Event-Feeded Dimension Solution

Tho Manh Nguyen[1,2], Jaromir Nemec[1], and Martin Windisch[1]

[1] T-Mobile Austria, Rennweg 97-99, A-1030 Vienna, Austria
{tho.nguyen, jaromir.nemec, martin.windisch}@t-mobile.at
[2] Institute of Software Technology and Interactive Systems, Vienna Uni. of Technology
Favoritenstr. 9-11/188, A-1040 Vienna, Austria
tho@ifs.tuwien.ac.at

Abstract. From the point of view of a data warehouse system its part of collecting and receiving information from other systems is crucial for all subsequent business intelligence applications. The incoming information can be classified generally in two types, the state-snapshot data and the state-change or event data usually called transactional data, which contains information about the change processes applied on the instances of information objects. On the way towards active data warehouses it becomes more important to provide complete data with minimal latency. We focus in this paper on dimensional data provided by any data-master application. The information transfer is done via messages containing the change-information of the dimension instances. The receiving data warehouse system is able to validate the event-messages, reconstruct the complete history of the dimension and provide a well applicable "comprehensive slowly changing dimension" (cSCD) interface for well-performing queries on the historical and current state of the dimension. A prototype implementation of "active integration" of a data warehouse is proposed.

1 Introduction

The upcoming integration technology standards [12,13] based on message exchange between information systems provide benefits not only to operative systems. Also non-OLTP systems like data warehouses can gain some profits out of this development [4]. In the past a restricted integration of the source systems traditionally led to batch-oriented data load approaches for data warehouses.

This situation is also in place at T-Mobile Austria, where we have done the analysis and development of the solution, proposed in this paper. The data warehouse at T-Mobile Austria runs on an Oracle 9.1.3 relational database and has a data volume of about six terabyte (TB). The complexity and number of operational source systems is very high. Therefore, the data warehouse provides its information as a single point of truth for nearly all units of the enterprise.

The data freshness for transactional data is very high, although the data is currently batch loaded. CDRs (call detail records) are usually loaded every four hours. The dimensional data loaded from legacies like the billing system (BSCS, Business Support & Control System), SAP or the CRM Systems is received via daily snapshots.

A Min Tjoa and J. Trujillo (Eds.): DaWaK 2005, LNCS 3589, pp. 22 – 31, 2005.

Because of the limitations of the snapshot based approach[1], a more efficient approach being more near-real time is considered. Data changes in the operational sources are captured and provided near real time as event messages via the event-based infrastructure TIBCO [12]. This approach implies also the discussion of the message content and its validation. Mainly three quality aspects are under inspection: the completeness, uniqueness and order of the event-messages.

For the further processing of the event-messages we developed one comprehensive and general applicable SCD representation inspired by Kimball's three SCD-types [7] and propose a valid alternative to snapshot based information transfer. This solution is applicable especially in cases, where the information requirements of the receiving system focus on complete and detailed historical information for all instances combined with a minimal latency demand. For a given latency time-interval the advantages of this method are obvious for a dimension with a small number of changes compared to its cardinality.

The proposed method provides much more than a data replication. The primary target is not a physical mirror of the dimension object. Moreover all necessary views including the change history of this object are implemented in a standardized way with quite realistic efforts. The event feeded cSCD approach has been designed according to the main goals of T-Mobile Austria's data warehouse, which are simple: "to provide a single point of truth easy to access".

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 introduces the Event Model concerning the event and event processing descriptions. The Event-feeded cSCD prototype implementation is described in Section 4. Finally, in Section 5, we present our conclusion and the future work.

2 Related Works

Active data warehouses [4,13] prefer to provide complete data within minimal latency. The well known limitations of processing dimensional snapshot-data [11] can be overcome by the proposed method, which is preferred for a dimension with a small number of changes compared to its cardinality because of less resource consumption. The complete history of the dimensional change events is also an advantage compared with historical (periodic) snapshots.

In some cases daily snapshots [1] have been used to provide change information out of the differences of two consecutive snapshots. To hold an appropriate history of such dimensions the only way was, to store the received snapshots chronologically, which means, that the storage request for the historical data does primarily not depend on the change fluctuation and volatility of the instances. One can apply in a second step some compression algorithms to overcome these disadvantages.

R. Kimball [7] has introduced the slowly changing dimension (SCD) types 1, 2 and 3 to track changes in dimension attributes. In SCD type 1, the changed attribute is

[1] Multiple change events between snapshots miss completely. For each snapshot comparison the number of records to process is high, requiring also high computing-resources and -time and the history is kept by tremendous daily snapshot versions consuming a lot of storage.

simply overwritten to reflect the most current value thus does not keep the historical changes. SCD type 2 creates another (dimension) record to keep trace the changed attributes, but could not keep the old value both before and after the change. For this purpose, the SCD type 3 uses the "current value" and "previous value" but it isn't appropriate for the unpredictable changes.

R. Bliujute et al. [2] suggested the temporal star schema to overcome the SCD issues with event and state-oriented data. They tackled the SCD type 2 in fixed attributes with timestamp. We instead propose a more general event model where the target dimensional object can be fine tailored depending on business requirements.

The flexibility in choosing the event timestamp (e.g. between transaction timestamp and processing timestamp) enables in the proposed cSCD representation the handling of time-consistency issues as discussed by R. Bruckner et al. [3].

J. Eder et al. [8] propose a temporal multi-dimensional data model to cope with the changes of dimension via multi versions and valid time. Our purpose is keeping track not only the versions of instances but also their relationships in dimensional data.

W. Inmon recommends the usage of normalized dimensions [6]. This is a very important aspect as the event based maintenance of denormalized dimensions although possible is not very practical [10].

3 Event Model

For a formal description of an event and event processing a UML based model is created. The core part of this model is an UML profile describing the event metamodel. Additionally, the semantic of event is defined and shown by a simple example. Possible strategies of event interpretation are discussed.

Based on the defined model and the interpretation of the event a broader discussion is performed, demonstrating that the traditional distinction between fact and dimension in event based DWH environment represents only a different specialization of our event based model.

3.1 Profile

To describe a general event it is necessary to raise the model to the meta level M2 [9] as the structure of each event type is very proprietary based on the transferred business information. The simplified profile definition is depicted in Fig. 1.

The key concepts of the event profile are as follows:

- Stereotype «Event» describes the object containing the event data.
- Stereotype «Efd» (abbreviation for event feeded dimension) depicts the target object that is maintained via the event stream.
- Stereotypes «Trans» and «Snap» as subtypes of «Efd» are discussed below

Those stereotypes are applicable on the class level, the rest of stereotypes are connected with an attribute:

- Stereotype «Key» is used to mark the (natural or surrogate) primary key of the dimension

- Stereotype «Order» is intended to define the order in which the events were created and should be processed.
- Stereotype «Timestamp» identifies an attribute containing the timestamp information of an event. The transaction time, event creation time, event processing time are various examples of this stereotype.
- Stereotype «Action» describes the nature of the change represented in the event (insert / update / delete)
- Stereotype «Status» enables the depiction of a logical deletion of a dimension instance.

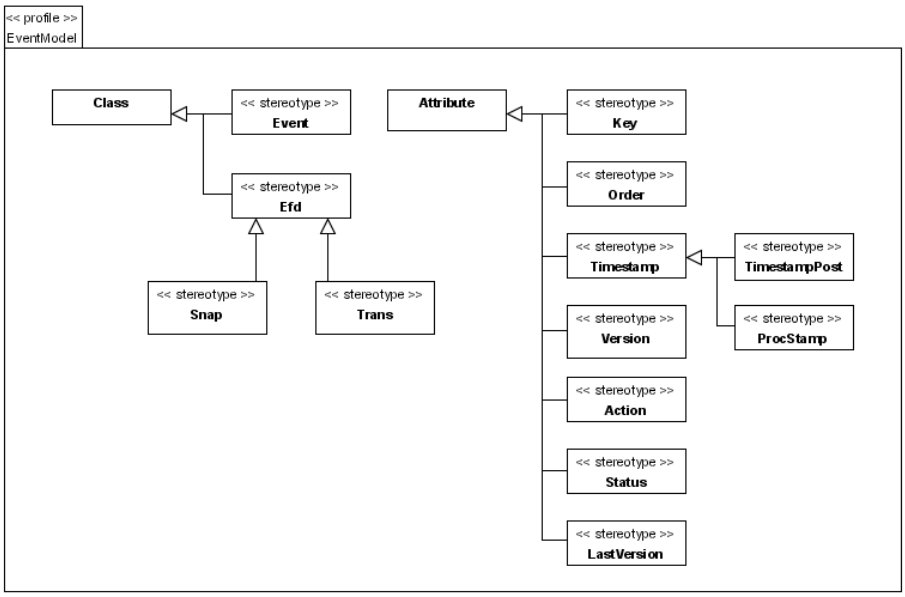

Fig. 1. Simplified Profile Event Definition

Not all of the listed stereotypes are mandatory, the usage is constrained by semantic rules (see below) such as: The «Event» and «Efd» classes must contain at least one *Key* attribute (i.e. an attribute with stereotype «Key»). *Order* and *Timestamp* attributes may coincident, e.g. in cases when the time grain is to large to distinct the events uniquely, the *Order* attribute is used to define the unique event sequence. The opposite extreme when neither of those attributes is defined is also valid. In that case the timestamp of event processing can be used as a default *Timestamp* attribute (of course the unique order of events must be established in this case as well).

3.2 Example

To illustrate the usage of Event profile lets consider a simplified application that maintains the customer attributes via an event interface. The customer is identified with an attribute id, the customer attributes consist of name, address and tariff.

The class with associated stereotype *Event* describes the customer-value-change event. This event is generated on each change of at least one attribute of a particular customer. As marked with stereotype *Key* the primary key of the customer dimension is the attribute id. The attribute timestamp is stereotyped as *Timestamp* i.e. this attribute defines the point in the time of the change of customer attributes. The rest of attributes have no stereotypes they are regular event attributes containing additional information.

The second class in Fig. 2 describes the target object maintained via the event feed (stereotype *Trans* defines that a full versioned history of the target object will be build; see the detailed discussion in 3.3). The meaning of the additional attribute is discussed below.

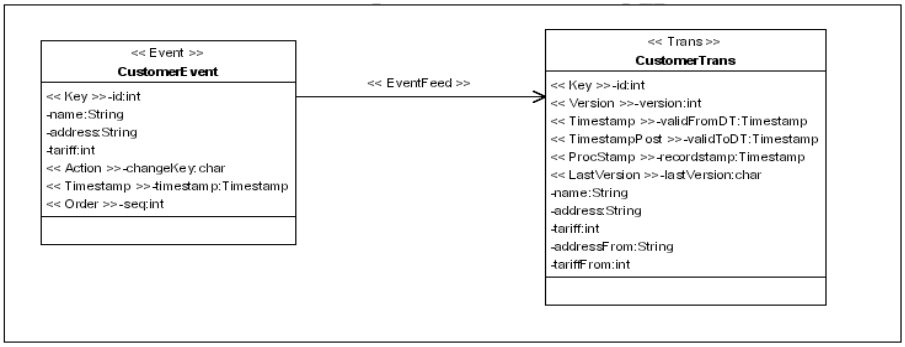

Fig. 2. Customer Event Profile Example

3.3 Event Processing

The profile based event model must be enriched with semantic rules defining the interpretation of an event. The most important feature is the sub-typing of the *Efd* object. In the profile two main examples are defined *Snap* and *Trans*.

The Snap object is maintained with overwrite policy, i.e. new records are inserted; existing records are updated or deleted. In a *Snap* object only one record per primary key is stored. *Snap* is mnemonic for dimension snapshot.

The *Trans* object is maintained cumulatively, each event is added to the target object, building a complete transactional history of the dimension.

The handling of primary key of the build dimension can be configured. The Primary key option defines if the target object uses the natural key (as provided within the event) or if a surrogate key should be generated while the event is processed. In any case the primary key always uniquely identifies the dimension instance, so if a complete history of the dimension is maintained an additional attribute stereotyped as «Version» must extend the primary key of the target table.

Other option is defined on the level of attribute; an attribute noted as *Timestamp-Post* is applicable for *Trans* object only. It is filled with the value of the corresponding *Timestamp* attribute of the successor version decreased by the smallest grain of

the time dimension (e.g. 1 ms). The default value is an artificially set high date (e.g. 31-12-9999 00:00:00). The usage of two timestamps in a full history table is not a "pure relational" design but extreme practical solution as for the selection of a version of a particular dimension occurrence a simple logic can be applied (*required timestamp* between *Timestamp* and *TimestampPost).*

If an attribute has a suffix *From* it contains the value "before the change", i.e. in *Trans* object this is the value stored in the preceding version. The association between the corresponding attribute is established with naming conventions.

A different aspect of semantic is the validation of the event model, i.e. the verification if the model is well-formed. Examples of constraints that must be checked are listed below:

- Event class must have at least one *Key* attribute
- Each *Timestamp* attribute must have a type compatible with date/time.

The final role of semantic in event context is the event validation. It is possible to extend the event data with redundant information that can be checked while the event is processed. The exceptions can be interpreted as an advice of lost or corrupted events.

For examples adding an *Action* attribute to the event (possible values: insert / update / delete) enables additional checks:

- key must exists in the target object on update and delete
- key must not exists in the target object on insert

Other types of validation can be alternatively implemented as services on the event transport layer, e.g. guaranteed delivery or de-dup filtering [5].

4 Event-Feeded cSCD Implementation

The described implementation represents a particular instantiation of the presented model in Section 3. The target object is implemented as a *Trans* table; natural key option is used; *Action* and *from attributes* are supported.

4.1 Development Environment

Because the target DWH is also based on Oracle DBMS, we decided to keep the current development environment, i.e. developing the event feed cSCD solution as an Oracle PL/SQL package and call easily the functionality from ETL (e.g. Informatica Powercenter) mappings.

4.2 The UTL_EVENT_SCD Package

The package can be used to trace the changing attributes of any (dimensional) table. It accepts a variant of parameters for the detailed configuration of the event-processing and -correction such as *traced entity* (via table name parameter), *correct option* (optional, mandatory or automatically), *refresh option* (incremental or from scratch), *filer criteria*, etc.

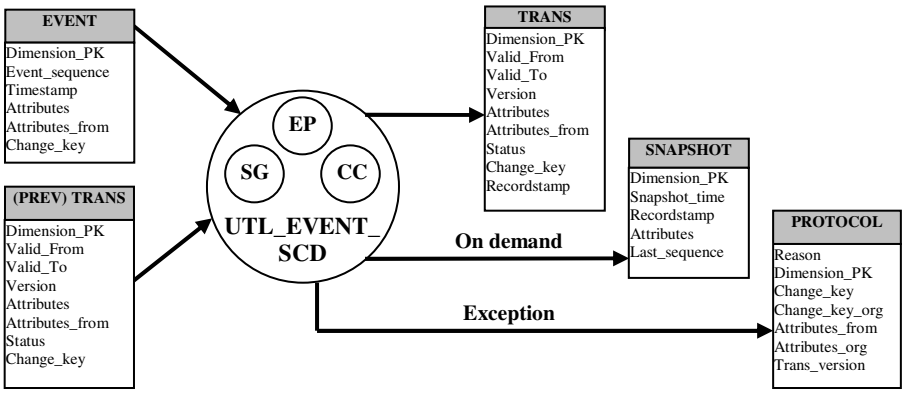

Fig. 3. UTL_EVENT_SCD Package modules and its related tables

The package (Fig. 3) contains 3 main modules: Event Processing (EP), Snapshot Generation (SG), and Consistency Checking (CC) providing the following options:

- Validating the events before refreshing the historical transactions of the entity instances (update TRANS table) with full historical tracing and versioning.
- Providing the state information at any point in time for any instance or subset of instances (generate SNAPSHOT table on demand)
- Checking the consistency between the entity state data of the legacy system and the data in DWH, and solving the inconsistency issue.

4.2.1 Event Processing (EP)

EP applies event data and refreshes the TRANS table as follows. It first accesses the event data, filters those which happened since the last refresh (i.e. those records which do not appear in TRANS or have different states with the current records in TRANS). The event validation then checks the events with automatic correction options to override some invalid events. This validation and correction processes are based on some useful attributes such as *change_key, attribute_from* or *sequence order*. The invalid or overridden events are kept in the PROTOCOL table.

The valid events are used to refresh the TRANS table. For each event data related to an entity instance, an equivalent transaction record in TRANS table is created. If there are other events related to the same entity, the extended SCD type 2 [1,5] is applied to keep trace over all transactions (with versions). The TRANS table thus contains the complete transaction history of dimension changes.

Examples: We apply the UTL_EVENT_SCD package to trace the Customer's attribute changes. Suppose that we have currently two customers Robert and Sonja until 7 am,14/02/2005. At 7:10, Robert informs that he changes his address from 20 Rennweg to 25 Favoriten. 7:12 am, a new customer Micheal has registered into the system, and Robert changes his tariff from type 1 to type 2 at 7:13. At 7:14, Sonja changes her tariff from type 2 to type 1. The UTL_EVENT_SCD package is executed at 7:15 to refresh the previous TRANS table. (Fig. 4)

CUST_TRANS (Before Event applying)

ID	Valid_from	Valid_to	Name	Address	Tariff	Address_from	Tariff_from	Recordstamp	version	Last_version	Change_key
1	14-02-2005 07:00:00	31-12-9999 00:00:00	Robert	20 Rennweg	T1			14-02-2005 07:00:00	1	Y	I
2	14-02-2005 07:00:00	31-12-9999 00:00:00	Sonja	15 Kargan	T2			14-02-2005 07:00:00	1	Y	I

CUST_EVENT

ID	Timestamp	Seq	Name	Address	Tariff	Address_from	Tariff_from	Change_key
1	14-02-2005 07:10:00	1	Robert	25 Favoriten	T1	20 Rennweg		U
3	14-02-2005 07:12:00	2	Micheal	10 Rathaus	T2			I
1	14-02-2005 07:13:00	3	Robert	25 Favoriten	T2		T1	U
2	14-02-2005 07:14:00	4	Sonja	15 Kargan	T1		T2	U

CUST_TRANS (After Event applying)

ID	Valid_from	Valid_to	Name	Address	Tariff	Address _from	Tariff _from	Recordstamp	version	Last_version	Change_key
1	14-02-2005 07:00:00	14-02-2005 07:09:59	Robert	20 Rennweg	T1			14-02-2005 07:15:00	1	N	I
1	14-02-2005 07:10:00	14-02-2005 07:12:59	Robert	25 Favoriten	T1	20 Rennweg		14-02-2005 07:15:00	2	N	U
1	14-02-2005 07:13:00	31-12-9999 00:00:00	Robert	25 Favoriten	T2		T1	14-02-2005 07:15:00	3	Y	U
2	14-02-2005 07:00:00	14-02-2005 07:13:59	Sonja	15 Kargan	T2			14-02-2005 07:15:00	1	N	I
2	14-02-2005 07:14:00	31-12-9999 00:00:00	Sonja	15 Kargan	T1		T2	14-02-2005 07:15:00	2	Y	U
3	14-02-2005 07:12:00	31-12-9999 00:00:00	Micheal	10 Rathaus	T2			14-02-2005 07:15:00	1	Y	I

Fig. 4. TRANS table refresh after UTL_EVENT_SCD package execution

The investigation of the performance behavior based on the prototype implementation showed a near linear scalability of the processing-time per event with an average throughput of about 300 TRANS-records per second on a dimension with the cardinality of one million records. The minimum refresh period is about 3-4 seconds caused by process overheads. However, with the high number of events (e.g. over 20000 events), the more events accumulated, the less efficient of the event-SCD approach compared to the snapshot based SCD approach.

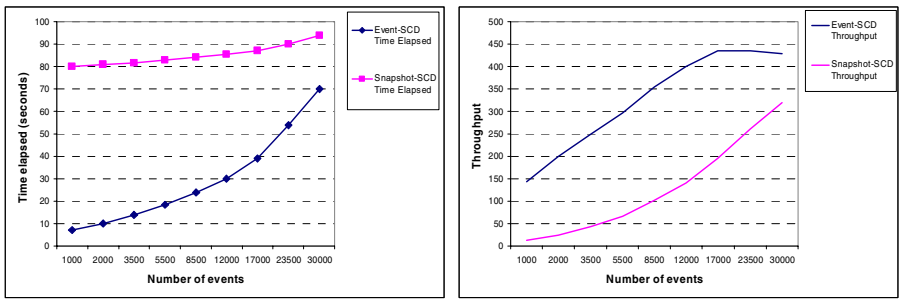

Fig. 5. Elapsed processing time and performance throughput comparison between event-SCD and snapshot based SCD approach

4.2.2 On Demand Snapshot Generation (SG)

Despite the series of snapshots is not kept as previously, the requirement to have a snapshot at one point in time for any subset of entity instances remains. From the TRANS table, we can rebuild these required snapshots. The package provides two options to generate a snapshot: (1) from scratch (Fig. 6) and (2) based on an existing snapshot, further referenced as based snapshot (Fig.7). The generated Customer snapshots at 7:00 and 7:15 are shown in Fig. 8.

```
CREATE TABLE CUST_SNAP AS
SELECT ID, i_timepoint as Snaptime, Name,Address, Tariff
FROM CUST_TRANS
WHERE CHANGE_KEY <> 'D' AND
i_timepoint BETWEEN VALIDFROM_T AND VALIDTO_T;
```

Fig. 6. Create Snapshot from scratch (i_timepoint is the time point of the snapshot data)

```
CREATE TABLE CUST_SNAP AS
SELECT * FROM
(SELECT ID,i_timepoint as Snaptime, Name, Address, Tariff
 FROM CUST_TRANS WHERE CHANGE_KEY <> 'D' AND
 i_timepoint BETWEEN VALIDFROM_T AND VALIDTO_T
 AND VALIDFROM_T > v_prev_time
UNION ALL
 SELECT ID,i_timepoint as Snaptime, Name, Address, Tariff
 FROM BASED_CUST_SNAP
 WHERE ID NOT IN
       (SELECT ID FROM CUST_TRANS WHERE
        i_timepoint BETWEEN VALIDFROM_T AND VALIDTO_T
        AND VALIDFROM_T > v_prev_time)
 );
```

Fig. 7. Create Snapshot from based snapshot (BASED_CUST_SNAP is the based snapshot table, v_prev_time is the time point of the based snapshot data)

SNAPSHOT at 14-02-2005 7:00				
ID	**Snaptime**	**Name**	**Address**	**Tariff**
1	14-02-2005 07:00:00	Robert	20 Rennweg	T1
2	14-02-2005 07:00:00	Sonja	15 Kargan	T2

SNAPSHOT at 14-02-2005 7:15				
ID	**Snaptime**	**Name**	**Address**	**Tariff**
1	14-02-2005 07:15:00	Robert	25 Favoriten	T2
2	14-02-2005 07:15:00	Sonja	15 Kargan	T1
3	14-02-2005 07:15:00	Micheal	10 Rathaus	T2

Fig. 8. SNAPSHOT tables generated at 7:00 and 7:15

4.2.3 Consistency Checking and Recovery (CC)

In the event based cSCD approach, an inconsistent state could be detected when we are able to access on a truthful snapshot source (usually provided from the legacy systems). The input requirements of this process are the mandatory truthful snapshot (S_i, t_j) table and the metadata parameters describing the record-structure. The consistency checking process compares a truthful snapshot(-part) taken on any subset of instances S_i, at any point of time t_j with the corresponding on demand snapshot (S_i, t_j) (see Section 4.2.2) which is temporary stored in a TEMP_SNAP (S_i, t_j) table. The found inconsistencies between the snapshots are applied again as new change events to correct the TRANS records.

5 Conclusion and Future Work

The event feeded cSCD approach presented in this paper significantly reduces the number of records to be processed compared to the snapshot based approach. Besides, compared with the Kimball's classification of SCD [7] we see that the SDC types 1,2

and 3 are only examples of possible instantiations of the proposed cSCD approach (SDC 1 and 2 respectively use the *Snap* object without and with *from attributes*; SDC 3 is based on *Trans* object without *from attributes).*

Although the target object was up to now considered as a dimension, this is not a limitation of the proposed model. A typical fact table can be described also as a versioned dimension (fast changing dimension), using the add-version update policy (each event creates a new record in the fact table) with appropriate validation e.g. to maintain a balance attribute.

Further more extending our model with summarizing stereotypes (e.g. add the actual value of the attribute to the previous value) the way is opened for describing running aggregates. On the other hand the correlation of system-dependent event-messages as an alternative to the join of dimensional snapshots needs further inspection.

Acknowledgement

This research was funded by T-Mobile Austria and supported by the IT department providing the needed infrastructure and environment.

References

1. Arun Sen, Atish P. Sinha, A Comparison of Data Warehousing Methodologies. Communications of the ACM, Vol. 48, No. 3, March 2005.
2. Bliujute, R., Saltenis, S., Slivinskas, G., and Jensen, C.S. (1998). Systematic Change Management in Dimensional Data Warehousing. In Proc. of the 3rd Intl. Baltic Workshop on Databases and Information Systems , Riga, Latvia, (pp. 27-41).
3. Bruckner R., Tjoa A., Managing Time Consistency for Active Data Warehouse Environments. DaWaK 2001, Springer-Verlag LNCS 2114, pp. 254–263, 2001.
4. Brobst, S., Enterprise Application Integration and Active Data Warehousing, In Proc. Data Warehousing 2002, pp. 15-22, Physica-Verlag 2002.
5. Hohpe G., Woolf B., Enterprise Integration Patterns, Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, 2004
6. Inmon, W., Building the Data Warehouse; Jon Wiley & Sons, Second Edition, 1996
7. Kimball R. et al., The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd Edition, John Wiley & Sons, 2002.
8. Koncilia, C., Eder, J., Changes of Dimension Data in Temporal Data Warehouses, DaWaK 2001, Springer-Verlag LNCS 2114, pp. 284–293, 2001.
9. Meta Object Facility (MOF) Specification http://www.omg.org/docs/formal/00-04-03.pdf
10. Rieger B., Brodmann K., Mastering Time Variances of Dimension Tables in the Data Warehouse, Osnabrueck University, 1999
11. Rocha R., Cardoso F., Souza, M., Performance Tests in Data Warehousing ETLM Process for Detection of Changes in Data Origin. DaWaK 2003, LNCS 2737, pp. 129-139, 2003.
12. TIBCO Software Inc.: http://www.tibco.com
13. Vandermay J., Considerations for Building a Real-time Oracle Data Warehouse, DataMirror Corporation White Paper,2000.