

# Benefits and Drawbacks of the Special Purpose Database Management System Calanda

Manuel Bleichenbacher, Werner Dreyer, Duri Schmidt

Ubilab, Union Bank of Switzerland  
Bahnhofstr. 45, CH-8021 Zurich

e-mail: {Manuel.Bleichenbacher, Werner.Dreyer, Duri.Schmidt}@ubs.com

*Calanda is a special purpose database management system tailored for the management of time series. In contrast to general purpose database management systems such as relational ones, Calanda is greatly adapted to its problem domain as regards the basic data model, manipulation and query facilities and performance for typical access patterns. The benefits and drawbacks of applying a special purpose database management system will be illustrated by comparing it to time series management with a relational database management system as the most important representative of general purpose database systems.*

## 1 Introduction

The data management research group of Ubilab is constructing Calanda [Dre94a, Dre94b], a database management system (DBMS) specialized in the management of time series. In this report we discuss the advantages and drawbacks of this special purpose database management system (SPDBMS). Calanda is a system under development. It is implemented with ObjectStore from ObjectDesign [Obj96] as its storage manager. No complete implementation exists yet. For this reason, the analysis is based on our experiences with the implementation which is currently (beginning of September 1996) being delivered to the Economic Research Department of UBS. There, it will provide about 30 people with access to 200,000 time series from their spreadsheet software and from their Web-browser. For the problem of replication and directory services issues, we rely on analysis of our design.

Analyzing benefits and drawbacks means comparing the feasible Calanda solutions with solutions based on other systems. In this report we compare Calanda with relational DBMSs (RDBMSs) because they are the most important representatives of general purpose DBMSs (GPDBMSs), and because many organizations still try to manage time series with DBMSs of this kind.

GPDBMSs are built with the goal of solving the data management problems of all application domains. Of course, it would be ideal if a single system could be used for the solution of all data management problems. Users and engineers would only have to master one DBMS and system maintenance would be reduced too. However, according to our experience, one often has to pay for the broad applicability of GPDBMSs with database applications that are inflexible, inefficient and difficult to use and develop.

In contrast to this is the goal which special purpose DBMSs like CALANDA try to achieve. Their goal is to solve a single data management problem thoroughly, in the case of Calanda the management of time series, while neglecting others. The belief is that the drawback of reduced overall applicability is offset by the advantages the system provides in its application domain.

In order to achieve a thorough solution for the data management problems of time series management, Calanda is being built according to the following principles:

- Provide object types which efficiently implement the essential properties and the functionality of the application domain. Consequently, users can work with abstractions they are familiar with from their application expertise, and the complex mapping caused by the gap between the problem domain and the data model of GPDBMSs is avoided.
- Make these object types easily and adequately customizable. This solves the problem that even though the object types are already application domain specific, there is often a need to adapt them to the very special requirements of a particular application.

In the following article we firstly discuss benefits and drawbacks related to data modeling and data manipulation features. Afterwards, we look at benefits and drawbacks related to other aspects. Finally, we draw some conclusions.

## **2 Benefits and drawbacks related to data modeling and data manipulation features**

Adequate data modeling and data manipulation features are a precondition for the successful use of a data base management system in an application domain. The issues involved will be discussed in this chapter. We first concentrate on the requirements and afterwards compare Calanda to RDBMS with regard to these requirements.

### **2.1 Requirements for data modeling and data manipulation features**

In general, a time series consists of a general description – in our parlance the *header* – and of a chronologically ordered collection of *observations* [Dre94b]. Figure 1 shows the two main components of a time series object.

Time series recording one value per observation time point are called *univariate*, while time series recording multiple values are called *multivariate*. Time series with a constant observation interval are called *regular* time series, those with a variable observation interval are called *irregular*. A *calendar* is associated with every time series. It defines the time points for which a time series may have observations. The calendar functionality needed is similar to the requirements for calendars in temporal DBMS [Sno95]. Different time series may be based on different calendars, e.g., a calendar with five business days per week or a calendar with national bank holidays.

In databases with large number of time series, it is difficult for users to find the time series they are interested in if no adequate searching facilities are provided. To ease the search process it is sensible to partition the time series into categories or groups according to various criteria. In a database with historical stock prices, for example, one could establish a group for each stock exchange. These groups could then be split up into industry groups. The industry groups would finally refer to the individual stock price time series. This results in a hierarchical ordering of time series similar to the organization of files in nested directories. A user may then navigate through the group hierarchies to find

the time series relevant to his work. Notice also, that a time series often belongs to more than one category. For this reason it should be possible for a time series to be a member of more than one group. Finally, it is desirable that a group can also store descriptive information about itself in a similar way to the general description of a time series.

name	UBS Bearer			
source	Telekurs			
industry_group	Bank			
date	open	close	high	low
17/7/96	1238	1221	1244	1221
18/7/96	1243	1221	1245	1218
19/7/96	1250	1228	1250	1220

Figure 1. A time series for financial analysis

We have identified four major access patterns:

- 1) Access along the time dimension, i.e., the retrieval of all observations over a range of consecutive dates.
- 2) Retrieve all observations which satisfy a certain condition. Notice that in many cases it is vital to specify conditions which take into account the value of certain prior or later observations.
- 3) Finding members of a group whose header satisfies a certain condition.
- 4) Locating specific time series, retrieving specific attributes of their observations over a range of dates and combining these slices in a table for further processing with a statistics program.

An example of the last query may facilitate understanding: To select stocks for a portfolio, analysts investigate the stock prices of all companies in an industry group over a certain period of time. To do that, they need to extract a table from the time series database that contains the prices for the companies of interest.

Figure 2 shows the table used for analysis. The first column contains the date, each following column contains the stock prices for one company.

A time series management system has to provide facilities to model times series and groups with the structure as explained, it has to support calendars and it must provide query and access facilities with which the various access requests and queries can be expressed easily and which execute the access requests and queries with good performance levels.

Date	UBS Bearer	UBS Reg.	CSH Reg.	SBC Reg.
15.7.	1259	264	129	242
16.7.	1234	261	127	238
17.7.	1228	258	126	239
18.7.	1222	255	127	237
19.7.	1249	258	130	237

Figure 2. Table for comparing companies

## 2.2 Example

For illustration purposes we would like to introduce an example time series: investment research requires financial time series such as historical stock or bond prices. All in all, there are hundreds of thousands of potentially interesting time series.

These financial time series are multivariate. Every observation consists of *open*, *close*, *high* and *low*. Furthermore, *name*, *industry group* and *source* are the descriptive header attributes (see figure 1). The time series have an observation for every business day. Figure 3 summarizes the example time series.

```
Financial time series:  
  Type:      multivariate  
  Calendar:  business days  
  Description: name, industry group, source (strings)  
  Observations: open, close, high, low (floats)
```

Figure 3. Definition of example time series

## 2.3 Basic abstractions in Calanda

One of the basic object abstractions in Calanda are *time series*. They consist of a header to store the general description and a chronologically ordered sequence of tuples to store the observations, closely matching the requirements stated above. The header and every observation is a tuple of simple types and/or arrays of simple types. Eventually, a calendar is associated with every time series in Calanda, again as specified by the requirements above.

The user simply specifies what header attributes, what observation attributes and what calendar a time series type needs to have. Figures 4 and 5 show the graphical user interface of Calanda for defining a new time series type.

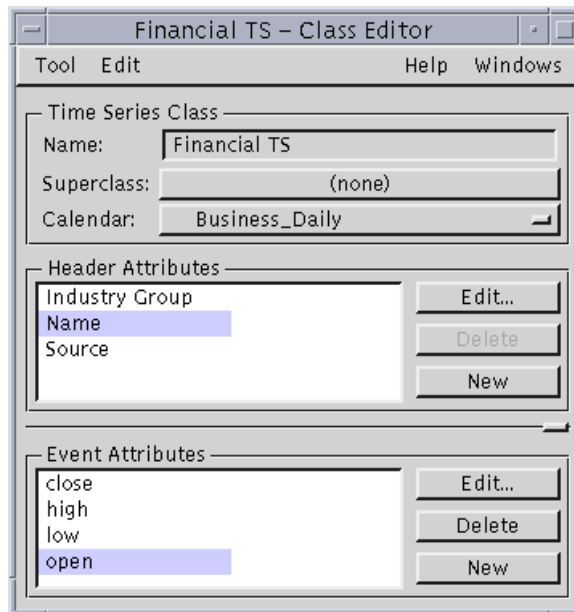


Figure 4. Calanda's class editor

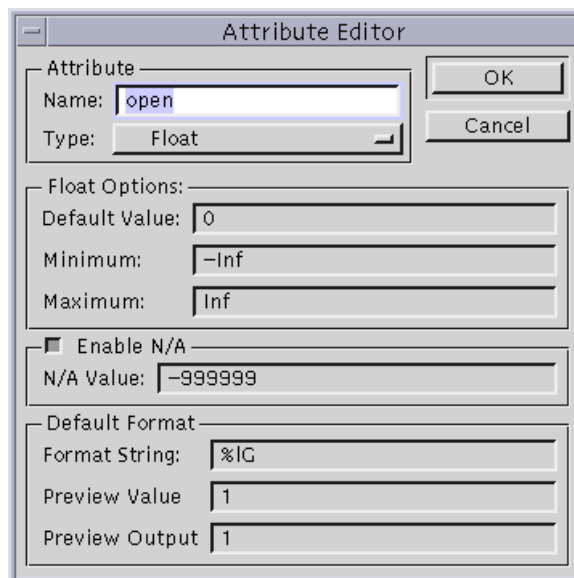


Figure 5. Calanda's attribute editor

Since the concept of time series is known to the type system, there is no semantic gap between the concepts of the problem domain and the data model. No transformations are required. The basic time series type captures the essence of time series: every time series automatically has a header, observations and a calendar. It may then be adapted to the special needs of an application by specifying the header and observation attributes and selecting the calendar.

As explained above, it makes sense to order time series in semantically related groups. In order to do so, Calanda provides *groups* as another basic object abstraction. They contain references to other groups and to time series in their member set. This allows the implementation of hierarchical ordering systems as required above. Navigation through such hierarchies is also supported. Furthermore, every group has a header to store

information about the whole group, in a similar way to the header of a time series. Once again, the basic group type captures the essence of groups, namely, the presence of a member set and a header. A user may then specify the header attributes in order to adapt the group to the requirements of a specific application.

Since Calanda was built specifically for the management of time series, special emphasis was put on efficient storage structure to achieve good performance for common access patterns. The observations are stored as a consecutive vector of tuples. For time series with a regular calendar, the observation date is not stored but can be computed.

It can be seen that the data model of Calanda closely matches the data modeling requirements as stated above. The benefit is that modeling of time series and groups is very easy. Furthermore, the specialized implementation helps to achieve a good query and access performance.

## 2.4 Queries in Calanda

The access and query facilities were also designed with the above requirements in mind. Accessing specific observations over a consecutive range of dates is expressed in our query language as follows:

```
SELECT *
FROM "Zurich/Banks/UBS Bearer"
START D"01-01-1990" END D"31-12-1995"
```

To further restrict the retrieved observations, one can add a *where* clause:

```
SELECT *
FROM "Zurich/Banks/UBS Bearer"
START D"01-01-1990" END D"31-12-1995"
WHERE open > 300
```

The *where* clause may also include sequential predicates that refer to observations before or after the current observation. The selection condition

```
... WHERE close > PREV(close, 5)
```

selects those observations where the closing price is higher than the closing price five periods earlier.

Queries about the members of a group make it easy to find time series satisfying a specific condition:

```
SELECT MEMBERS
FROM "Zurich/Banks"
WHERE name = "UBS Bearer"
      OR name = "CS Registered"
```

Finally, a combined query for the extraction of a table as explained in figure 2 can also be directly expressed with the query facilities of Calanda:

```
SELECT close FROM "Zurich/Banks"
START D"01-01-1990" END D"31-12-1995"
```

This query extracts the closing price for all stocks from the industry group "Banks" which itself is a member of another group "Zurich" from the beginning of 1990 until the end of 1995. The number of columns in the resulting table is dependent on the number of banks in the group "Zurich/Banks". Of course, the query could easily be extended with a condition to select only a subset of all the members in the group.

For an a priori known list of time series the query would be formulated as follows:

```
SELECT close
FROM "Zurich/Banks/UBS Bearer", "Zurich/Banks/CS"
```

```
Registered", "Zurich/Banks/SBC Registered"  
START D"01-01-1990" END D"31-12-1995"
```

Altogether, the specialized query facilities are clearly also an advantage of Calanda because they make it easy to express the common queries and ensure that they are executed efficiently.

## 2.5 Mapping time series and groups onto the relational model

With a RDBMS, time series must be mapped onto relations since the only types available are relations. So the question to be answered during data modeling is how to map time series and groups onto relations [Sch95]. Furthermore, the queries explained above have to be expressed with the set based query facilities available in RDBMSs.

### *Time series*

As mentioned above, a time series consists of two structurally different parts, namely, the header and the observations. In Calanda these parts form a single complex object that can be manipulated as one unit. With a RDBMS this is not possible. Therefore, users of RDBMSs need to worry about all the parts, e.g., for copying a time series.

Another issue concerns unique identifiers. In Calanda every time series automatically has a unique identifier. With RDBMS it is left to the user to provide a unique identifier and to keep track of it.

### *Time series header*

The headers of a single time series type can be stored in a relation with the structure

```
TSTypeHeaders(TSId, attribute1, ..., attributeN)
```

In the case of the financial time series example the relation would be as follows:

```
SecurityHeaders(TSId, IndustryGroup, Source)
```

Modeling the header data is straightforward. Finding time series whose headers satisfy a certain condition is easy as long as they are of the same type. An annoying fact is that users have to know what type the time series has in order to know in which relation its header is stored.

An alternative is to store all the headers in a universal relation style. With this modeling approach one can make queries involving time series of different types. However, the queries become rather complex.

### *Time series observations*

The mapping of the observations we encountered most often is one relation for every time series type. Thus, the relations have the following structure:

```
TSType(TSId, date, attribute1, ..., attributeN)
```

For the financial time series example the relation would look like this:

```
Security(TSId, date, open, close, high, low)
```

To store the identifier and the date with every observation results in a considerable storage overhead. For physical data organization one can choose between a heap organization with an additional combined index on *TSId* and *date* and a clustering index on the same attributes. The latter is preferred since it clusters data on the disk, however, many RDBMSs do not support this. Both organizations add to the waste of disk space since the indices further duplicate data.

The space overhead problem is aggravated in contexts dominated by univariate time series. In some databases this problem is alleviated by putting more than one observation in one row, while the date is only stored for the first value per row. Such a relation would look like this:

```
EconomicTS(TSId, date, value1, ... value10)
```

However, the reduced space requirement is paid for with a considerably increased query complexity. Furthermore, joining such tables to derive multivariate time series is even more complex.

Accessing specific observations over a consecutive range of dates is hindered by the necessary index lookup and by the space overhead. In addition, heap organization distributes consecutive observations over many pages and leads to a further deterioration of data retrieval. Performance is a very real problem when storing time series in RDBMSs.

Less frequently, we have come across other modeling solutions. Since they are not better approaches overall and as a more detailed discussion would be beyond the scope of this report, we will only mention them very briefly.

The solutions for the observations we encountered were:

- 1) creating a separate relation for the observations of every time series,
- 2) combining structurally identical relations in one large relation,
- 3) using one relation for all observations of all time series in a universal relation style,
- 4) splitting multivariate time series into univariate and using one relation per observation attribute type,
- 5) storing the observation attributes of one observation in an unstructured field,
- 6) storing the observations of a whole time series in a BLOB field.

These solutions usually solve one of the problems at the expense of increased query complexity or strongly diminished functionality as in solutions 5 and 6.

Finally, in RDBMS there is no easy way to store arrays as a header or as observation attributes without losing the array semantics, as it is sometimes necessary with derived time series. An example is a time series in which a covariance matrix should be stored together with the estimated values.

### *Groups*

Groups can be modeled in different ways. One approach is to attach category attributes to the header of the time series. This works for a static categorization system, but it does not provide the full functionality of the groups in CALANDA.

Another approach is to fully map groups onto relations in a similar way to the mapping of the time series. One relation per group type can be used for the header as discussed for the time series. Another relation can be used for the member sets. This relation would look like this:

```
Membersets(GroupId, MemberId)
```

Having reviewed all the details, what does our analysis show? It supports our statement that the SPDBMS has strong advantages in its application domain and that RDBMSs may cover some but not all data modeling requirements. With a RDBMS a time series is no more a single complex object. Data modeling is quite complex with a RDBMS and performance usually suffers due to space overhead and complicated mapping.

## 2.6 Querying time series in RDBMS

With SQL, RDBMSs have a powerful query facility. Selecting specific observations from a single time series is as simple as in Calanda, except for the sorting, which is always necessary:

```
SELECT *
FROM "Zurich/Banks/UBS Bearer"
WHERE date >= DATE '1990-01-01'
      AND date <= DATE '1991-12-31'
      AND open > 300
ORDER BY date;
```

Sequential predicates, however, are not supported. This limits the usefulness of SQL for querying time series.

Combined queries are complicated or even impossible in SQL. With an a priori selected list of companies such a query would look like this:

```
SELECT A.date, A.close, B.close,
       C.close, D.close
FROM Security A, Security B,
     Security C, Security D
WHERE A.name = "UBS Bearer"
      AND B.name = "UBS Registered"
      AND C.name = "CSH Registered"
      AND D.name = "SBC Registered"
      AND A.date = B.date AND B.date = C.date
      AND C.date = D.date
      AND A.date >= DATE '1990-01-01'
      AND A.date <= DATE '1991-12-31'
ORDER BY A.date;
```

As we can see, it is rather complicated to formulate the query, and the relation has to be joined with itself several times. Often one would like to investigate dozens or hundreds of securities in this manner, leading to n-way joins with n being too large for efficient handling by RDBMSs. The same query for a varying number of time series is – as far as we can see – not possible with standard SQL since it would require that the list of relation and attribute names is the result of a subquery. The *where* clause would also have to be dynamically adjusted. For this reason, one has to escape to dynamic SQL and external programs.

Overall, these explanations show that the query facilities of RDBMS have not been developed for time series management and that they only cover part of the required functionality.

## 2.7 Calendar

Calanda supports a variety of calendars, taking into account various base calendars (such as Gregorian calendar, 360 days/12 months calendar etc.), different observation intervals (seconds, minutes, hours, weeks, months, years; also multiples and fractions thereof), business and non-business calendars and calendars with local holidays. The functionality includes operations to define all these calendars, to transform time units between calendars, to scan calendars sequentially, to compare and perform arithmetic calculations involving dates and time spans.

GPDBMSs, on the other hand, do not provide comparable calendar functionality. For example, SQL-92 as a standard for RDBMSs defines a *Timestamp* data type, but this is merely a standard for representing a point in time. There is no associated calendar functionality. Thus, the entire calendar functionality would have to be explicitly programmed.

Eventually, nothing would prevent users from joining time series with incompatible calendars, since RDBMS do not support calendars sufficiently.

Accordingly, in the area of calendars the benefits of Calanda are obvious. There is almost no support available in GDBMS and this is a major obstacle for the use of RDBMS or other GDBMS in time series management.

### **3 Benefits and drawbacks related to other aspects**

There are other aspects which also have to be considered to complete the picture, namely integration with other data, connectivity, replication, directory service and corporate standards. All these issues will shortly be discussed, first giving a problem description and then an evaluation of the advantages and disadvantages of Calanda and RDBMS.

#### **3.1 Integration with non-time series data**

Some applications not only need time series but also other, non-time series data. In a portfolio management application, for example, security prices are used, but customer information also.

Calanda is tailored for time series management. Except for the headers of time series and groups, non-time series data cannot be managed. This may result in the usage of another DBMS for storing other data. This makes software development more complex, since the software engineers then have to deal with all the problems related to the use of multiple DBMS such as consistency and integrity problems, distributed transactions, and perhaps manifold storage of the same data in both databases.

Theoretically, GPDBMS can be used for every kind of applications and various kinds of data can be managed by one DBMS. However, the flexibility comes—as we have shown above—at the price of increased modeling complexity, increased query complexity and reduced performance as far as the time series and groups are concerned.

Whether Calanda in combination with another DBMS, or a GPDBMS for all data is the better solution if non-time series data also have to be managed, cannot be simply decided without an evaluation of the application for which the solution has to be provided. Finally, Calanda's integration capabilities could be improved with facilities to make it easier to participate in distributed transactions or to automatically synchronize data residing in different databases.

#### **3.2 Connectivity**

Analysts do not work directly on the server that runs the time series. Instead they want to access data from their personal computer, preferably from within their favorite statistics package. For this reason, the possibility to access a TSMS from various applications is very important.

For PCs running a Windows operating system, Calanda can be accessed via an *OLE automation server* [Bro95]. Many applications are able to communicate with an automation server including Visual Basic, Excel, Word, Delphi and many others. The interface of the OLE automation server supports the basic object abstractions of Calanda: databases, time series and groups. Thus, the semantically rich object abstractions are also retained at the client interface and users do not need to worry about the internal storage organization of

Calanda. This makes it feasible that financial analysts with basic programming knowledge can write custom applications by themselves.

Figure 6 shows the most important objects which the OLE automation server exports. They directly correspond to the objects found in Calanda.

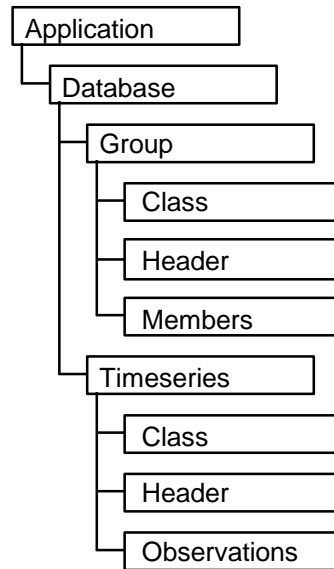


Figure 6. Calanda's OLE Automation Object Hierarchy

A large variety of products for connecting desktop applications to RDBMSs exist on the market. The best supported standard for database connectivity is *Open Database Connectivity* (ODBC) [Gei95]. Many statistics packages, including Excel, can directly retrieve data from an RDBMS via ODBC, and there is an ODBC driver available for almost every RDBMS.

However, getting time series data from an RDBMS into a statistics package is not easy since ODBC only supports relations. Retrieving a time series means extracting data from many relations and then reconstructing it. Many properties of the original time series are lost in this process. In order to reassemble a time series on the client side, one needs intimate knowledge of the mapping of time series onto relations and complex queries must be constructed. Thus, client applications must be built by database specialists. Analysts cannot write scripts to repeatedly retrieve time series data. In addition, the functionality that translates the relational model in the time series model must either be built into every client or a shell must be build around the RDBMS, which is akin to building a SPDBMS with a storage manager inappropriate to the problem. Finally, showing the relations at the interface also means that one cannot change the modeling of time series with relations without breaking all the client applications.

Overall, the special connectivity features of Calanda have a clear advantage over the general purpose facilities of RDBMS. With the Calanda solution one can bring the concepts of the database to the user without falling back on a semantically weaker data model.

### 3.3 Replication

Our analysis [Dre96] showed that economic and financial researchers want their own project databases and that they want to have the full autonomy over these databases. To ease

the setup and maintenance of these databases, a time series management must provide replication services which cover the following requirements:

- A researcher must only specify what time series to replicate. The system should then handle all the details.
- In principle, it must be possible to replicate time series from any other time series database.
- It must be possible to designate and revoke time series for replication at any time.
- It must be possible to start and terminate the replication of a time series at any time.
- A user must be free to choose the replication interval for updates at his will.

Calanda incorporates replication services [Dre96] that are exactly tailored to meet above requirements. To cope with a dynamically varying set of replicated time series, the replication mechanism is based on a subscription mechanism. An owner of a time series database may publish time series or revoke publication at any time, and a user may subscribe to a new time series or stop the subscription at any time. The scheduling facility provides the necessary flexibility by allowing the definition of individual schedules for each user and each replicated time series. A user only has to specify what time series he or she wants to replicate. If necessary, Calanda automatically handles the required schema replication at the begin of the subscription and when the schema of the source time series is changed.

For GPDBMSs, the situation is different. There exists a set of replication systems designed to be used with commercial DBMSs, mainly RDBMSs [Moi95, Ora95, IBM94, Com94, Mic95, Tri93, Pra94]. As these DBMSs are normally used for applications and data that differ greatly from our requirements, from our point of view, their replication systems also have considerable disadvantages for the intended usage.

These systems lack the flexibility we need. They statically define which databases participate in a replication, and which data are to be replicated. Furthermore, they employ a static replication setup, that is, the schema of the replicated data cannot be changed while the system is running. Lastly, the scheduling facilities are rather limited. This lack of flexibility is not a major problem for typical applications that are based on RDBMSs. In such applications, it can be decided in advance which data are to be replicated, the schema remains unchanged, and the scheduling is simple too.

In all, time series management has special requirements concerning replication. Calanda satisfies them accordingly, while the replication services of GPDBMS are targeted to other application domains with other requirements.

### **3.4 Directory services**

In an organization such as UBS many different time series databases exist and most users need time series from more than one database. For this reason users need an easy way to find out what time series are available in which database.

To ease the search, Calanda will incorporate a directory that stores metadata about all the published time series [Dre96]. Beyond the system defined metadata, an owner of a time series can declare arbitrary header attributes as metadata. Users can query the metadata and the directory server delivers the metadata of all time series that match. Users can subscribe to the resulting time series in the same way as they would when directly

browsing through a time series database since the directory server keeps track of the origin of the metadata.

The directory itself, of course, can be replicated at various locations. Mutual updates of directories are realized by the same infrastructure that replicates time series.

The currently available GPDBMSs do not support the concept of metadata and they do not provide any directory services. The main reason is that GPDBMS are not designed for this type of applications where the data is stored in many databases and where the appropriate data is searched in an exploratory way.

Considering that directory services do not exist in GPDBMS, it is obvious that Calanda has an advantage also in this problem area and that GPDBMS do not cover at all the requirements concerning directory services.

### **3.5 Single DBMS as a corporate standard**

This point may be seen as a non-issue from an academic point of view, but in real life it is often a considerable problem for the application of a SPDBMS such as Calanda. Many companies try to settle on one DBMS product as a corporate standard and there are good reasons for keeping the number of DBMSs in a company to a minimum. Having an additional DBMS in the tool portfolio can mean significant additional costs for application development and system support. Application developers and system administrators need additional know-how of the new system. Furthermore, the effort involved in installing new releases of a DBMS and adapting applications accordingly is substantial. Lastly, the engagement of several systems multiplies the license costs.

It should be clear now that a SPDBMS really has to provide substantial benefits, as introducing a new DBMS to an organization could kill of the SPDBMS altogether.

## **4 Conclusions**

Our analysis showed that in the time series management application domain Calanda has benefits in areas such as data modeling, queries and other data manipulation features, connectivity, replication, and directory service. On the other hand, it has drawbacks in the area of integration with other data and it may deviate from corporate standards.

To minimize or even overcome the data integration problem, additional functionality should be added to Calanda. How this functionality should look like, i.e., whether Calanda should be extended in order that it could participate in distributed transactions or whether a data reconciliation mechanism should be considered, has to be analyzed.

## **Reference**

- [Bro95] Brockschmidt K. *Inside OLE, 2nd ed.* Microsoft Press, Washington, 1995
- [Com94] Computer Associates, Inc.: *CA-OpenIngres/Replicator*. Product Description, 1994
- [Dre94a] Dreyer W, Kotz Dittrich A, Schmidt D: *Research Perspectives for Time Series Management Systems*. SIGMOD RECORD, Vol. 23, No. 1, March 1994
- [Dre94b] Dreyer W, Kotz Dittrich A, Schmidt D: *An Object-Oriented Data Model for a Time Series Management System*. Proceedings of the 7th International Working

- Conference on Scientific and Statistical Database Management, Charlottesville, Virginia USA, Sept. 28-30, 1994
- [Dre96] Dreyer W, Schmidt D, Kotz Dittrich A, Bleichenbacher M: *Research Perspectives for Time Series Management Systems*. 8th International Working Conference on Scientific and Statistical Database Management; Stockholm, Sweden, June 18-20, 1996, pp. 208-215
- [Gei95] Geiger K: *Inside ODBC*. Microsoft Press, Washington, 1995
- [IBM94] IBM Corp.: *Data Replication: The IBM Solution*. IBM White Paper, May 1994
- [Ill95] Illustra Information Technologies, Inc.: *Illustra TimeSeries DataBlade Guide*. 1995
- [Mic95] Microsoft, Inc.: *SQL Server Data Replication*. <http://www.microsoft.com/SQL/sqlrevg4.htm>, 1995
- [Moi95] Moissis A: *Sybase Replication Server: A Practical Architecture for Distributing and Sharing Corporate Information*. Sybase White Paper, [http://www.sybase.com/Products/Whitepapers/repserver\\_wpaper.html](http://www.sybase.com/Products/Whitepapers/repserver_wpaper.html), 1995
- [Obj96] ObjectDesign: <http://www.odi.com>
- [Ora95] Oracle: *Oracle7 Asynchronous Distributed Capability Overview*. Oracle White Paper, <http://www.oracle.com/info/products/symrep/chapter4.html>, 1995
- [Pra94] Praxis International, Inc.: *OmniReplicator: Concepts and Facilities*. Praxis International, Inc., 1994
- [Sch95] Schmidt D, Kotz Dittrich A, Dreyer W, Marti R: *Time Series, a Neglected Issue in Temporal Database Research?*. Proceedings of the Int. Workshop on Temporal Databases, Zurich, Switzerland, Sept. 17-18 1995, pp. 214-232
- [Sno95] Snodgrass R (Ed.): *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995
- [Tri93] Trinzic Corp.: *InfoPump: Client/Server Middleware for Routing, Integrating and Synchronizing Dissimilar Data*. Trinzic Product Description, 1993