_students(C,T1_V,T1_t) \wedge T_V<T1_V \wedge T_t \leq T1_t$, that is a interelement integrity constraint, enforces the property that "if a course *C* has more than one student, then this course cannot have fewer than one student in the future". $many\_students(C,T_V,T_t) \leftarrow takes(S1,C,T_V,T_t) \wedge takes(S2,C,T_V,T_t) \wedge S1 \neq S2$ is a deductive rule for the derived predicate $many\_students(C,T_V,T_t)$ and expresses that "in a course *C* there are many students if at least two different students were enrolled in this course *C*". We could violate it if we have many students in a course *C* at T_V and T_t, and we delete students just to less only one student enrolled in a valid time greater than T_V and greater than or equal to T_t, or if we have a student enrolled in a course *C* at T_V and T_t and in a transaction time greater than or equal to T_t insert students in the course at a valid time lesser than T_V, or if at the same transaction time we delete students from a course *C* at valid time T_V and insert students in a valid time lesser than T_V. These are all the cases that violate *ic3*, and all them are expressed by the temporal transition rules.

6 Representational Data Model

We adopt a closed time point-based interval model used in the valid time representation presented in [Sar93], adding a transaction time dimension. Including transaction time we ensure that every old state is preserved. If we store only valid

time one cannot remember if during a given period one knew another information different from the current one. We are interested in the history of the database and we willing to pay a high cost of the storage of old states. [Sar93] uses two segments to representing current and history data, in which two valid time points are added, named FROM and TO (defining a valid time interval), valid time start and end in our case. We only use the equivalent to one segment and we add two more time points (transaction time start and end) to represent transaction time and to define a transaction time interval as we show in figure 6.1.

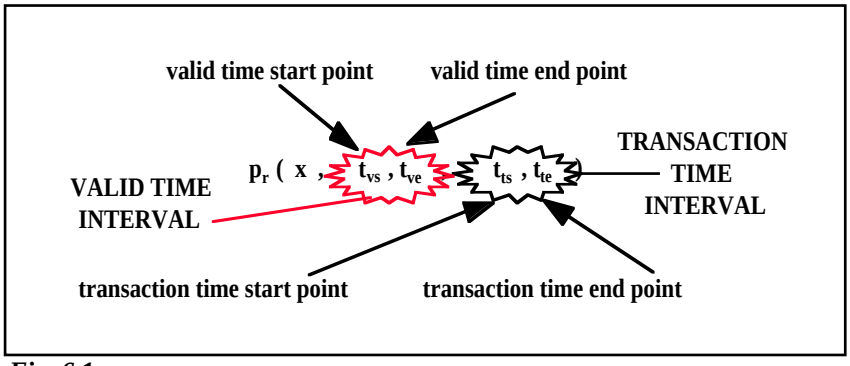

Fig. 6.1.

A fact is a ground atom. Last terms of any fact are four time points values ranging over the temporal domain τ: valid time-start (t_{VS}), valid time-end (t_{Ve}) corresponding to the lower and upper bounds of the valid time interval and transaction time-start (t_{TS}), transaction time-end (t_{Te}) corresponding to the lower and upper bounds of the transaction time interval.

7 Interaction between Conceptual and Representational Data Models

Each fact has a precise valid time-start t_{VS} value stored from transaction time-start t_{TS} to *now* (denoting the current time) or to transaction time-end t_{Te} when finally the fact cannot be accessible from current time any more. However, the valid time-end value t_{Ve} may not be known. In this case, t_{Ve} is given the default value *forever* denoting an artificial time point for the end of time ready to handle future information, but that will change to precise value t_{Ve} when the user knows it.

Note that an event happens at some time instant, while we require time intervals to express the changes produced by the transaction. Therefore, when an insertion event $\iota p(x,t_V)$ happens in a transaction time t_t we really represent: $p_r(x,t_V,forever,t_t,now)$, as it is shown in figure 7.1. And when a deletion event $\delta p(x,t_V)$ occurs in a transaction time t_t we modify $p_r(x,t_{VS},t_{Ve},t_{TS},now)$ by $p_r(x,t_{VS},t_V\text{-}1,t_t,now)$ and $p_r(x,t_{VS},t_{Ve},t_{TS},t_t\text{-}1)$, as one can see in figure 7.2. When a deletion event occurs we do not really remove information; instead we store the fact that it has existed from one valid time to another valid time.

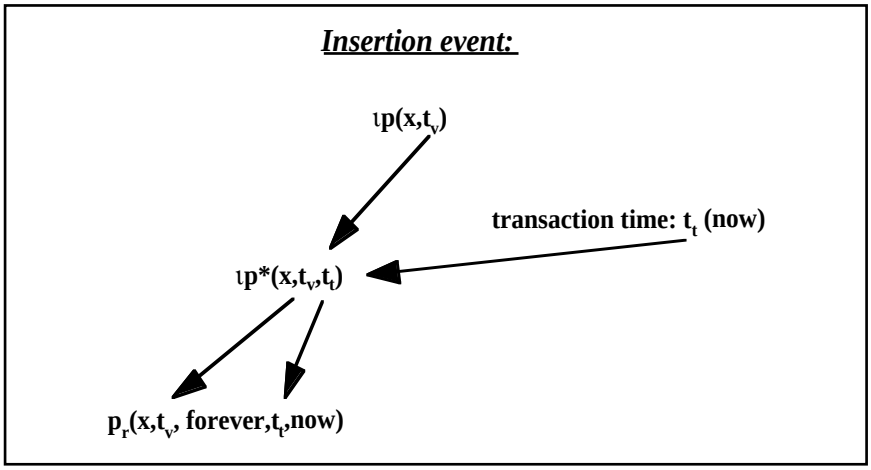

Fig. 7.1.

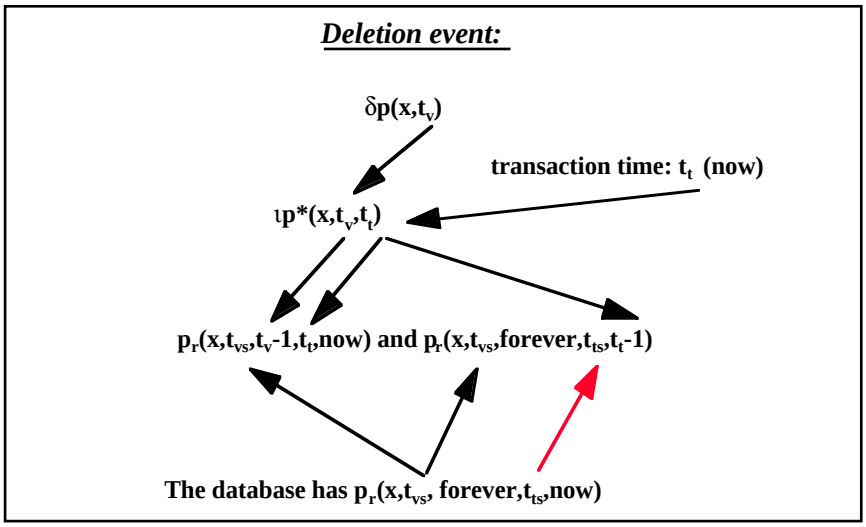

Fig. 7.2.

8 Conclusions and Further Work

In this paper we have presented how and why applying temporal transition and event rules is possible a complete handling of temporal integrity constraints in bt-ddbs. We clarify that temporal transition rules are important to recognise all the possible cases that produce violations of temporal integrity constraints. We have described our conceptual data model and our representational data model and the interaction between them. A graphical idea of the concepts introduced in this paper can be seen in figure 8.1.

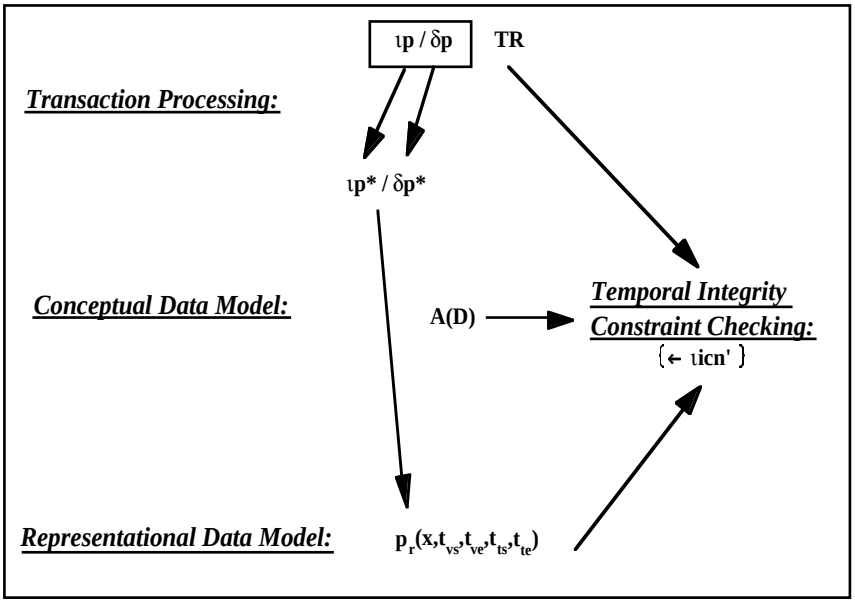

Figure 8.1.

Our further work consists in completing our approach for temporal integrity constraint checking in bt-ddbs with simplifications of the event rules and more simplifications of transition rules to increase the efficiency. To achieve this goal we are going to study carefully the idea of temporal integrity constraint handling presented in section 5.

Acknowledgements

The author would like to thank Antoni Olivé and Jaume Sistac for the support they have given to this work and also P. Costa, D. Costal, E. Mayol, J. A. Pastor, C. Quer, M. R. Sancho, E. Teniente, T. Urpí and specially Michael Gertz for many useful comments and discussions.

References

[All83] Allen, J.F. "Maintaining Knowledge about Temporal Intervals". In Communications of the ACM. Vol. 26. No. 11. 1983. pp 832-843.

[ABW88] Apt, K.R.; Blair, H.A.; Walker, A. "Towards a Theory of Declarative Knowledge". In Foundations of Deductive Databases and Logic Programming (J.Minker ed.). Morgan Kaufmann. 1988. pp 89-148.

[Böh94a] Böhlen, M. "Managing Temporal Knowledge in Deductive Databases". PhD thesis, Swiss Federal Institute of Technology. Zürich, 1994.

[Böh94b] Böhlen, M. "Valid Time Integrity Constraints". Technical Report 94-30. Computer Science. University of Arizona, 1994.

[BMM90] Bry, F.; Manthey, R.; Martens, R. “Integrity Verification in Knowledge Bases”. ECRC Report D.2.1.a, München, April 1990, 26 p.

[Cho92] Chomicki, J. "History-less Checking of Dynamic Integrity Constraints". In the 8th Int. Conf. on Data Engineering, IEEE Computer Society Press. Phoenix AZ, February, 1992. pp 557-564.

[Cho95] Chomicki, J. "Efficient Checking Encoding of Temporal Integrity Constraints Using Bounded History Encoding". ACM Transactions on Database Systems. June, 1995. pp 149-186.

[GL95] Gertz, M.; Lipeck, U.W. ""“Temporal” Integrity Constraints in Temporal Databases". Proc. of the International Workshop on Temporal Databases. Zürich. (Clifford/Tuzhilin Eds). Springer-Verlag, September, 1995. pp 77-92.

[JCG+92] Jensen, C.S.; Clifford, J.; Gadia, S.K.; Segev, A.; Snodgrass, R.T. "A Glossary of Temporal Database Concepts". Proc. SIGMOD-RECORD. Vol. 21. 1992.

[KS86] Kowalski, R.; Sergot, M. “A Logic-Based Calculus of Events” New Generation Computing. Vol 4, No. 1. February, 1986. pp 67-95. OHMSHA LTD and Springer-Verlag.

[Llo87] Lloyd, J.W. "Foundations of Logic Programming". Second edition. Springer, 1987.

[MBJ+90] Mylopoulos, J.; Borgida, A.; Jarke, M.; Koubarakis, M. "Telos: Representing Knowledge About Information Systems". ACM Transactions on information systems. Vol. 8. No. 4. 1990. pp 324-362.

[MS94] Martín, C.; Sistac, J. "Integrity Constraints Checking in Historical Deductive Databases". Proc. of the 5th. Int. Workshop on the Deductive Approach to Information Systems and Databases. Aiguablava, Catalonia, 1994. pp 299-324.

[MS96a] Martín, C.; Sistac, J. "A Method for Integrity Constraint Checking in Temporal Deductive Databases". Proc. of the 3th. Int. Workshop on Temporal Representation and Reasoning. IEEE Computer Society Press. Key West, Florida, May, 1996. pp 136-141.

[MS96b] Martín, C.; Sistac, J. "Applying Transition Rules to Bitemporal Deductive Databases for Integrity Constraint Checking". Proc. of the Int. Workshop on Logic in Databases. San Miniato, Italy. [Pedreschi/Zaniolo Eds.]. Springer Verlag, July, 1996. pp 117-134.

[Oli89] Olivé, A. "On the Design and Implementation of Information Systems from Deductive Conceptual Models". Proc. of the 15th. Int. Conf. on VLDB'89, pp 3-11.

[Oli91] Olivé, A. "Integrity Constraints Checking in Deductive Databases". Proc. of the 17th. Int. Conf on VLDB'91. pp 513-523.

[Ple93] Plexousakis, D. "Integrity Constraint and Rule Maintenance in Temporal Deductive Knowledge Bases". Proc. of the 19th. Int. Conf. on VLDB'93. pp 146-157.

[SK95] Sadri, F.; Kowalski, R. “Variants of the Event Calculus”. Proc. of the 12th Int. Conf. on Logic Programming, 1995. pp 67-81.

[Sar93] Sarda, N.L. "HSQL: A Historical Query Language". In [TCG+93], pp 110-140.

[SA86] Snodgrass, R.; Ahn, I. "Temporal Databases". IEEE Computer. Vol 19. No. 9. September, 1986.

[SO94] Sancho, M.R; Olivé, A. "Deriving Transactions Specifications from Deductive Conceptual Models of Information Systems". Proc. of CAiSE'94 conference. 1994, pp 311-324.

[Sri95] Sripada, S.M. "Efficient Implementation of the Event Calculus for Temporal Database Applications". Proc. of the 12th Int. Conf. on Logic Programming, 1995. pp 99-113.

[TCG+93] Tansel, A.U.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; Snodgrass, R. "Temporal Databases: Theory, Design and Implementation". Benjamin/Cummings. 1993.

[TK96] Tsotras, V.J.; Kumar, A. "Temporal Database Bibliography Update". In ACM SIGMOD Record. Vol 25. Num 1. March, 1996. pp 41-51.

[VDD95] Van Belleghem, K.; Denecker, M.; De Schreye, D. “Combining Situation Calculus and Event Calculus”. Proc. of the 12th Int. Conf. on Logic Programming, 1995. pp 83-97.

[Wüt91] Wüthrich, B. "Large Deductive Databases with Constraints". PhD thesis, Swiss Federal Institute of Technology. Zürich, 1991.