The Chronus II Temporal Database Mediator

Martin J O'Connor MSc, Samson W Tu MS, Mark A Musen MD PhD
Stanford Medical Informatics, Stanford University School of Medicine,
Stanford, CA 94305-5479

Clinical databases typically contain a significant amount of temporal information. This information is often crucial in medical decision-support systems. Although temporal queries are common in clinical systems, the medical informatics field has no standard means for representing or querying temporal data. Over the past decade, the temporal database community has made a significant amount of progress in temporal systems. Much of this research can be applied to clinical database systems. This paper outlines a temporal database mediator called Chronus II. Chronus II extends the standard relational model and the SQL query language to support temporal queries. It provides an expressive general-purpose temporal query language that is tuned to the querying requirements of clinical decision support systems. This paper describes how we have used Chronus II to tackle a variety of clinical problems in decision support systems developed by our group.

Introduction

Questions relating to time are pervasive in medical decision-making. Common queries include simple questions of duration: "How long did a particular condition last?" Queries relating to temporal order include: "Did the condition occur after the administration of a particular drug?" "When was the most recent occurrence?" These queries can be extended to include some contextual information: "Did the condition occur during the administration of a particular protocol?" Queries relating to numbers of occurrences are common: "Were there at least three occurrences of the condition lasting over a week?" As interactions between clinical decision support systems and databases grow more complex, these temporal queries can become very elaborate[1].

Although temporal queries are common in clinical decision-making, the medical informatics field has no standard means for making them. A number of systems address this shortcoming. One of the first systems to address the time representation problem was the Time Oriented Database (TOD)[2]. TOD had a three-dimensional view of clinical data, with time represented explicitly as one of the dimensions. Data relating to a particular patient visit, for example, were indexed by patient identifier, clinical parameter type, and visit time. TOD supported a set of basic temporal queries that allowed data values following certain temporal patterns to be extracted. The contemporary Arden Syntax[3] also supports a basic temporal instant-based temporal representation.

Both TOD and the Arden Syntax model time by associating an *instant timestamp* with particular records. An instant timestamp permits a range of simple temporal questions about associated data, such as "Was the patient HIV-positive prior to the visit date?" or "Did the patient receive tamoxifen in the past week?" However, associating an *interval timestamp* with data enables more complex queries. Interval timestamps are composed of a start timestamp and a stop timestamp, and delimit the interval of time during which the data are held to be true. This interval is often referred to as the data's *valid-time range*. Most recent temporal database systems focus on this type of temporal information[4]. These databases are termed *valid-time* or *historical databases*.

TOD and the Arden Syntax do not provide direct support for intervals, and both have limited temporal querying abilities. The Arden Syntax does, however, support a type of query that allows the combination of multiple temporally concurrent data elements into a single list. This process provides a simple way of doing a temporal join. The semantics of this join are not well specified, however, and the process cannot be extended to interval-based data. Also, because instant timestamps encode only explicit time values, they cannot directly represent ongoing clinical situations.

The original Chronus system supported an interval-based temporal representation[5]. Chronus was based on the relational model and performed temporal queries with a query language that was an extension of SQL. Its temporal model and query language had a number of shortcomings, however. Chronus assumed all tables were temporal, so a Chronus database schema was incompatible with a standard relational schema. Hence, it could not work with legacy non-temporal data. Furthermore, its query language was a small subset of SQL with limited expressive power. Many temporal queries had to be expressed as a sequence of two or more subqueries queries requiring temporary tables to hold intermediate results.

Temporal Databases

Modern clinical database systems typically use relational databases, which provide well-defined data models and query languages. However, the relational

model has two significant shortcomings concerning temporal data: 1) it has no temporal model, and 2) its query language has very limited temporal operators:

(1) The relational model provides poor support for storing complex temporal information. A simple instant timestamp is all that it provides, and there is no consistent mechanism for associating the timestamp with non-temporal data. For example, if a database row contains some temporal information, there is no indication as to the relationship between it and the non-temporal data in the row. Does the timestamp refer to the point at which the information was recorded, or the point at which it was known? Other shortcomings include no standard way to indicate a timestamp's *granularity*. A timestamp's granularity defines the resolution at which it is recorded. For example, some timestamps are recorded with a resolution of days; others are specified with a resolution of minutes. Another shortcoming is that the relational model does not support automatic *coalescing* or merging of temporally overlapping data. In a temporal database, if two or more non-temporal data elements are identical over overlapping or adjacent intervals, combining — or coalescing — these intervals into a single longer interval may be appropriate. This process can be complex and time-consuming. A further shortcoming is the lack of a standard means of referring to the current time or 'now'. References to the current time are pervasive in temporal queries[6].
(2) The SQL query language also provides very limited support for expressing temporal queries. These queries typically require formulating multiple SQL queries, with the creation of one or more intermediate temporary tables.

Applications that work with complex temporal data must thus define their own temporal models and query systems. The query systems often require custom code and reside outside the database. Because of the relational model's limited temporal support, thousands of applications per year must reinvent the wheel and develop their own temporal query systems.

The fundamental problem is that the relational model is two-dimensional — rows and columns provide the indexes to two-dimensional tables. Time adds a third dimension to the model, and simply adding extra time columns to a two-dimensional temporal table is not sufficient to turn it into a three-dimensional temporal table. The standard relational model treats all columns equally and would not capture the semantics of the extra temporal columns. Incorporating time into the relational databases requires extensions to the relational model and the SQL query language.

Several proposed extensions to the relational model address these shortcomings. Most research has focused on *valid-time* databases, in which temporal information is attached to all rows in a temporal table[4]. This structure extends two-dimensional relational tables to incorporate a third dimension. In these tables, every tuple holds temporal information denoting the information's valid-time.

Two types of temporal tables are *event tables*, which hold instant timestamps, and *state tables*, which hold interval timestamps. For example, laboratory values are typically stored in event tables, while information about drug regimens can be held in state tables, because drug treatments generally extend over time.

The most comprehensive outline of a temporal query system is the TSQL2 query language specification[7]. The TSQL2 specification integrates much of the current thinking in relational temporal database research and serves as a useful synthesis of research in the area. Some of its features have influenced the design of time related components in the SQL3 standard[8]. Unfortunately, no implementations of the TSQL2 specification exist. Because of its size and complexity, a complete implementation would require a considerable amount of work. Not all features of the TSQL2 specification are necessary to build a usable temporal query system, however. We have selected a set of core features that we have found essential when building temporal query systems for clinical decision support systems and have incorporated these features into Chronus II.

Chronus II Database Mediator

Chronus II is a temporal database mediator that extends the standard relational model and the SQL query language to support valid-time temporal queries. Its design was influenced by the original Chronus query system and by TSQL2. Chronus II adopts a valid-time temporal model and supports state, event, and non-temporal tables. It provides an expressive temporal query language. It is a general-purpose temporal query language but is tuned to the querying requirements of clinical databases.

Chronus II is implemented in Java and is designed to operate as a layer above existing relational databases. Chronus II takes a temporal command, generates standard SQL statements for the non-temporal part of the command, and completes the processing of the temporal part in memory. Results are passed to the user or written to a database. It interacts with the database through a JDBC interface and is not tied to any particular database implementation.

Chronus II was designed to incorporate a number of core features that are required to provide useful

temporal query support for clinical systems. The primary features that we selected are as follows:

Valid Time Temporal Model We selected this model because we have found that interval support is crucial in our systems. The basic features required by this model are supported. They include timestamp representation, granularity representation and conversion, and coalescing. The temporal query language provides the standard interval-based Allen operators that allow interval-based temporal queries[9].

Expressivity The original Chronus query language had shortcomings that made concise query expression difficult. For example, many typical clinical temporal queries had to be written as two or more subqueries. The temporal query language proposed in the TSQL2 specification also has shortcomings. For example, the proposed syntax is unwieldy, and complex temporal queries are typically not very concise or readable. Chronus II provides a temporal query language that is more concise yet equally expressive.

Portability Chronus was designed to operate above a relational DBMS using ODBC as its access layer. This approach dramatically increases portability, allowing it to be used with any ODBC-compliant database. An additional facet of portability is query language conformance with existing SQL standards. Ideally, Chronus II should support any SQL query that conforms to the SQL-92 standard. However, given that no complete implementations of this standard exist, support for a significant subset of SQL is sufficient.

The Chronus II temporal query language is a strict superset of the SQL89 grammar, and our temporal schema is a strict superset of the standard relational schema. Thus, our system accepts all standard SQL queries and works without modification on standard relational databases. Additionally, Chronus II supports existing non-temporal database schema. Thus, in addition to being compatible with the relational model at the query level, it is also compatible at the schema level. Hence, if necessary, Chronus II can be used as a query processor interface to existing non-temporal DBMSs.

Maintaining compatibility with future database standards is desirable. Although there are no standards for temporal databases *per se*, parts of the TSQL2 specification have influenced a number of temporal extensions in the proposed SQL3 standard[8]. The temporal model and query language we have adopted maintain a relatively close mapping to this standard.

Efficiency DBMS vendors have gone to significant effort to optimize SQL queries. Chronus II was written to exploit these optimizations by offloading as much work as possible to the DBMS. All the non-temporal parts of a temporal query are processed by the standard SQL processor. Ideally, the entire functionality of Chronus II would be implemented at the DBMS level. This approach would dramatically improve performance. Altering an existing DBMS to support this functionality would be difficult, however, as source code is typically inaccessible. The solution would also be non-portable. Of course, writing a DBMS from scratch to support temporal queries is also a possibility, but this approach would be even less desirable.

Temporal Joins Performing relational joins on temporal tables is non-trivial, and any system that includes temporal information must rigidly define join semantics. For example, two temporal tables cannot be joined using the semantics of the standard relational join. The resulting table would have two valid-time columns and would not fit the temporal model. Thus, successful multiple-table queries require a temporal join that can combine time-stamped data from two or more temporal tables and produce a new table that maintains the temporal semantics in the original tables.

Although, in principle, standard SQL queries can produce such a join, the queries are very long, complex and difficult to write. Chronus II incorporates an approach for performing temporal joins[10]. Without temporal join support, queries that involve temporal and non-temporal tables would not be possible.

Indeterminacy Not all temporal information is known with certainty[11]. For example, a physician may only know that an event happened "between 1 a.m. and 6 a.m." or "sometime last year." These are examples of *temporal indeterminacy* – that some event occurred is known but the exact time of its occurrence is not. Indeterminacy can arise for several reasons: many dating techniques are inherently imprecise and the time of many events is often imprecisely specified.

In an abstract sense, all temporal information has some degree of uncertainty associated with it. The degree of accuracy of any measurement is *at the very least* limited by the resolution of the measuring device. This resolution must be specified at some granularity, and thus, by definition, the measurements at all finer granularities are imprecise.

Temporal Abstraction A crucial part of temporal reasoning in our clinical systems is creating high-level temporally extended concepts from raw time-stamped data. This task is often called *temporal abstraction*. For example, a sequence of daily blood pressure values may be abstracted into multi-week intervals of high or low blood pressure. We developed a system called RASTA to perform this type of temporal reasoning[12]. RASTA generates interval-based time-related abstractions from time-oriented data and operates on data values that are time-stamped with points or intervals. We have integrated RASTA with Chronus II to provide temporal abstraction at the query level.

Patient	Problem	ValidTime
J. Smith	diabetes	{[14/Feb/2002-'+']}
J. Smith	CHF	{[10/Mar/2002-'+']}
P. Jones	hypertension	{[1/Apr/2002-'+']}
R. Franks	hypertension	{[13/Feb/2002-'+']}

Table 1: PROBLEMLIST. A temporal table containing diagnoses for a list of patients. The ValidTime column shows the time periods during which the associated tuple is valid. The '+' value denotes that the tuple is still valid.

Patient	Drug	Dosage	ValidTime
J. Smith	atenolol	10 mg	{[20/Mar/2002-10/Apr/2002]}
J. Smith	atenolol	15 mg	{[11/Apr/2002-12/May/2002]}
P. Jones	atenolol	10 mg	{[1/Apr/2002-6/May/2002]}
R. Franks	pindolol	25 mg	{[4/Feb/2002-14/May/2002]}

Table 2: DRUGS. A temporal table containing drug regimen information for the patients in the PROBLEMLIST table.

Sample Chronus II Query

The following example illustrates some of the features of the Chronus II query language. Suppose a developer is given two Chronus II temporal tables containing problem list (Table 1) and drug regimen (Table 2) information for a set of patients and asked to write the following query:

"Show problem and drug combinations for patients whose problem was diagnosed in the last year. Restrict the results to combinations involving drug regimens lasting longer than two weeks and excluding combinations where a drug regimen was initiated before the onset of a problem."

This query can be written as a temporal join in Chronus II as follows; the result is presented in Table 3:

```
TEMPORAL SELECT P.Patient, P.Problem, D.Drug
FROM PROBLEMLIST AS P, DRUGS AS D(Patient, Drug)
WHERE P.Patient = D.Patient
WHEN DURATION(D, 'weeks') > 2 AND
        DURATION(BEGINNING(P), 'now', 'years') < 1 AND
        BEFORE(BEGINNING(D), BEGINNING(P))
```

This query resembles standard SQL with some extra clauses. Apart from the initial TEMPORAL keyword, the most obvious addition is the WHEN clause. This clause is analogous to the WHERE clause and contains temporal predicates. In this case, the predicates are two forms of the temporal operator DURATION and the BEFORE operator. The first form of the DURATION operator takes a table and a granularity as parameters and calculates interval lengths in the table at the specified granularity. The second form takes two timestamps in addition to a granularity specification. The first timestamp is extracted from the start of the valid-time of the PROBLEMLIST table using the BEGINNING function, and the second uses the special 'now' value, which represents the time that the query is executed.

Patient	Problem	Drug	ValidTime
J. Smith	CHF	atenolol	{[20/Mar/2002-12/May/2002]}
P. Jones	hypertension	atenolol	{[1/Apr/2002-6/May/2002]}

Table 3. Temporal join of the PROBLEMLIST and DRUGS tables, restricted to problems lasting longer than two weeks, and whose onset occurs before each drug regimen.

The FROM clause also contains a unique temporal extension called a *temporal projection*. The DRUGS table shows a history of drug and dosage information for a set of patients. In the query, however, the dosage is not relevant. Hence, the query forms a projection that excludes the dosage information. Thus, for example, the two adjacent intervals of the atenolol prescriptions at differing doses for J. Smith are projected into table P and coalesced into a single interval.

This six-line query demonstrates a temporal projection, a temporal join, coalescing, granularity conversion, and several temporal operators. An equivalent SQL query would be far less concise or readable, would probably need to be written as multiple queries, and would require temporary tables to hold intermediate results.

Experience with Chronus II

We have used Chronus II in a variety of ways in clinical systems that we have developed:

Decision Support We are using Chronus II in a hypertension advisory system called ATHENA. ATHENA has been deployed at three clinical sites in the Veterans Administration Health Care System[13]. Chronus II is used in three principal ways in ATHENA: (1) it serves as the underpinning of a conversion tool that generates a temporal database from the VA's MUMPS hierarchical database; (2) it is used by ATHENA's decision support component to perform temporal queries on this database; and (3) it is used by ATHENA's user interface component when selecting and displaying current and historical patient data.

Temporal Predicate Language We are also using Chronus II as a temporal predicate language in a guideline modeling system[14]. The language is used to define abstract concepts in the model. The concepts, in turn, are used to model decision-making and guideline eligibility criteria, which generate recommendations for clinical decisions and actions.

Temporal Constraint Language Another project using Chronus II was the QUIL project for guideline-based quality assessment[15]. QUIL is a language representing quality indicators that are derived from low-level plan elements and higher-level intentions in medical guidelines or performance indicators. QUIL queries consist of *goal* and *enabling* temporal constraints.

Satisfying a goal constraint defines a clinical execution of the medical guideline that satisfies the quality indicator given the appropriate context defined by the enabling constraint. The target language for the goal and enabling temporal constraints is Chronus II. The system compiles QUIL temporal constraints into Chronus II temporal queries.

Discussion

Clinical decision-making can depend on detecting temporal trends and temporal patterns. Formulating temporal queries to detect these patterns is difficult in standard relational databases. Extensions are required to the standard relational model and to the SQL query language to enable concise expression of these queries. Interval-based data are also required to formulate sufficiently complex temporal queries. Although few current clinical databases support interval-based data, these intervals can often be constructed from existing instant timestamps. This process usually requires detailed knowledge of the underlying data, and may entail a customized solution for each clinical domain. However, it does provide a path to enable the exploitation of the full power of valid-time temporal databases.

An added impetus for interval support in clinical databases is that the temporal database community has accumulated considerable experience with valid-time databases. This experience can be applied to the clinical domain. Future SQL standards are also likely to use ideas from this body of research. For example, some components from the TSQL2 temporal query specification have already been incorporated into the SQL3 standard[8].

This paper has outlined a system called Chronus II, which incorporates some of the ideas proposed in the TSQL2 specification and other temporal database systems and extends them to the temporal querying needs of clinical decision support systems. We have used Chronus II extensively in several clinical systems that we have developed.

Acknowledgements

This work has been supported by grant LM05708 from the National Library of Medicine, and by a grant from FastTrack Systems, Inc.

We thank Valerie Natale for her valuable editorial comments. We also thank Amar Das for his work on the original Chronus system.

References

1. Clifford J, Dyreson CE, Isakowitz T, Jensen CS Combi C, Shahar Y. *Temporal reasoning and temporal data maintenance in medicine: issues and challenges*. Computers in Biology and Medicine, 1997; **27**(5): 353-368.
2. Wiederhold G. *Databases for healthcare*. Lecture Notes in Medical Informatics, Heidelberg, Germany, Springer-Verlag, 1981.
3. Hripcsak G, Ludemann P, Allan Pryor T, Wigertz, OB, Clayton P. *Rationale for the arden syntax*. Computers and Biomedical Research, 1994; **27**: 291-324.
4. McKenzie LE, Snodgrass RT. *Evaluation of relational algebra incorporating the time dimension in databases*. ACM Computing Surveys, 1991; **23**:501-543.
5. Das AK, Musen MA. *A temporal query system for protocol-directed decision-support*. Methods of Information in Medicine, 1994; **33**:358-370.
6. Snodgrass RT. *On the semantics of 'now' in databases*. ACM Transactions on Database Systems, 1997; **22**(2): 171-214.
7. Snodgrass RT (Ed). *The TSQL2 temporal query language*. Kluwer Academic Publishers, Boston, 1995.
8. Snodgrass RT, Jensen CS, Steiner A. *Transitioning temporal support in TSQL2 to SQL3*. Temporal Databases: Research and Practice, 1998; 150-194.
9. Allen JF. *Maintaining Knowledge about Temporal Intervals*. Communications of the ACM, 1993, **26**(11): 832-843.
10. O'Connor MJ, Tu SW, Musen MA. *Applying temporal joins to clinical databases*. Proceedings of the AMIA Annual Symposium, Washington, D.C., 1999: 335-339.
11. O'Connor MJ, Tu SW, and Musen MA. *Representation of temporal indeterminacy in clinical databases*. Proceedings of the AMIA Annual Symposium, Los Angeles, CA, 2000: 615-619.
12. O'Connor MJ, Grosso WE, Tu SW, Musen MA. *RASTA: A distributed temporal abstraction system to facilitate knowledge-driven monitoring of clinical databases*. MedInfo 2001, London, U.K., 2001; 508-512.
13. Advani A, Tu SW, O'Connor MJ, Coleman R, Goldstein MK, Musen MA. *Integrating a modern knowledge-based system architecture with a legacy VA database: The ATHENA and EON projects at Stanford*. Proceedings of the AMIA, Annual Symposium, Washington, D.C., 1999.
14. Tu SW, Musen MA. *Modeling data and knowledge in the EON guideline architecture*. MedInfo 2001, London, UK, 2001: 280-284.
15. Advani A, Musen MA, Shahar Y. *Medical quality assessment by scoring adherence to guideline intentions*. Proceedings of the AMIA Annual Symposium, Washington, DC, 2001.