Proc. Internat. Database Engineering and Applications Symposium (IDEAS 2000), September 18-20, 2000, Yokohama, Japan, IEEE Computer Society Press, 2000.

Design of RapidBase—an Active Measurement Database System

Antoni Wolski Jorma Kuha Tapio Luukkanen Antti Pesonen

Technical Research Centre of Finland (VTT)
VTT Information Technology
P.O. Box 1201, FIN-02044 VTT, Finland
e-mail: first-name.lastname@vtt.fi

Abstract

In data-intensive industrial on-line applications utilizing live process data, one faces an unusual set of database requirements. The process measurement data need to be acquired at great speed, organized in time series and made available for time-based retrieval. Active capabilities and functional extensibility are needed to implement a flexible data-driven processing paradigm. An efficient transaction logging and recovery mechanism is needed in order not to impede the data acquisition flow. RapidBase is a system that meets these requirements. It utilizes a main-memory database, a unique temporal-relational data model for handling time series, and an elaborate trigger subsystem. It is implemented as a server program equipped with interfaces of high power of expression.

1 Introduction

Although the requirement for data management is omnipresent in various advanced applications, the main-stream notion of a database may be not a best choice in all cases. Certain application classes require a special collection of features not found in general-purpose database systems. This paper presents the design of RapidBase — a database system tuned to the needs of handling measurement data in modern industrial applications.

In complex industrial installations, such as a paper mill, a power station, a power grid or a telecomm network, huge amounts of measurement data are generated and need to be processed for the purposes of control room processing and process monitoring. Here, we do not deal with the needs of the process automation systems because they are, typically, satisfied by dedicated equipment (like PLCs—programmable logic controllers). On the other hand, the needs for the process data management are obvious and the requirements are becoming more and more demanding. For example, in a typical power generating unit of a power station, the data acquisition rate requirement has grown, in few years, from a couple of thousands of measurements per second to almost ten thousand measurements per second. The data needs not only be collected promptly, but also summarized and visualized for the operators, within a tolerable time delay of 1—2 s. Such applications—the so-called process management systems—require special process database systems, such as the one described here.

We have studied the needs for process databases, and the technologies thereof, since 1992, in various projects sponsored by industry-led consortia. In retrospect, the flow of ideas and requirements from industrial partners has been indispensable in achieving the proper set of system functionality. We presented a case study and a first prototype in [WKP96]. Since then, we have introduced a new data model and a more general data language, extended the active capabilities and added new recovery and extendibility features, all of which are discussed in the sequel.

In the research community, the notion of ARCS (Active Rapidly Changing Data Systems) was introduced [Dat94], but no significant implementations were reported at that time. The support for time series (which is a central concept in a process database) has received more attention [Dre94], and several implementations are known but they are mostly tuned to the needs of financial applications. The same is true for time series extensions of commercial database systems, like Informix Dynamic Server (the DataBlade technology acquired from Illustra), Oracle8i and IBM DB2 Universal Database. Very few generalized systems for handling industrial temporal data are known. Some of them are reviewed in Section 8. In research, the focus has been on concurrency control and real-time scheduling, as in [Shi+93, Gra93], rather than on ease of performing temporal operations on data.

In our time series implementation we used some of the results produced in the research on temporal databases, including the time concept taxonomy and some of the syntactical constructs of TSQL2 [Sno95].

The research on active databases [WC96, Pat98] also inspired our work a lot. We introduced new types of database triggers to satisfy special needs of process management.

The next section deals with architectural issues. In Section 3, the RQL (RapidBase Query Language) is presented, together with the related data model. In Section 4, the active capabilities are explained, in Section 5 the extensibility features are revealed, and the recovery mechanisms are summarized in Section 6. Section 7 contains implementation notes and performance figures. A comparative review of some commercial implementations intended for the same range of applications is given in Section 8.

2 Architecture

RapidBase is built according to a classic Client/Server architecture where the Server runs in a single operating system process and applications connect to it over TCP/IP connections. In addition to being a typical Client application, a process may register itself with the Server as an Action Executor application, for performing trigger-invoked and user-written actions. In order to increase the robustness of the system, a deliberate choice was made for not allowing the applications to share the same address space with the Server. Consequently, the Server is protected by the network interface layer scanning all the messages and detecting errors therein. When compared to shared-memory-based solutions, the architecture introduces a communications-induced overhead which, however, does not turn out to be significant. It was easy for us to meet (and exceed) the original performance requirements of 500 measurement update transactions per second on a regular PC (Windows NT) platform and at least 1000 on a low-cost UNIX platform.

The main application interface is a C++-based class library RAPI (RapidBase API) that is an object-oriented incarnation of the standard SQL CLI (Call Level Interface) [CLI-95]. ODBC and JDBC drivers are also available for Windows and Java application development, respectively. Also, Action Executor programming interfaces are available for C++ and Java. There is also a C++ interface for developing RapidBase User-Defined Functions (UDFs). UDFs are compiled into plug-in modules (dynamic link libraries) that are run-time loadable into the Server. A number of administrator's tools have been implemented as Java applets.

A distinguishing feature of RapidBase is that its database is totally main-memory-based. This allows for single-digit millisecond response times. The database is backed up on disk using an optimized transaction logging method described in Section 6.

3 RQL Data Model

Originally, the idea of using SQL as a basis for our design, came from our industrial partners. They were concerned about the issues of a learning curve and supply of trained programmers. They felt an SQL-based system would be easier to introduce to the engineers than, for example, an object-oriented database. The decision can be appreciated now when one sees how easy it is to fill a spreadsheet with the measurement data, and how an ad-hoc ODBC-based VisualBasic application is just a few mouse clicks away.

A shortcoming of SQL from our point of view was the lack of support for time series. So, we designed RQL—a language based on SQL-92 [SQL-92], with extensions for time series support. We proposed the first model in [WKP96]. A new, improved model is presented in this paper. It has the advantage that it folds into a relational model in the absence of temporal data.

3.1 HISTORY column type

The time series support is encapsulated in a special RQL column type called HISTORY. A value of type HISTORY is a nested table of its own, as can be seen in Fig. 1 where a base table with two history columns is shown. The gray table seen in the foreground of Fig. 1 represents an SQL view of the RQL table—it is a snapshot table comprising the currently valid column values. This SQL view is called a *current view* of the RQL table. In the same time, previous values of history columns are stored as time series and are accessible through extended syntax. Thus, for a temporal RapidBase table, there always exists the SQL view and the RQL view, and they may be used interchangeably. The system maintains identity of rows, and the temporal information, by introducing implicit columns shown in light gray, in Fig. 1. The OID (object identifier) columns are used for automatically assigned row and subrow identifiers. Each (history column) subrow (called a history record) carries also two timestamps (OTS and OTS_END) used for denotation of the subrow's valid time. The user need not to define the implicit columns, or supply values for them.

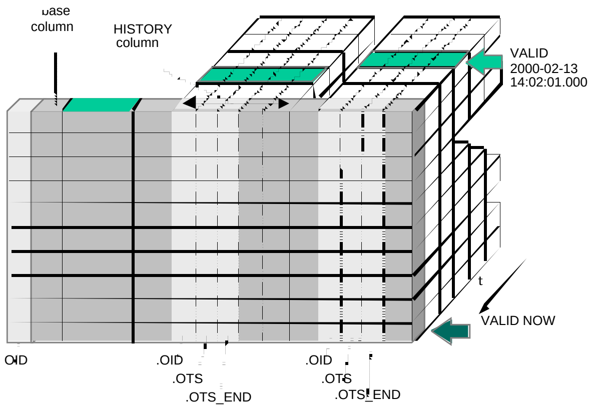

Fig. 1. An RQL table with two history columns.

Assume the table in Fig. 1 represents a set of temperature probes. The table can then be created with the following definition statement:

```
CREATE TABLE tempr_probes (
  probe_id CHAR(8),
  name     VARCHAR(80),
  type     CHAR(10),
  measur_h HISTORY (
     tempr    FLOAT,
     quality  SMALLINT
     ) SIZE 10000,
  scale    INT,
  state_h  HISTORY (
     state    CHAR(8)
     ) SIZE   1000
);
```

The logical structure of the table is different from a generalized temporal data model like TSQL2 [Sno95] in that a user may choose which columns are of a snapshot nature (base columns) and which ones have temporal characteristics (history columns). Another difference is the introduction of subcolumns. The justification for this is that, very often, measurement data is acquired as value vectors that needs to be treated as identifiable units in subsequent processing. Instances of history columns may have different sizes (in terms of number or history records), but the maximum size (user-defined or default) cannot be exceeded.

3.2 Temporal semantics

Although a history column resembles an ordinary table, it maintains special semantics which can be summarized in the following way:

- The size (cardinality) of a history is fixed (note the SIZE clause in the example; if it is omitted, a default size is used). When the history becomes full, the size is maintained by replacing the oldest history record with the latest one (the circular buffer principle). If the old history data are not to be lost, active objects called *archivers* may be defined and enabled in the database, with the purpose of moving the outdated data to some other medium. The absolute size limit may seem a strong measure but it allows to control the database size buildup accurately.
- The history records are ordered chronologically, based on their valid time. The latest record belongs to the current view of the database.
- All the subcolumns are accessed using the dot notation, e.g. " measur_h.ots, measur_h.tempr".
- History columns contain implicit timestamp subcolumns that can be accessed in queries:
  .OTS: the beginning of the record's valid time (user or system supplied); .OTS_END: the end of the record's valid time (derived).

RapidBase database is a *valid-time temporal database* [Jen92]. Consequently, all temporal joins are also valid-time joins. There is a notion of valid time associated with each history record. A valid time of a history record is an (absolute) time period for which the values in the history record reflect the modeled reality. The user usually supplies the value of the valid-time start point, in the form of the measurement timestamp (the .OTS subcolumn). The system automatically maintains the end point value which is a time point just before (with the system time resolution of 1 ms) the value of the timestamp of the chronologically following record. The valid time of the latest history record extends up to the current time, depicted as NOW in the RQL parlance. The valid-time end point of a record is always accessible via the implicit subcolumn .OTS_END. The values of this column are not stored—they are derived from the database at run-time. If an .OTS value is not supplied with the measurements, the system automatically assigns .OTS the transaction time. Effectively, this leads to automatic translation from *transaction time* [Jen92] to valid time.

The most convenient way to specify temporal search conditions is to use the VALID predicate. With the VALID predicate, one can specify time points, intervals and periods, to restrict the temporal query.

The RapidBase temporal table model folds into the standard SQL table model, when standard SQL statements are used. This results from the RQL convention that the VALID predicate is always present (explicitly or implicitly) in a SELECT statement. If it is not explicitly specified, the default form is "VALID NOW". With, VALID NOW, the table of Fig. 1 is reduced to the current view table shown shaded in gray. For example, the standard SQL query

```
SELECT probe_id, name
   FROM tempr_probes
   WHERE state_h.state = 'ON';
```

is restricted to the current view of the table because of the default VALID predicate.

An absolute valid-time point may be specified in RQL, as in

```
SELECT probe_id, name, state_h.state,
measur_h.tempr
   FROM tempr_probes
   WHERE VALID '1998-02-13 14:02:01.000';
```

In this case, the valid-time join is performed between the two history columns (`state_h` and `measur_h`). In multi-table joins, the valid-time join is conceptually performed before applying other join conditions.

There are various ways to specify a valid-time interval of your interest. In the following query, all the measurements of the last minute are retrieved if their probe state was 'ON':

```
SELECT ots, ots_end, probe_id, name,
measur_h.tempr
```

```
        FROM tempr_probes
        WHERE state_h.state = 'ON'
        AND VALID FROM NOW – INTERVAL '1' MINUTE;
```

It is also possible to "sample" the query result with a given time resolution. If we are interested in retrieving the measurement data in 10 s time steps for the latest hour, we would say:

```
    SELECT ots, probe_id, state_h.state,
    measur_h.tempr
        FROM tempr_probes
        TIMEPOINT SERIES INTERVAL '10' SECOND
        WHERE VALID FROM NOW – INTERVAL '1' HOUR;
```

3.3 Result table format

All columns potentially available in a query accessing the example table "tempr_probes" are shown in Fig. 2.

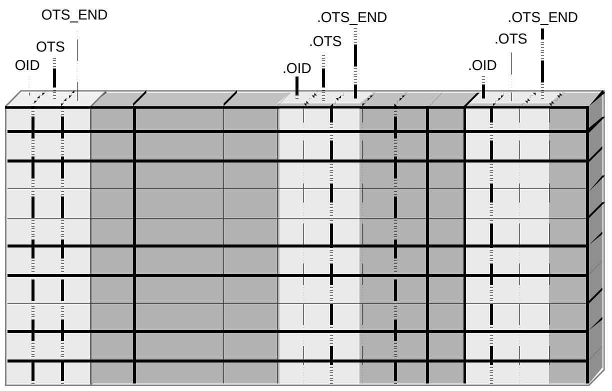

Fig. 2. Implicit (light gray) and explicit (dark gray) columns available in the result table.

The column names OTS and OTS_END found in the two last query examples are virtual columns of the result set. They indicate the synthetic valid time of the result row. The values of the columns are calculated from the OTS values of the histories composing the temporal join. Fig. 2 depicts all the implicit and explicit columns of the example table, potentially available in the query result set.

It was a deliberate choice to represent query results as "flat" tables. Thanks to that, standard tools for result set processing, such as ODBC and JDBC may be used. The result set can be sorted with the standard ORDER BY clause.

3.4 Data acquisition in RQL

Adding new data to measurement histories is done by updating the current view rows. A standard SQL UPDATE statement suffices, as in:

```
    UPDATE tempr_probes
        SET measur_h.tempr = 134
        WHERE probe_id = 'TEMP34'
```

The result of executing the statement is the introduction of a new (current) history record in the history. The previous current record becomes (chronologically) the one preceding the current one. It may be accessed by using a VALID predicate with a correct time point value, but it is not accessible any more with the standard SQL SELECT statement.

In RQL, new statements were introduced for maintaining existing histories. For example, the history records can be changed with the UPDATE HISTORY statement and they are removed with the DELETE FROM HISTORY statement. Also, INSERT INTO HISTORY was considered (for inserting history records "into the past"), but has not been implemented yet.

4 RapidBase Triggers

The RapidBase trigger subsystem covers a broad range of active database capabilities. RapidBase triggers are based on the classic ECA (event-condition-action) model [MCD89]. In their simplest form, they follow the trigger definition syntax of the SQL-99 standard [SQL-99]. In RapidBase, various extensions have been made to the basic ECA trigger model. As most of them are described elsewhere [WKP96, WB98] , only a summary is given below.

Multiple condition-action blocks

This simple syntax extension allows for more expressive triggers. For a single event type, it is possible to define a control structure to invoke alternative actions (or action sets) depending on the evaluation of the conditions. With the following trigger, the two abnormal probe states are set (and reset to NORMAL) when the measurement values exceed given thresholds:

```
CREATE TRIGGER Setting_states UPDATE OF
measur_h ON tempr_probes
    WHEN OLD.measur_h.tempr <= 1000 AND
NEW.measur_h.tempr > 1000
        DO SET state_h.state = "OVERHEAT"
    WHEN OLD.measur_h.tempr >= 900 AND
        NEW.measur_h.tempr BETWEEN 0 AND 900
        DO SET state_h.state = "NORMAL"
    WHEN OLD.measur_h.tempr >= 0 AND
NEW.measur_h.tempr < 0
        DO SET state_h.state = "FREEZE"
    WHEN OLD.measur_h.tempr < 10 AND
        NEW.measur_h.tempr BETWEEN 10 AND
1000
        DO SET state_h.state = "NORMAL";
```

The SET statement is a simplified version of UPDATE (it is just the SET clause). It is used for changing values in the triggering row.

Built-in composite events

Instead of supporting a generalized composite event language, a few pre-defined composite event types are available. They are modeled after well-known notions, like timers and counters. Timer and counter triggers were presented in [WKP96]. An alarm state column is assigned a new value, resulting from the following timer trigger, when a probe value has been over a given threshold for more than 10 minutes:

```
CREATE TRIGGER
Delayed_overheat
   TIMER ON tempr_probes
   SET INTERVAL '10' MINUTE
   START ON UPDATE (
measur_h.tempr > 1000 )
   CLEAR ON UPDATE (
measur_h.tempr < 900 )
   DO SET state_h.state =
"OVERHEAT";
```

The CLEAR clause nullifies the timer when a separate condition is satisfied.

Fuzzy triggers

The power of fuzzy inference is utilized in fuzzy triggers [WB98, Bou+97]. In *C-fuzzy* (condition-fuzzy) triggers, fuzzy rule sets may be used in the condition evaluation. In *CA-fuzzy* (condition-action-fuzzy) triggers, the action part contains a fuzzy inference block which is evaluated to select a concrete action. Implementation of *fuzzy events* [BW97] was considered but has not been carried out.

Trigger variables

Trigger-scoped variables allow passing results of arbitrary queries to other components of a trigger. Set-oriented variables (collections) may be also fuzzy-quantified using quantifiers like "few", "most", etc. Values for collections are supplied with SELECT statements as in:

```
...
 VAR COLLECTION temperatures =
  (SELECT measur_h.tempr FROM tempr_probes)
...
```

Internal actions

Internal actions comprise of RQL statements and user-defined procedures, and they are run in the same isolation unit (transaction) as the triggering statement. They are executed in the deferred mode (at the end of the transaction). User-defined procedures (UDFs) are called with the EXEC statement, like the following one calculating a one-minute average of a history subcolumn and storing the result into another column:

```
...
DO EXEC average_1min('measur_h.tempr',
'aggr_h.value')
...
```

External actions

External actions are asynchronous invocations of procedures in external programs. They run in a detached mode, i.e. outside of the triggering transactions. The intended use of external actions is mostly data and event dissemination to autonomous applications. External actions represent a true "push" paradigm, whereby the Server invokes (in a general case) a multicast transmission to Clients. The CALL statement is used to invoke external actions, like in

```
...
DO CALL ProbeControl.Over_heat( h.temp )
...
```

whereby the *Over_heat* method is invoked at the Action Executor process called *ProbeControl.*

5 User Defined Functions

The RapidBase Server can be functionally extended at runtime with User Defined Functions (UDFs) which are user-written C++ methods contained within dynamic link libraries. UDFs can be used in two ways: as functions (which return a value) in RQL expressions, and as procedures (which do not return a value) in internal trigger actions. Both types can also be directly invoked by Clients. Functions and procedures can be written to accept any (even variable) number of parameters of any RQL type. In the following sample of an RQL script, a UDF named "smallest" is defined in the database, used in an expression and, finally, dropped:

```
CREATE OR REPLACE FUNCTION smallest
   EXTERNAL NAME 'Smallest'
   IN 'demodll.dll';

UPDATE foobar
   SET field1 = smallest(field2, field3,
      3.14159)
   WHERE foo_id = 101;

DROP FUNCTION smallest;
```

The RQL syntax for accessing the external functions (e.g. registering them to the Server, executing and dropping them) follows the example of existing commercial products and the SQL-99 standard. However, while the commercial implementations typically define a C language interface between the Server and the DLL, RapidBase implements a class-based C++ interface. A common header file defines the *CallContext* class which contains methods the UDF implementation uses for retrieving its parameters and their types, for passing results back to the Server, and for accessing other Server functionality.

The implementation allows for rebinding (reloading) the functions "on-the-fly" even if they are specified in the action part of a trigger (or some other pre-compiled statement).

Multiple operating systems (Windows NT, Linux, HP/UX) are supported through encapsulation of OS-dependent parts. The multi-platform support for UDF's was inspired by [Roe99].

6 Recovery with adjustable durability

The automatic recovery mechanism uses checkpoint files and a redo-log [GR92]. No undo log is needed because a shadow-based update method is used (a working copy of a row is created for the time of processing the transaction). The policy of redo logging is tunable to durability requirements. The usual WAL (write-ahead-log) policy is available by way of the standard SQL statement COMMIT WORK. This guarantees full durability but it compromises the response time. For the sake of rapidly changing data, a more efficient asynchronous policy (inspired by [JSS93]) was introduced, too. The statement COMMIT LOGGER results in the asynchronous flushing of the memory-resident transaction log to disk. Even more relaxed logging is available. If no COMMIT statements are used, the log is flushed to disk periodically, following a default (e.g. 5 s) or a specified time interval. Both the logging and automatic checkpointing may be also totally disabled.

The relaxed logging policies are meant to be used with measurement data streams of high density. This may result in loosing of a value or two, in a crash, but the loss need not be significant from the temporal consistency point of view.

7 Implementation, performance and utilization

The central part of the software, the RapidBase Server is written in C++ in a platform-independent way. Currently, versions for Windows NT, HP-UX and Linux are available. A subsystem initialization framework, that is activated upon startup, has made it possible to decompose the system into compile-time configurable modules (subsystems). For example, the fuzzy trigger subsystem may be included or not, in the final executable build. The solution includes also the configuration of RQL syntax, so that the legal RQL syntax always corresponds to the functionality of a given configuration. In addition to compile-time configurability, the Server's functionality may be controlled by way of startup parameters, dynamic control commands and user-defined functions and procedures.

Because no composite benchmark exists for the type of applications RapidBase is intended for, we present only some selective performance figures here, essentially related to executing UPDATE statements on histories.

On a reference platform of 200 MHz Pentium II PC running Windows NT 4.0, the RapidBase Server is capable of accepting up to 1000 UPDATE transactions per second. Simple SELECT statements are executed within single millisecond response time range. The above figures were obtained in the operational mode whereby each RQL statement was fully run-time interpreted by the Server. Performance improvements are expected when the so-called PREPARE processing (i.e. statement precompilation) will be implemented in the near future. Additionally, the practical data acquisition performance may be dramatically improved by using user-defined procedures (UDFs) for that purpose. In a recent test, a throughput of 6 000 pre-compiled UPDATE statements per second was attained from within a UDF.

An execution time of a typical trigger with internal actions is within 0.1 ms. For example, this results in the update rate of 900 updates per second if each update fires a trigger. The achieved performance of triggers stems from the fact that triggers are pre-compiled, including the condition and action parts. The same principle applies to fuzzy triggers. The execution time of a fuzzy trigger with at most 32 fuzzy rules is within 0.5 ms and scales linearly with the number of rules (for more on fuzzy trigger performance, see [Bou+97]). The stress on trigger performance reflects the idea that the primary responsibility of RapidBase is intended to be active monitoring of industrial processes.

Performance of RapidBase was demonstrated in the implementation of the Rubic Real-time OLAP Engine [Kiv+99] where a load of hundreds of update transactions and thousands of trigger executions per second was sustained on the reference platform.

RapidBase is currently being used in several pilot implementations in Finland and, recently, also in France. The applications are, among others, a telecomm protocol analyzer, traffic monitoring in a telephone switch, hydro-electric power station simulator, and a web breakage sensitivity indicator in a paper machine.

The most notable commercialization effort is emerging from the cooperation with ABB Industry, a leading supplier of industrial high-power drive systems. A typical application involves a drive system of a paper machine. It comprises of up to one hundred (mostly AC) electric motors together with the related frequency inverter equipment, automatic control and a diagnostic system. The new generation of ABB's drive diagnostic system called Operator's Diagnostic Tool (ODT) will be based on RapidBase. The operational measurement data of all the motors in a system will be fed into the RapidBase Server, in intervals ranging from 100 ms to few seconds. Various state behavior models will be implemented with triggers, and the knowledge about the process causality will be captured in fuzzy rule sets. The main purpose of ODT is to advise the operator about appropriate actions when malfunctions are detected in the system. The tool is planned to be on the market in year 2001.

8 Comparative review of related systems

It is difficult to compare RapidBase with other systems because of its unusual set of functionality that is not matched by any other typical database system. However, there are some overlapping areas of functionality with other systems. We will pick up three of them: (1) a main-memory database, (2) the time series support and (3) active capabilities. Below, we will review some prominent commercialized implementations in the above areas.

8.1 Main-memory database

A few main-memory based implementations have emerged from research into the market. One is DataBlitz of Lucent Technologies (known before as Dalí of Bell Labs [Jag+94]). DataBlitz uses shared memory for interprocess communications, and compile-time interfaces (C++ and Ode) for database access. Both result in good performance. TimesTen by TimesTen Performance Software (before known as SmallBase of HP Labs [LN96]) is a true SQL system characterized by a run-time SQL interpretation. The PREPARE optimization is available and so are main-memory access method optimizations. ClustRa by ClustRa AS (before by Telenor of Norway) is an example of high-performance, high-availability system [Hva+95]. It utilizes a network of computers to distribute the load and to perform diskless transaction logging. All of the mentioned systems beat RapidBase in term of raw speed. They also offer a full transactional service while, in RapidBase, the unit of atomicity, isolation and durability is at most one RQL statement.

8.2 Time series support

As noted before, vendors of traditional database systems do not deliver the time series functionality required by industrial applications. In this area, the most visible product on the market is Industrial SQLServer by Wonderware. The supported time series model is separate from the SQL model and is based on named data series. Some other products include time series support for measurement data, for example Polyhedra (by Polyhedra) and RAPID Historian (by Automsoft International). Their data models are very limited when compared to RQL, and time series data are difficult to access from a generalized data language.

8.3 Active capabilities

Common ECA triggers can be found in many database systems nowadays, including the big brands and some of the ones mentioned in Sec. 8.1. However the capabilities are usually limited to what the SQL-99 standard specifies. RapidBase goes further in enriching the power of expression of trigger definitions and providing an original concept of fuzzy triggers. Also, triggers in RapidBase are highly optimized and there are good chances they can stand up against any competition in terms of performance.

9 Conclusions and future work

It turned out to be feasible to design and implement a non-traditional set of database functionality that can serve a broad range industrial applications that process measurement data. Although the major characteristics are in place, still many more have to be provided. In addition to improving the raw data manipulation performance, other directions, dictated by industrial application needs are also attractive. One development path is to introduce new, more powerful reasoning mechanisms behind the active capabilities, in addition to the fuzzy rule sets. In order to detect complex process states in real time, a set of pattern recognition methods could be considered too (including methods for recognition of temporal patterns). A need for adjunct data mining capabilities for extracting inference and pattern recognition models has also emerged. With the growing complexity of the active mechanisms, it is becoming obvious that the explainability features (why the system took the given decisions?) will be requested more and more. They will be essential in maintaining the confidence of users in the technology. All of these areas are currently being studied at VTT Information Technology.

The latest, up-to-date information on RapidBase is at: `http://rapidbase.vtt.fi`.

References

[Bou+97] T. Bouaziz, J. Karvonen, A. Pesonen, and A. Wolski, "Design and Implementation of TEMPO Fuzzy Triggers", *Proc. Eighth Int’l conference on Database and Expert Systems Applications (DEXA’97)*, Sept. 1-5, 1997, Toulouse, France, pp. 91–100 (http://www.vtt.fi/tte/projects/industrialdb/publs/tempo-design.pdf)

[BW97] T. Bouaziz and A. Wolski, "Applying Fuzzy Events to Approximate Reasoning in Active Databases", *Proc. Sixth IEEE Int'l Conference on Fuzzy Systems (FUZZ-IEEE’97)*, July 1-5, 1997, Barcelona, Catalonia, Spain, pp. 729–735 (http://www.vtt.fi/tte/projects/industrialdb/publs/f-event-triggers.pdf).

[CLI-95] ISO/IEC 9075-3. Information processing systems - Database language SQL, Part 3: Call-level interface. International standard, fourth edition, 1995. Ref. No. ISO 9075-3 : 1995 (E).

[Dat94] A. Datta, "Research Issues in Databases for ARCS: Active Rapidly Changing Data Systems", *SIGMOD Record*, 23(3), September 1994, pp. 8–13.

[Dre94] W. Dreyer et al., "Research Perspectives for Time Series Management Systems", *ACM*

SIGMOD Record, 23(1), March 1994, pp. 10-15.

[GR92] J. Gray and A. Reuter, "Transaction Processing Systems, Concepts and Techniques", Morgan Kaufmann Publishers, 1992.

[Gra93] M.H. Graham, "How to Get Serializability for Real-Time Transactions without having to pay for it", *Proc. Real-Time Systems Symposium*, Raleigh-Durham, North Carolina, December 1993, pp. 56-65.

[Hva+95] S.-O. Hvasshovd, Ø. Torbjørsen, S.E. Bratsberg and P. Holander, "The ClustRa Telecomm Database: High Availability, High Throughput, and Real-Time Response", *Proc. 21th International Conference on VLDB,* September 11-15, 1995, Zurich, Switzerland, pp. 469-477.

[Jag+94] H.V. Jagadish et al, "Dalí: a High Performance Main Memory Storage Manager", *Proc. 20th International Conference on VLDB,* September 12-15, 1994, Santiago, Chile, *pp. 48-59.*

[Jen92] C.S. Jensen et al., "A Glossary of Temporal Database Concepts", *ACM SIGMOD Recor*d, 21(3), September 1992, pp. 35-43.

[JSS93] H. V. Jagadish, A. Silberschatz and S. Sudarsan, "Recovering from Main-Memory Lapses", *Proc. 19th International Conference on VLDB,* August 24-27, 1993, Dublin, Ireland, *pp. 391-404.*

[Kiv+99] J. Kiviniemi, A. Wolski, A. Pesonen and J. Arminen, "Lazy Aggregates for Real-Time OLAP", Proc. First International Conference on Data Warehousing and Knowledge Discovery (DaWak'99), Aug. 30 - Sep. 1, 1999, Florence, Italy. Lecture Notes in Computer Science, Springer-Verlag, 1999 (http://www.vtt.fi/tte/projects/industrialdb/publs/lazy-aggr.pdf)

[LN96] S. Listgarten and M.-A. Neimat, "Modelling Consts for a MM-DBMS", *Proc. First Workshop on Real-Time Databases (RTDB'96),* March 7-8, Newport Beach, CA, USA, pp. 77-83.

[MCD89] Dennis R. McCarthy and Umeshar Dayal "The Architecture Of An Active Data Base Management System", *Proc. 1989 ACM SIGMOD Conf.* (Portland, Oregon, USA), pp. 215-224.

[Pat98] N.W. Paton (ed.), "Active Rules in Database Systems", Monographs in Computer Science, Springer-Verlag, 1998.

[Roe99] E. Roe, "A Wrapper Class for Dynamically Linked Plug-Ins", *C/C++ Users Journal,* 17(5), May 1999, pp.27-41.

[Shi+93] H. Shimakawa, H. Ohnishi, I. Mizunuma and M. Tagetaki, "Acquisition and Service of Temporal Data for Real-Time Plant Monitoring", Proc. Real-Time Systems Symposium, Raleigh-Durham, NC, U.S.A., Dec. 1-3, 1993.

[Sno95] R. Snodgrass (ed.), "The TSQL2 Temporal Query Language", Kluwer Academic Publishers, 1995, 674 s.

[SQL-92] ISO/IEC 9075. Information processing systems - Database language SQL. International standard, third edition, 1992. Ref. No. ISO 9075 : 1992 (E).

[SQL-99] ISO/IEC 9075-2. Information processing systems - Database language SQL, Part 2: Foundation. International standard, fourth edition, 1999. Ref. No. ISO 9075-2 : 1999 (E).

[WB98] A. Wolski and T. Bouaziz, "Fuzzy Triggers: Incorporating Imprecise Reasoning into Active Databases", *Proc. 14th International Conference on Data Engineering (ICDE'98),* Feb. 23-27, 1998, Orlando, Florida, pp. 108-155, also at: http://www.vtt.fi/tte/projects/industrialdb/publs/f-triggers.pdf.

[WC96] J. Widom and S. Ceri (eds.), "Active Database Systems: Triggers and Rules For Advanced Database Processing", Morgan Kaufmann, 1996.

[WKP96] A. Wolski, J. Karvonen and A. Puolakka, "The RAPID Case Study: Requirements for and the Design of a Fast-Response Database System", *Proc. First Workshop on Real-Time Databases (RTDB'96),* March 7-8, Newport Beach, CA, USA, pp. 32–39, also at: http://www.vtt.fi/tte/projects/industrialdb/publs/case.pdf.