Interoperability of Heterogeneous Temporal Relational Databases

Cristina De Castro **Fabio Grandi** **Maria Rita Scalas**

C.I.O.C.-C.N.R. and Dipartimento di Elettronica, Informatica e Sistemistica
Università degli Studi di Bologna, Italy

Abstract

Interoperability is concerned with the interaction of database systems of different kinds. In this paper we consider semantic interoperability between relational databases of different temporal types. Temporal applications must be allowed to share and exchange information which are stored in databases without time support (snapshot), in databases supporting transaction or valid time (monotemporal) or supporting both time dimensions (bitemporal). Moreover, the necessity for internal interoperability also arises in a temporal system which allows the coexistence of relations with different temporal formats.

The temporal natural join is the most important operation for combining data stored in different relations and, thus, is also mainly concerned in temporal interoperability problems. Therefore, a suitable variant of algorithms for temporal natural join is presented. The variant is based on the translation of temporally heterogeneous relations into a common format and on a mechanism of completion of relations which store incomplete information along some time dimension. The process is designed in order to retrieve the largest amount of certain data and avoid the creation of spurious information. Semantic and operative issues connected with the proposal are discussed.

Keywords: *Heterogeneous Databases, Semantic Interoperability, Temporal Database, Relational Database, Temporal Conversion Maps, Incomplete Information, Temporal Natural Join.*

1 Introduction

The interaction of heterogeneous database systems [EP90, Sh91] is one of the requirements of the today's advanced applications. The purpose of dealing with semantic heterogeneity is to overcome the problems connected with management of related data in a federation of systems. Another important issue is the representation of time and the capability of managing versioned data. This requirement arises from many application areas, such as CAD/CAM/CIM, legal and medical information systems, and from the re-engineering of old applications. The support of at least two kinds of time dimensions is widely emphasized in current literature [So91, TSC+93]: *transaction time*, which tells when an event is recorded in a database, and *valid time*, which represents when an event occurs, occurred or is expected to occur in the real world. According to this taxonomy, *snapshot* databases without time support, *transaction-* or *valid-time* databases supporting one time dimension (monotemporal) or *bitemporal* databases supporting both time dimensions can be defined [JCE+93]. In this work, without loss of generality, we consider temporal

relational databases with *interval* time-stamping at *tuple* level. The assumption of dealing with relational databases is not too restrictive, since different database systems may at least provide a relational interface towards the external world [LA86]. Mono- or bi-temporal relations can be defined by the addition of time attributes describing the endpoints of a transaction-time interval [IN, OUT) and of a valid-time interval [FROM, TO).

The problem of temporal semantic interoperability dealt with here consists of the sharing and exchange of information between relational databases of different temporal types or even between relations of different temporal types within the same system. Other kinds of temporal interoperability (e.g. between different time representations [JW93]) are not considered here. In a multidatabase environment, different sites can hold data based on different temporal models. For instance, most of the existing databases are snapshot but should be made operative in a temporal environment, as required in many application areas. In the context of a single database, the system may allow the coexistence of relations with a different temporal structure [Sn87]. For instance, *selective versioning* techniques can be used to partition attributes of a relation into groups with different update frequencies. Thus, storing them separately reduces the amount of storage space and can improve the overall database efficiency. Different groups should at least divide constant from time-varying attributes. Hence, owing to data semantics or application aims, one can design some relations as snapshot, some as monotemporal and some as bitemporal. It should be noticed that without selective versioning constant attribute values would be duplicated in all the versions of a tuple with time-varying attributes. Therefore, selective versioning can be considered as a further normalization step which can be applied to relations during the logical design of a temporal database (e.g. enforcement of *Time Normal Form* [NA89]). However, the reconstruction of unnormalized data from the resulting relations requires a sort of *internal* interoperability.

The common task of temporal interoperability problems is to provide a correct interaction between relations containing information with a different temporal nature. To this purpose, an algebra working on relations of different temporal types must be defined. In particular, we consider algebraic *select-project-join expressions*, on which temporal extensions of snapshot query languages (SQL and Quel) can be based [Ul88]. In the temporal case, select and project operators might include specific temporal counterparts (e.g. time-slice, temporal selection) but they do not present interoperability problems, since they work on one operand relation and usually produce a result relation of the same temporal type. Interoperability facilities are indeed required in the extension of the join operation, since operand relations may be of different temporal types. In this paper we only consider the *natural* join as the basic relational join operator; extensions of the results to the general θ-join and Cartesian product are straightforward. On the other hand, the join operation gives rise to interesting semantic and operative issues which will be discussed in this work. In general, the join operation, allowing data to be retrieved from distinct relations, reconstructs associations which are very meaningful from an application point of view, but which have been split among the relations by the normalization process. In a temporal database the reconstruction must be performed maintaining temporal synchronization of the data, otherwise meaningful temporal associations between contingent data might be lost and spurious temporal associations between non-contingent data could be created.

This paper is concerned with the two main problems connected with temporal interoperability: the translation of data from a temporal model into another and the execution of a temporal join in the presence of temporal incompleteness, when part of the information along some time dimension is missing. The former problem will be dealt with by the introduction of temporal conversion maps with the main purpose of obtaining operand relations with a common temporal format — not necessarily the format required for the result — before the execution of join operations. A final conversion is required if the temporal type of the result differs from the most convenient format chosen for the join execution. Our position on the second problem is to use the largest amount of *certain* temporal information stored in existing relations, and manage them in a lossless and prudent way. If one relation has information on a given time-span and

another has not, we want all the information coming from the first relation to be preserved in the join. It can be noticed that temporal incompleteness is usually present in temporal relations, where inserted data do not cover the whole time-universe. Moreover, also the temporal conversion process gives rise to incompleteness, when time dimensions are added. Our solution consists in completing relations (with *null* values) before the join execution. To this aim we present a simple algorithm for the completion of temporal information in a monotemporal or bitemporal relation. The mapping mechanism and the completion method together allow safe and lossless joins to be performed on temporal relations.

The temporal conversion maps are the subject of Section 2. In particular, we present a taxonomy of data with respect to their temporal semantics and to the temporal structure of the relations in which they are stored. On the basis of this classification, we present a method for mapping a *source* relation of every temporal type onto every other *target* type. In Section 3 we discuss the issue of temporal join and introduce an algorithm for the completion of temporal information. Furthermore, the problem of the execution order of the operations is briefly discussed, considering correctness and efficiency aspects.

The following notation and terminology are adopted:

- Valid-time semi-axis: $[t_0, t_\infty)$.
- Transaction-time semi-axis: $[T_0, T_\infty)$; the symbol "T_∞" has here the conventional meaning of "undefined."
- Current transaction-time: T_{now}.
- *History*: is the collection of all the temporal versions of an object (entity or relationship instance).
- In a transaction-time or bitemporal history, we define as *current* a tuple with OUT $= T_\infty$. A current tuple contains attribute values not yet modified by a transaction. We define as *archived* the non-current tuples, which contain (historical) attribute values modified by a transaction.
- The symbol "$\oplus$" indicates the operation of tuple concatenation:

$$(a_1, a_2, \ldots, a_m) \oplus (b_1, b_2, \ldots, b_n) = (a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n).$$

- r denotes the non-temporal portion of a tuple.
- In a bitemporal tuple, transaction-time attributes precede valid-time ones.

2 Temporal Conversion Maps

Let us start with a classification of the attributes with respect to their variability along the time dimensions (*intrinsic* variability). For a given time dimension, we can partition data into *time-variant* and *time-invariant*. A further subtle distinction may partition time-invariant data into *independent* or *dependent* on the existence of an object they describe. With respect to valid-time, all three kinds of data can be found. For example, the salary of an employee is a time-variant data, the mathematics constant "π" is an independent time-invariant data, whereas a date of birth is a dependent time-invariant data. Dependent time-invariant data have an associated existence time [EEK90], which begins with the initial time of the existence of the object they describe, and never ends. With respect to transaction-time, all the data stored in a database are actually time-variant, because they are at least subject to correction in case of error, unless modifications of their value are prohibited. These facts allow us to define three classes of data with respect to their variability along valid-time: the class V composed of time-Variant data, the class D of Dependent time-invariant data, and the class I of Independent time-invariant data (all of them are variant along transaction-time). This classification is summarized in Table I.

	Transaction Time	Valid Time
V	variant	variant
D, I	variant	invariant

Table I: Intrinsic variability of data.

It should be noted that, when V, D or I data are effectively stored in a temporal database, their kind of variability along the time dimensions can or cannot be represented, depending on the temporal format adopted for their storage. For instance, according to the above classification, any kind of data is properly variant with respect to transaction time, but a snapshot relation cannot represent such a variability, since only the last version of data is maintained. As a matter of fact, the variability of data as stored in a temporal database derives from the temporal model adopted (*extrinsic* variability) and is independent of the temporal nature which is proper of the data.

The intrinsic variability of data along a time dimension can be maintained, added or dropped in their database representation. For example, when a date of birth (of class D) is stored in a valid-time relation, the variability along valid time is added, whereas the variability along transaction time is dropped. Table II shows the temporal dimensions which are added, dropped or maintained for each class of data

	Snapshot	Transaction-time	Valid-time	Bitemporal
V	− −	= −	− =	= =
D, I	− =	= =	− +	= +

Table II: Representation of data variability in a database.

in each kind of temporal database. The first entry refers to transaction time and the second to valid time. The symbols "+," "−" and "=" stand for "added," "dropped" and "maintained," respectively. Whenever a given time dimension is "dropped", a loss of information occurs in the database, since the different data versions produced by the variability along the dropped time dimension are not effectively stored. When a given time dimension is "added," we assume that no spurious information is stored in the database: for instance, data of class I are represented by exactly one version even if memorized in a valid-time relation.

An advanced temporal system should automatically recognize the kind of data by means of catalogue information and manage them consequently, yet in general this is not the case. Therefore, the temporal conversion maps are defined in the following, assuming that the information about the temporal kind of data may be not available.

We introduce now the temporal conversion maps. An order relation "$\prec$" can be defined between different temporal data models as follows:

$X \prec Y$ iff the data model Y includes all the time-dimensions of X and not *vice versa*

All and only the relationships $S \prec T \prec B$ and $S \prec V \prec B$ hold, where S, T, V, B stand for snapshot, transaction-time, valid-time and bitemporal, respectively.

The notation $M_{XY} : X \longrightarrow Y$ indicates the generic conversion map. For example, $M_{SV} : S \longrightarrow V$ represents the map that translates snapshot relations into valid-time ones.

The maps M_{XY} for which $X \prec Y$ are temporal *extensions* and are: M_{SB}, M_{TB}, M_{VB}, M_{ST} and M_{SV}. The maps M_{XY} for which $Y \prec X$ are *contractions* and are: M_{BS}, M_{BT}, M_{BV}, M_{TS} and M_{VS}. The conversion maps which are neither extensions nor contractions (namely M_{TV} and M_{VT}) will be called *orthogonal* as valid- and transaction-time are *orthogonal* dimensions.

The most critical definitions concern *extension* maps, since some kind of temporal information — not contained in the stored data — must be created. Contraction maps require the selection of the part of temporal information — contained in the stored data — to be discarded, but no new temporal information is created. Orthogonal maps can be defined in a consistent way by means of extension and contraction maps.

2.1 Extension Maps

The extension maps are defined on the basis of the following general criteria:

E_1: By definition, snapshot (valid-time) relations only store current data, that is the current (historical) state of the data. Therefore, in an extension along transaction-time, such data can be assigned a $[T_{now}, T_\infty)$ time-stamp, where T_{now} is the transaction time when the extension map is applied. No extension is however possible along transaction-time beyond T_{now}. As a matter of fact, transaction time is defined by the system and, in any case, the past history of the updates is unknown when an extension is made along transaction time. Any conjecture on past update history is definitely arbitrary and unsafe.

E_2: By definition, the value (last value) stored in a snapshot (transaction-time) relation is considered to be valid in the present but can also be considered valid in the future, since it cannot even be forecasted if a modification will occur or not. Therefore, in an extension along valid time, such data can be assigned a $[t_{now}, t_\infty)$ time-stamp, where t_{now} equals the transaction time T_{now} when the extension map is applied. If available, the information that some data is independent time-invariant (I), allows us to extend its validity to the whole valid time semi-axis $[t_0, t_\infty)$.

E_3: Transaction-time databases are normally used in the applications to represent a sort of past history of time-variant (V) data, even if such history is not explicitly managed by the user (as happens in databases with valid time) and only summarizes the sequence of updates. However, following this natural interpretation, data versioned along transaction-time only, can be extended along valid-time for synchronous values of the two time axes. In other words, the data versioning managed by the system along transaction time can also be reflected on the valid-time axis, reconstructing the view a user had throughout the database life, that is the bitemporal data portion seen by a *diagonal user* (for the concept of *user* see [BG93]).

When the catalogue information that data are time-invariant (D or I) is available, (E_3) cannot be applied in order to avoid the creation of spurious information. For the sake of simplicity, in the map definitions that follow, time-invariant data are assumed to be *dependent* (D, which is the most common case). *However, for all the maps which add the valid-time dimension* (namely extension maps M_{SV}, M_{SB}, M'_{TB}, and the orthogonal map M'_{TV}), *if the data type is actually known to be independent time-invariant (I) then the whole semi-axis* $[t_0, t_\infty)$ *rather than* $[t_{now}, t_\infty)$ *can be used in the maps.*

2.1.1 Snapshot to Transaction-Time $M_{ST} : S \longrightarrow T$

Thanks to (E_1), the extension can be defined as:

$$M_{ST}(r) = r \oplus (T_{now}, T_\infty) .$$

2.1.2 Snapshot to Valid-Time $M_{SV} : S \longrightarrow V$

According to (E_2), the extension is given by:

$$M_{SV}(r) = r \oplus (t_{now}, t_\infty) .$$

2.1.3 Snapshot to Bitemporal $M_{SB} : S \longrightarrow B$

As follows from criteria (E_1) and (E_2), the bitemporal extension of r is defined as:

$$M_{SB}(r) \;=\; r \oplus (T_{now}, T_\infty, t_{now}, t_\infty) \;.$$

2.1.4 Transaction-Time to Bitemporal $M_{TB} : T \longrightarrow B$

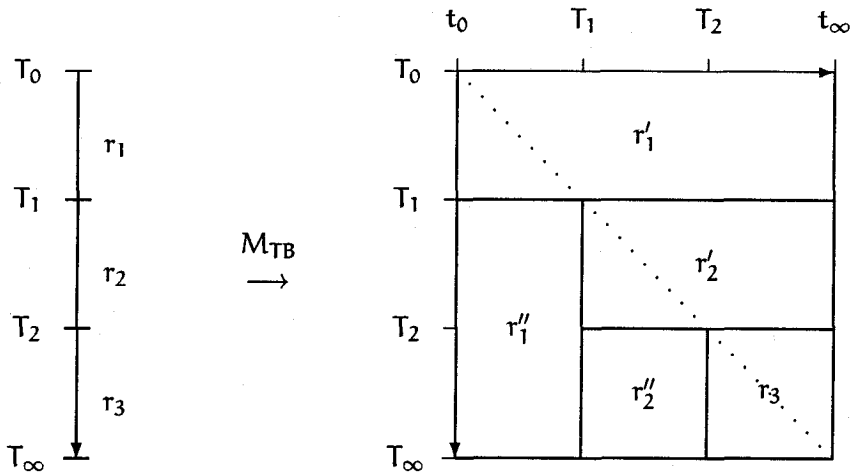

Figure 1: Transaction-time to Bitemporal M_{TB}

According to (E_3), the extension of a transaction-time relation along valid time can be made in a less restrictive way than in the previous cases, assuming data in r to be time-variant (V). A tuple can be considered valid from the time it was inserted, that is from its IN value. As far as the end of validity is concerned, a distinction must be made between current and archived tuples.

A current tuple (e.g. r_3 in Fig. 1) can be considered indefinitely valid from its IN value, since it cannot be forecasted if a modification transaction would ever arrive and archive such tuple. Therefore, in this case, r can be assigned a $[IN, T_\infty) \times [IN, t_\infty)$ time-stamp by the map. Notice that only one current tuple per history can exist.

The already archived tuples (e.g. r_1 and r_2 in Fig. 1) can be considered to be created by a transaction with effect in $[IN, T_\infty) \times [IN, t_\infty)$ and modified by a successive transaction with effect in $[OUT, T_\infty) \times [OUT, t_\infty)$, which limits the initial time pertinence of the tuple to an "L-shaped" region that is symmetric with respect to the diagonal. Such a region can easily be covered by two non-overlapping tuples (e.g. r'_1-r''_1 for the extension of r_1 and r'_2-r''_2 for the extension of r_2 in Fig. 1).

Therefore, the complete map for the bitemporal extension of $r \oplus (IN, OUT)$ can be defined as:

$$M_{TB}(r \oplus (IN, OUT)) \;=\; \begin{cases} r \oplus (IN, T_\infty, IN, t_\infty) & \text{if } OUT = T_\infty \\ r \oplus (IN, OUT, IN, t_\infty) \cup r \oplus (OUT, T_\infty, IN, OUT) & \text{otherwise} \end{cases} \;.$$

If catalogues maintain information about the temporal nature of data, and data in r are time-invariant, (E_3) cannot be safely applied. In this case, according to (E_2), the only safe extension is given by:

$$M'_{TB}(r \oplus (IN, OUT)) \;=\; r \oplus (IN, OUT, t_{now}, t_\infty) \;.$$

2.1.5 Valid-Time to Bitemporal $M_{VB} : V \longrightarrow B$

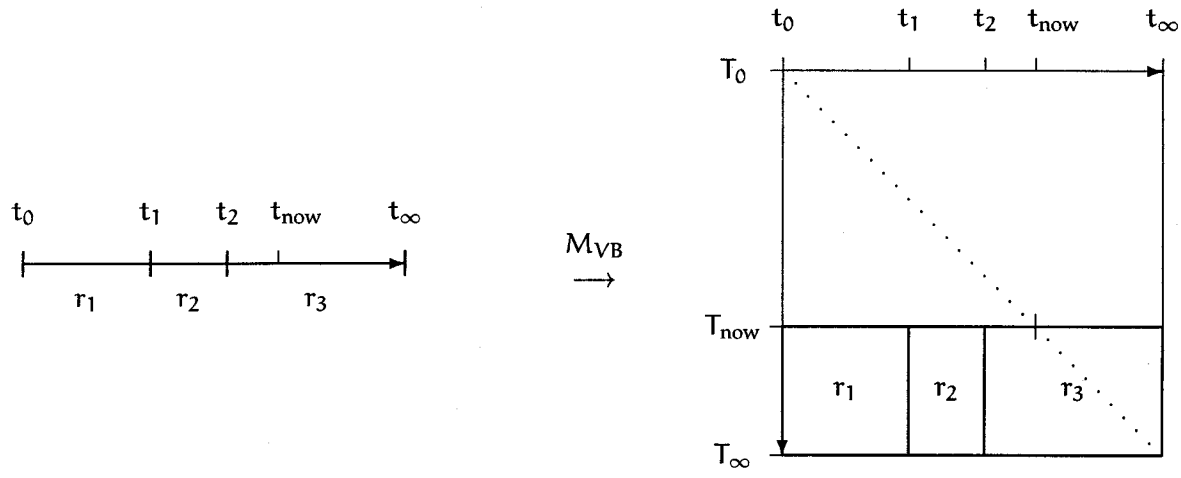

Figure 2: Valid-Time to Bitemporal M_{VB}

According to (E_1), the bitemporal extension of valid-time relations is given by:

$$M_{VB}(r \oplus (\mathrm{FROM}, \mathrm{TO})) \;=\; r \oplus (T_{now}, T_\infty, \mathrm{FROM}, \mathrm{TO}) \;.$$

The effects of the map are shown in Fig. 2.

2.2 Contraction Maps

The contraction counterparts of the extension principles can be formulated as follows. Provided that the most general and comprehensive representation of data is bitemporal, and bitemporal data can be represented on a rectangular time-domain obtained as a Cartesian product of the time semi-axes, we have:

- C_1: to drop transaction-time means to extract from the rectangular domain the data laying on the straight line $T = T_\infty$ parallel to the valid-time axis;
- C_2: to drop valid-time means to extract the data laying on the principal diagonal $T = t$ of the time-domain up to T_{now} (no "future" transaction time different from $\mathrm{OUT} = T_\infty$ must be generated);
- C_3: as follows from the concurrent application of (C_1) and (C_2), to drop both time dimensions means to extract the data (snapshot) on the point of coordinates (T_{now}, t_{now}). Moreover, the contractions from transaction- or valid-time to snapshot are trivial.

The contraction map definitions that follow, which are based on the above principles, are consistent with the extension maps. As a matter of fact, if the symbol "$\circ$" denotes the composition of operations and M_{XY} (with $X \prec Y$) is a generic extension map, it can easily be shown that:

$$M_{XY} \circ M_{YX} = \mathrm{Id} \;,$$

where M_{YX} is the corresponding contraction map and Id is the identity map.

2.2.1 Bitemporal to Valid-Time $M_{BV} : B \longrightarrow V$

According to (C_1), the map is defined as:

$$M_{BV}(r \oplus (IN, OUT, FROM, TO)) = \begin{cases} r \oplus (FROM, TO) & \text{if } OUT = T_\infty \\ \emptyset & \text{otherwise} \end{cases}.$$

If the condition $OUT = T_\infty$ is false, the information content of r is not preserved in the contraction result.

2.2.2 Bitemporal to Transaction-Time $M_{BT} : B \longrightarrow T$

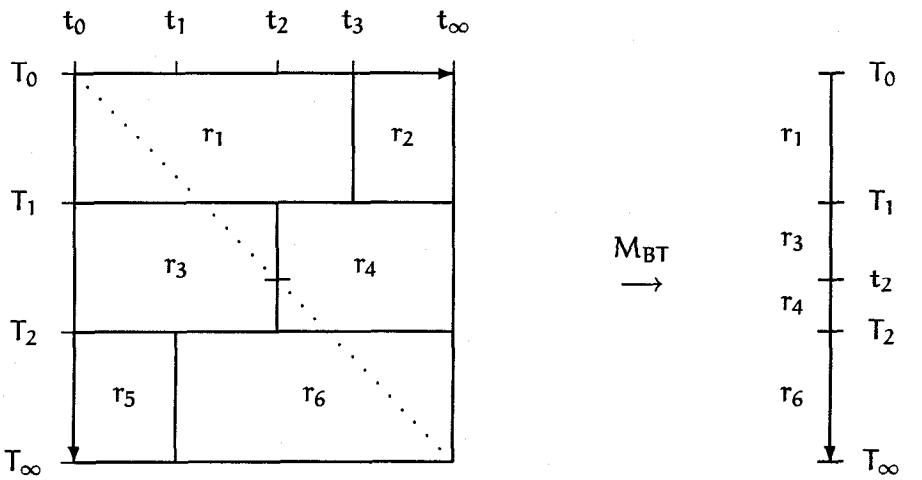

Figure 3: Bitemporal to Transaction-time M_{BT}

The mapping is defined according to (C_2). Only the tuples intersecting the time diagonal $T = t$ contribute to the map result. A necessary and sufficient condition for a tuple to intersect the diagonal is that its interval time-stamps $[IN, OUT)$ and $[FROM, TO)$ overlap. This is equivalent to the condition $\max\{IN, FROM\} < \min\{OUT, TO\}$. The transaction-time interval to be assigned to the resulting tuple can be computed as the projection, on one of the time axes, of the diagonal portion intercepted by the bitemporal tuple (see Fig. 3). It can easily be verified that this interval is given by $[\max\{IN, FROM\}, \min\{OUT, TO\})$. Therefore, the map can be defined as:

$$M_{BT}(r \oplus (IN, OUT, FROM, TO)) = \begin{cases} r \oplus (\max\{IN, FROM\}, \min\{OUT, TO\}) & \text{if } \max\{IN, FROM\} < \min\{OUT, TO\} \\ \emptyset & \text{otherwise} \end{cases}.$$

If the tuple does not intercept the diagonal, its information content r is not preserved in the result.

2.2.3 Bitemporal to Snapshot $M_{BS} : B \longrightarrow S$

According to (C_3), the only data contained in the current tuples that are present with respect to valid time are preserved in the mapping:

$$M_{BS}(r \oplus (IN, OUT, FROM, TO)) = \begin{cases} r & \text{if } OUT = T_\infty \wedge FROM \leq t_{now} < TO \\ \emptyset & \text{otherwise} \end{cases}.$$

If the condition $OUT = T_\infty \wedge FROM \leq t_{now} < TO$ is false, the contraction does not preserve the information in r.

2.2.4 Valid-Time to Snapshot $M_{VS} : V \longrightarrow S$

According to (C_3), we have:

$$M_{VS}(r \oplus (FROM, TO)) = \begin{cases} r & \text{if } FROM \leq t_{now} < TO \\ \emptyset & \text{otherwise} \end{cases} .$$

If the condition $FROM \leq t_{now} < TO$ is false, the contraction does not preserve the information in r.

2.2.5 Transaction-Time to Snapshot $M_{TS} : T \longrightarrow S$

According to (C_3), we have:

$$M_{TS}(r \oplus (IN, OUT)) = \begin{cases} r & \text{if } OUT = T_\infty \\ \emptyset & \text{otherwise} \end{cases} .$$

If the condition $OUT = T_\infty$ is false, the contraction does not preserve the information in r.

2.3 Orthogonal Maps

The orthogonal maps are not primitive since they can be defined in terms of the extension and contraction maps, according to the following principle:

O_1: The orthogonal maps can be defined as the composition of the extension from the source format to the bitemporal one, followed by the contraction to the target format, that is: $M_{VT} = M_{VB} \circ M_{BT}$ and $M_{TV} = M_{TB} \circ M_{BV}$.

Such compositions give rise to the definitions which follow.

2.3.1 Valid- to Transaction-Time $M_{VT} : V \longrightarrow T$

According to (O_1), we have:

$$M_{VT}(r \oplus (FROM, TO)) = \begin{cases} r \oplus (T_{now}, T_\infty) & \text{if } FROM \leq t_{now} < TO \\ \emptyset & \text{otherwise} \end{cases} .$$

If the condition $FROM \leq t_{now} < TO$ is false, the map does not preserve the information in r.

2.3.2 Transaction- to Valid-Time $M_{TV} : T \longrightarrow V$

As follows from (O_1), we have:

$$M_{TV}(r \oplus (IN, OUT)) = r \oplus (IN, OUT) .$$

When the data are known to be time-variant (V), the extension map to be used in the composition is M'_{TB} instead of M_{TB}, yielding:

$$M'_{TV}(r \oplus (IN, OUT)) = \begin{cases} r \oplus (t_{now}, t_\infty) & \text{if } OUT = T_\infty \\ \emptyset & \text{otherwise} \end{cases} .$$

If the condition $OUT = T_\infty$ is false, the map does not preserve the information in r.

3 Temporal Natural Join with Incomplete Information

Let us define a *history* as a set of tuples with the same (snapshot) key value in a relation [GST91]. A history represents the collection of all the versions — memorized as tuples — of an *object* [JCE+93]. We denote by object a single *entity* or *relationship* instance. Histories cannot contain different tuples overlapping in time since attribute values, at any given time, uniquely depend on the key value. By the term *temporal incompleteness* of the information concerning an object, we mean that the tuples of the object history do not completely cover the time-universe (e.g. $[T_0, T_\infty) \times [t_0, t_\infty)$ in the bitemporal model).

The *temporal natural join* — which is the temporal counterpart of the snapshot operator used for the reconstruction of lossless decompositions — must operate with synchronized time attributes. Algorithms for temporal join, along one time dimension, can be found in the literature (e.g. Event-Join), where a unique surrogate (which always acts as a key) is considered as a join attribute [SG89]. Tuples are joined if their surrogates match and time-intervals overlap. The intersection of the time-intervals is the time-stamp assigned to the resulting join tuple. As far as incomplete relations are concerned, the algorithms proposed for the Event-Join provide the *run-time detection* of portions of histories not covered by matching histories in the other relation. For such regions, outerjoin tuples (with *null* values) are produced. Smart algorithms, exploiting the ordering of the relations or using additional data structures, have been proposed.

In the most general case of *bitemporal natural join*, there are two main differences with respect to Event-Join. First, a total ordering cannot be defined on bitemporal intervals and complex data structures (e.g. R-trees [Gu84] instead of B-trees) should be used to efficiently represent covered/uncovered time regions. Second, the join attributes are not necessarily *candidate keys* of one or both of the participating relations. For instance, in the (snapshot) natural join:

$$R_1(\underline{A}, \underline{B}, P_1, \ldots, P_m) \bowtie_S R_2(\underline{A}, \underline{C}, Q_1, \ldots, Q_n) \ ,$$

we have many-to-many relationships between A and B values in R_1 and between A and C values in R_2. In the particular case of temporal natural join between relations containing incomplete histories, multiple coverings must be considered in order to correctly generate "outerjoin" tuples, for each matching value of the join attributes. This operation may require a large main memory buffer to keep track of all the uncovered time-regions during the join execution. History completion in single relations (before the join execution) avoids this problem, since only single coverings must be considered.

3.1 History Completion

If the relations to be joined contain *complete* histories only, no "outerjoin" tuples must be generated. In this case, relations can be processed by examining each combination of tuples of the operand relations only once, and a "standard" algorithm for the extraction of tuple pairs can be used (e.g. "nested loop," if more convenient methods are not available).

If the relations do not only contain complete histories, the join operation must be preceded by a *completion phase* where uncovered time regions of incomplete histories are filled up with *all-null* tuples. An all-null tuple has all the non-temporal attributes but the key set to *null*, with the meaning that no information is available in the time-span defined by the time-stamps of the tuple. In a monotemporal model, the uncovered regions are intervals which can simply be used to time-stamp the all-null tuples to be added. In the bitemporal model, the uncovered regions must be decomposed into rectangles in order to time-stamp the all-null tuples to be added (see Fig. 4). A Pascal-like description of the history completion algorithm is presented below. If the temporal type of the relation includes transaction-time, the relation is assumed to be ordered on the IN attribute (which occurs if tuples are clustered following their natural order of insertion). An efficient implementation of the algorithm also requires the histories to

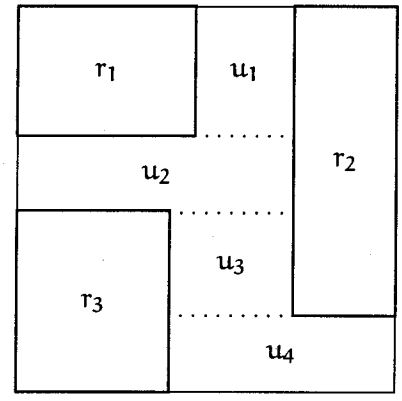

Figure 4: An incomplete history (formed by r_1, r_2, r_3) with uncovered regions decomposed into rectangles (u_1, u_2, u_3, u_4).

be processed separately, but not necessarily in key order. To this purpose, the relation can be reorganized at run-time by means of a *hash function* on the key. A sequential scan of the original relation allows the ordering on IN to be preserved as a secondary ordering after the hashing, as required by the algorithm.

3.2 Algorithm for history completion

- R denotes the relation to be completed.
- The variable u denotes the tuple of R currently selected.
- The expression $u.\mathcal{I}$ denotes the (bi)temporal interval of the tuple u.
- The variable $\mathcal{T}$ denotes a temporal interval.
- The variable NC denotes a set of temporal intervals and is used to collect the intervals not yet covered on the currently processed history.
- The function NotCov$(\mathcal{T}, u)$ returns the set of temporal intervals consisting of the regions of the temporal interval $\mathcal{T}$ not covered by the interval time-stamp of u.

```
for each key value K in R do
    NC := {full time domain}      /* e.g. {[T0, T∞) × [t0, t∞)} for a bitemporal relation */
    for each tuple u with key K in R do
        for each T in NC do
            if T.OUT ≤ u.IN then    /* only if R is ordered on attribute IN */
               generate an all-null tuple s with key K and s.I = T
               NC := NC \ {T}
            end if
            if u.I overlaps T then NC := NC \ {T} ∪ NotCov(T, u)
        end for
    end for
    for each T in NC do
        generate an all-null tuple s with key K and s.I = T
    end for
end for
```

In the bitemporal case, the NotCov function can be computed as:

$$\begin{aligned}\text{NotCov}(\mathcal{T},u) \;=\; & \{\mathcal{I} \mid \mathcal{T}.\text{IN} < u.\text{IN} \wedge \mathcal{I} = [\mathcal{T}.\text{IN}, u.\text{IN}) \times [\mathcal{T}.\text{FROM}, \mathcal{T}.\text{TO})\} \\ \cup\; & \{\mathcal{I} \mid \mathcal{T}.\text{FROM} < u.\text{FROM} \\ & \qquad \wedge \mathcal{I} = [\max\{u.\text{IN}, \mathcal{T}.\text{IN}\}, \min\{u.\text{OUT}, \mathcal{T}.\text{OUT}\}) \times [\mathcal{T}.\text{FROM}, u.\text{FROM})\} \\ \cup\; & \{\mathcal{I} \mid u.\text{TO} < \mathcal{T}.\text{TO} \\ & \qquad \wedge \mathcal{I} = [\max\{u.\text{IN}, \mathcal{T}.\text{IN}\}, \min\{u.\text{OUT}, \mathcal{T}.\text{OUT}\}) \times [u.\text{TO}, \mathcal{T}.\text{TO})\} \\ \cup\; & \{\mathcal{I} \mid u.\text{OUT} < \mathcal{T}.\text{OUT} \wedge \mathcal{I} = [u.\text{OUT}, \mathcal{T}.\text{OUT}) \times [\mathcal{T}.\text{FROM}, \mathcal{T}.\text{TO})\}\ .\end{aligned}$$

In a monotemporal case, being START = IN (FROM) and END = OUT (TO) for transaction (valid) time, we simply have:

$$\begin{aligned}\text{NotCov}(\mathcal{T},u) \;=\; & \{\mathcal{I} \mid \mathcal{T}.\text{START} < u.\text{START} \wedge \mathcal{I} = [\mathcal{T}.\text{START}, u.\text{START})\} \\ \cup\; & \{\mathcal{I} \mid u.\text{END} < \mathcal{T}.\text{END} \wedge \mathcal{I} = [u.\text{END}, \mathcal{T}.\text{END})\}\ .\end{aligned}$$

3.3 Execution Order of Operations

In this section the order of execution of operations and conversion maps is briefly discussed. Some constraints are introduced in order to perform safe operations.

On the basis of what has been discussed so far, a first rule is to perform the completion phase *before* the join execution. However, it can be shown that the history completion and the conversion maps can be applied in any order. Furthermore, *optimized* extension maps could be defined, so that all-null tuples covering missing information along an added time dimension can directly be generated during the conversion process.

In general, if R_1 and R_2 are two relations with temporal format X_1 and X_2, respectively, their join can be effected as:

$$M_{JY}(M_{X_1J}(R_1) \bowtie_J M_{X_2J}(R_2))\ ,$$

where J is the common temporal format adopted for the join execution and Y is the temporal format desired for the final result. The following constraints must be imposed on J in order to avoid loss of information:

$$X_1 \preceq J, X_2 \preceq J \quad \text{or} \quad Y \preceq J\ .$$

If the first two conditions are not met, we may have loss of information along a time dimension which was present in R_1 or R_2 and which is required in the final result. In this case some information, originally stored in R_1 or R_2, may be lost during the application of the maps M_{X_1J} and M_{X_2J} or during the join execution. However, if the condition $Y \preceq J$ is verified, such information can be discarded since it is not required for the result. In other words, *time dimensions required for the result, and already present in the operand relations, must never be eliminated.*

For instance, if R_1 is snapshot (S), R_2 is valid-time (V) and the result needs to be bitemporal (B), a correct choice is J = V and a convenient operation sequence is therefore:

$$M_{VB}(M_{SV}(R_1) \bowtie_V R_2)\ .$$

If the result needs to be snapshot, a more convenient correct choice is J = S, yielding:

$$R_1 \bowtie_S M_{VS}(R_2)\ .$$

In the former case $X_1 \preceq J, X_2 \preceq J$ holds, whereas in the latter case $Y \preceq J$ holds.

Assuming that a snapshot join is less expensive than a monotemporal one and that a monotemporal join is less expensive than a bitemporal one, because of the different amount of data to be usually processed, the following procedure can be used as a heuristic approach to the optimization of a correct temporal natural join with conversion maps:

- Do the operand relations include time-dimensions not included in the temporal format Y of the result?
  - **Yes:** Drop them.
- Are the operand relations of the same temporal type?
  - **Yes:** Execute the join and then convert the result to Y (if necessary).
  - **No:** Convert the relation(s) to Y and then execute the join.

4 Conclusions

The cooperative use of heterogeneous temporal databases offers a new and attractive perspective for temporal applications and provides several challenging issues in the field of database semantic interoperability. Morevover, a single database may require a sort of internal interoperability, since design techniques (e.g. selective versioning) may lead to relations with different temporal format.

In this paper three main problems have been faced. The first concerns the translation of relational data from one temporal model into another. This problem has been solved by means of the definition of temporal conversion maps. The maps proposed can be used as a general interface for data exchange between different temporal models, and provide the preliminary step for joining relations with different temporal format. The second problem concerns the extension of the temporal natural join operation in order to work in the presence of incomplete information. A general algorithm has been introduced for the elimination of temporal incompleteness before the execution of a join. After the completion, the join can be executed by means of standard algorithms. The third problem concerns the constraints to be imposed on the execution order of conversion, join, and completion operations in order to obtain correct results without loss of information. Among correct operation sequences, a global procedure for join execution which tries to minimize the complexity of the operations has been proposed. Although the general problem of query processing and optimization in a temporal heterogeneous environment is far from being solved, future work may take advantage of the solutions outlined in this work.

References

[BG93] Bhargava G., Gadia S.K., "Relational Database Systems with Zero Information Loss," IEEE Trans. on Knowledge and Data Engineering, Vol. 5, No. 1, Feb. 1993.

[EEK90] Elmasri R., El-Assal I., Kouramajian V., "Semantics of Temporal Data in an Extended ER Model," Proc. Intl. Conf. on E-R Approach, Lausanne, 1990.

[EP90] Elmagarmid A.K., Pu C. (eds.), Special Issue on Heterogeneous Databases of ACM Computing Surveys, Vol. 22, No. 3, Sept. 1990.

[GST91] Grandi F., Scalas M.R., Tiberio P., "A History Oriented Data View and Operation Semantics for Temporal Relational Databases," C.I.O.C.-C.N.R. Tech. Rep. No. 76, Bologna, Jan. 1991.

[Gu84] Guttman A., "The R-trees: A Dynamic Index Structure for Spatial Searching," Proc. of ACM SIGMOD Conf., Boston, 1984.

[JW93] Jajodia S., Wang X.S., "Temporal Mediators: Supporting Uniform Access to Heterogeneous Temporal Databases," Proc. Workshop on Interoperability of Database Systems and Database Applications, Fribourg, 1993.

[JCE+93] Jensen C.S., Clifford J., Elmasri R., Gadia S.K., Hayes P., Jajodia S. (eds.), et al., "A Consensus Glossary of Temporal Database Concepts," in *Final Report* of Intl. Workshop on an Infrastructure for Temporal Databases, 1993.

[LA86] Litwin W., Abdellativ A., "Multidatabase Interoperability," IEEE Computer, Dec. 1986.

[NA89] Navathe S.B., Ahmed R., "A Temporal Relational Model and a Query Language," Information Sciences, Vol. 49, 1989.

[SG89] Segev A., Gunadhi H., "Event-Join Optimization in Temporal Relational Databases," Proc. of Intl. VLDB Conf., Amsterdam, 1989.

[Sh91] Sheth A.P. (ed.), Special Issue on Semantic Issues in Multidatabase Systems, ACM SIGMOD Record, Vol. 20, No. 4, Dec. 1991.

[Sn87] Snodgrass R.T., "The Temporal Query Language TQuel," ACM Trans. on Database Systems, Vol. 12, No. 2, June 1987.

[So91] Soo M., "Bibliography on Temporal Databases," ACM SIGMOD Record, Vol. 20, No. 1, Mar. 1991.

[TSC+93] Tansel A., Snodgrass R.T., Clifford J., Gadia S.K., Segev A. (eds.), *Temporal Databases, Theory, Design and Implementation*, Benjamin-Cummings, 1993.

[Ul88] Ullman J.D., *Principles of Database and Knowledge-Base Systems*, Vol. I, Computer Science Press, 1988.