Querying Data with Multiple Temporal Dimensions

Carlo Combi[1] and Angelo Montanari[2]

[1] Dipartimento di Informatica, Università degli Studi di Verona
Ca' Vignal 2, strada le Grazie, 15, I-37134 Verona, Italy
combi@sci.univr.it
[2] Dipartimento di Matematica e Informatica, Università degli Studi di Udine
Via delle Scienze, 206, I-33100 Udine, Italy
montana@dimi.uniud.it

Abstract. This paper focuses on the problem of designing a query language for data models with multiple temporal dimensions.

1 Data Models with Multiple Temporal Dimensions

Valid and transaction times are commonly recognized as the basic (orthogonal) temporal dimensions for data [2], and a variety of issues related these two dimensions have been systematically explored in the literature [3]. In parallel, a considerable effort has been devoted to understand whether or not valid and transaction times suffice to capture all relevant temporal aspects in a natural and efficient way. In [1] we defined a new conceptual data model with four temporal dimensions that refines and extends a number of previous proposals. Besides valid and transaction times, such a model includes event and availability times. The *event times* of a fact are the occurrence times of the events that respectively initiate and terminate its validity interval. The *availability time* of a fact is the time interval during which the fact is available to and believed correct by the information system (such an interval does not necessarily coincide with the transaction time interval of the fact). In this paper, we focus on basic problems involved in querying data with multiple temporal dimensions. The outcome is the identification of the *general requirements* that a query language must satisfy to allow one to manage multiple temporal dimensions.

2 Querying Data with Multiple Temporal Dimensions

The management of multiple temporal dimensions has an impact on many components of query languages. In this section, we restrict our attention to those aspects specifically related to queries involving the four temporal dimensions. To exemplify the features of the query language, we will consider (a portion of) a clinical database, consisting of two tables, namely, *pat_sympt* and *pat_ther*, which contain data on patients' symptoms and therapies, respectively. Table schema is as follows:

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 711–714, 2002.

pat_sympt(P_id, symptom, VT, ET_i, ET_t, AT, TT)
pat_ther(P_id, therapy, VT, ET_i, ET_t, AT, TT)
where the attributes VT, ET_i, ET_t, AT, and TT respectively denote the valid time, the initiating event time, the terminating event time, the availability time, and the transaction time of stored facts.

WHERE and WHEN clauses. Comparisons between temporal dimensions can be supported in two alternative ways [3]: the first one is to deal with temporal and non-temporal data in the WHERE clause; the second one is to constrain purely temporal conditions to appear in an ad-hoc WHEN clause. Obviously, to make it possible to check conditions mixing temporal and non-temporal data, we must allow temporal dimensions to occur, whenever necessary, in the WHERE clause, no matters what option we choose. In case we opt for the second alternative (this allows one to exploit specific tools to manage temporal data, at logical and physical levels), another design issue is the specification of the temporal dimensions we can refer to in the WHEN clause. The most general option is to allow all the four temporal dimensions to be used in the WHEN clause. According to this choice, a query that returns all and only the patient's symptoms that appeared no later than 3 days after their initiating event can be formulated as follows:

```
SELECT symptom
FROM pat_sympt S
WHEN BEGIN(VALID(S)) - INITIATING_ET(S) ≤ 3
```

where INITIATING_ET(·), VALID(·), and BEGIN(·) are functions that return the initiating event time of a tuple, the valid time interval of a tuple, and the starting point of a given interval, respectively.

Querying past states. By default, a query is evaluated with respect to the current state of the database (and thus to the knowledge currently available to the information system). However, there must be the possibility of evaluating a query over both the current and the past states of the database and/or of the information system. If we are interested in querying the database about its past states, we can use the well-known AS OF clause. We add an AS KNOWN clause that allows the user to query the database about the knowledge which was available to the information system at given time points in the past. We may also need to jointly use both clauses. As an example, the following query determines which data was available to the physician on October 18, according to the database contents on October 20:

```
SELECT *
FROM pat_sympt S
AS OF 97Oct20
AS KNOWN 97Oct18
```

Temporal joins. Some issues arise when more than one relation comes into play in a query. For example, we have to assign a proper meaning to the cartesian product of the relations that appear in the FROM clause. A possible choice is to

impose that temporal tuples of different relations can be associated only when both their transaction times and their availability times overlap. Even though this solution seems meaningful, it does not allow the users to formulate some queries, such as, for instance, hypothetical queries where information available at different times has to be considered. In general, the language should allow the user to choose among different ways of executing the cartesian product between temporal relations (one can even opt for the usual cartesian product of relations, where the temporal attributes are treated as the standard ones).

Furthermore, besides the standard (atemporal) join, the language should support some forms of temporal join, where the join condition allows the user to join tuples taking into account different temporal features. As an example, consider a scenario where the physician wants to determine the symptoms of patients for which the valid time intervals of symptoms and therapies intersect. The query can be formulated as follows:

```
SELECT symptom, S.P_id
FROM pat_sympt S, pat_ther T
WHEN NOT(VALID(S) BEFORE VALID(T)) AND
   NOT(VALID(T) BEFORE VALID(S))
WHERE S.P_id = T.P_id
```

The `WHEN` and `WHERE` clauses can actually be replaced by a suitable join in the `FROM` clause, as shown by the following equivalent formulation of the query:

```
SELECT symptom, S.P_id
FROM pat_sympt S TJOIN pat_ther T ON S.P_id = T.P_id AND
   NOT(VALID(S) BEFORE VALID(T)) AND NOT(VALID(T) BEFORE VALID(S))
```

where we used the keyword `TJOIN` in the `SELECT` clause to point out that the join involves the temporal dimensions of information.

Further complex temporal queries can be defined when the different temporal dimensions have to be considered together. For example, for any given patient and symptom, the following query determines the therapies decided after that the information system became aware of the symptom and started before the end of the symptom:

```
SELECT S.P_id, therapy, symptom
FROM pat_sympt S JOIN pat_ther T ON S.P_id = T.P_id
WHEN INITIATING_ET(T) AFTER BEGIN(AVAILABLE(S)) AND
    BEGIN(VALID(T)) BEFORE END(VALID(S))
```

where `AVAILABLE(·)` returns the availability time of the considered tuple.

Defining temporal dimensions of query results. When several temporal relations are involved in a query, the language must provide some mechanisms to allow the user to obtain a consistent result, that is, a temporal relation endowed with valid, transaction, event, and availability times[1].

[1] In this paper, we consider only temporal databases where each relation has all the four temporal dimensions. As in the case of valid and transaction databases, one can

While transaction and availability times of the resulting relation can be obtained (according to the standard definition of the cartesian product given above) as the intersection of the transaction times and the availability times of the considered tuples, respectively, valid and event times are often explicitly defined by the user. In order to properly evaluate a query, however, some default criteria must be specified to determine the values of the valid and event times of the resulting relation, whenever the user does not provide any explicit rule. As an example, the valid time of each tuple belonging to the resulting relation can be defined as the intersection of the valid times of the corresponding tuples belonging to the relations that occur in the `FROM` clause, while its initiating and terminating event times can be defined as the maximum of the initiating and terminating event times of the corresponding tuples, respectively. However, we expect that, in most cases, the user explicitly defines the way in which the valid and event times of the resulting relation must be computed, taking into account the meaning he/she assigns to the query and the relative data. As an example, in the following query the user assigns to the valid and event times of the resulting relation the values that these attributes have in the *pat_sympt* relation:

```
SELECT symptom, S.P_id WITH VALID(S) AS VALID, INITIATING_ET(S)
    AS INITIATING, TERMINATING_ET(S) AS TERMINATING
FROM pat_sympt S TJOIN pat_ther T ON S.P_id = T.P_id AND
   NOT(VALID(S) BEFORE VALID(T)) AND NOT(VALID(T) BEFORE VALID(S))
```

3 Conclusions

This paper focused on the design of a query language for temporal databases with multiple temporal dimensions. Even though we can debate whether or not a database system needs to provide a special support, at the logical/physical level, to manage multiple temporal dimensions, it is not controversial that these dimensions must be taken into consideration at the level of requirements engineering applied to the design of information systems.

References

1. C. Combi and A. Montanari. Data Models with Multiple Temporal Dimensions: Completing the Picture. In K. R. Dittrich, A. Geppert, M. C. Norrie (eds.), Advanced Information Systems Engineering, 13th International Conference, (CAiSE 2001). LNCS 2068, Springer, Berlin Heidelberg, 187–202, 2001. 711
2. C. Jensen, C. Dyreson (Eds.) et al. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In *Temporal Databases - Research and Practice*, LNCS 1399, Springer, Berlin Heidelberg, 367–405, 1998. 711
3. G. Özsoyoglu and R. T. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4): 513–532, 1995. 711, 712

provide suitable clauses and/or keywords to allow the user to obtain relations with a proper subset of temporal dimensions.