Available online at www.sciencedirect.com

ScienceDirect

Data & Knowledge Engineering 63 (2007) 26–43

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

An approach towards an event-fed solution for slowly changing dimensions in data warehouses with a detailed case study

Tho Manh Nguyen [a,*], A. Min Tjoa [a], Jaromir Nemec [b], Martin Windisch [b]

[a] *Institute of Software Technology, Vienna University of Technology, Favoritenstr. 9-11/188, A-1040 Vienna, Austria*
[b] *T-Mobile Austria, Rennweg 97-99, A-1030 Vienna, Austria*

Received 13 October 2006; received in revised form 13 October 2006; accepted 13 October 2006
Available online 13 November 2006

Abstract

From the point of view of a data warehouse system, collecting and receiving information from source systems is crucial for all subsequent business intelligence applications. Incoming information can generally be classified into two types: (1) the state-oriented data and (2) event-oriented data or transactional data, which contains information about the change performed by processes on the instances of information objects. On the way towards achieving the goal of a full-fledged active data warehouse it becomes more and more important to provide data with minimal latency. In this paper we focus on dimensional data which is provided by general data warehouse applications. The information transfer is performed via messages containing the change of information on the dimension instances. The proposed approach 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 queries on the historical and current state of the dimension. A description of the prototype implementation for this kind of an "active integration" in a data warehouse and a case study at T-Mobile conclude the paper.

Keywords: Active data warehousing; Slowly changing dimension; Event-based data integration; Data refresh

1. Introduction

The presence of time and the dependence upon it is one of the properties that sets special attention in data warehouse applications apart from traditional operational systems. Time is one of the four basic characteristics of a Data Warehouse as pointed out by Inmon [8]. In Data Warehousing the explicit and inherent existence of the time dimension enables the handling of historical data and time-dependent analysis (e.g. analysis of time series). This means that users of data warehouses can analyse aspects of their organization at every specific point or over every period of time for which historical data is recorded. Consequently this enables

* Corresponding author.
E-mail addresses: tho@ifs.tuwien.ac.at (T. Manh Nguyen), amin@ifs.tuwien.ac.at (A. Min Tjoa), jaromir.nemec@t-mobile.at (J. Nemec), martin.windisch@t-mobile.at (M. Windisch).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2006.10.004

the observation of behavioural patterns in the course of time which capacitate comparisons between similar or dissimilar periods, e.g. this year versus last year, seasonal trends. Ultimately forecasting can be regarded as one of the most important managerial issues deals, i.e. using information of the past to predict the future.

We can clearly discern the temporal requirements (i.e., time based historical presentation of data) of a data warehouse from those of an operational system. While in a data warehouse it is required to cover the full historical nature of data, operational systems are only aimed to keep the current or very recent data. Therefore, we must periodically feed information about changed data to the data warehouse to refresh the data warehouse. A major role of the data warehouse is to keep the historical data consistent to provide the correct information at any point in time the user requires. Until recently restricted integration of source systems has led to batch-oriented data load approaches for data warehouses. The data warehouse refresh is thus performed in batch mode when it receives multiple operational source snapshots, extract the changes, and refresh the warehouse data. Upcoming integration technology standards [7,21] based on message exchange between information systems not only provide substantial benefits to operative systems, but also for the data warehouses [2].

However, there exist some significant limitations of this traditional snapshot-based approach. Most importantly multiple change events between snapshots are completely ignored. Furthermore, for each snapshot comparison the number of records to be processed is high, requiring high computing-resources and -time and finally the history data is kept by very large daily snapshot versions consuming a huge amount of storage. Therefore, a more efficient approach towards a near-real time solution builds an essential research challenge in the area of data warehousing.

In this paper, we introduce the research work at T-Mobile Austria to develop an event-based refresh paradigm to solve the problem of updates in slowly changing dimension (SCD). For the further processing of event-messages, we develop one comprehensive and general applicable SCD representation based on Kimball's typology of SCD's [11]. In addition we propose a valid alternative to the snapshot-based information transfer. This solution is especially applicable in those cases, where the information requirements of the receiving data warehouse system is focused on complete and detailed historical information for all instances combined with the requirement of a minimal latency. For a given latency time interval the advantages of this method are evidently given for a dimension with a small number of changes compared to its cardinality.

The proposed method provides much more than solely a data replication. The primary target is not a physical mirror of the dimension object. All necessary views including the change history of this object are implemented in a standardized way. The event-fed cSCD (comprehensive SCD) approach has been designed according to the main goals of a modern data warehouse, which provides a single point of consistent data access.

2. Related works

Active data warehouses [2,21,15] have the tendency to provide complete data with minimal latency. The well-known limitations of processing dimensional snapshot data [19] 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 preservation of the complete history of the dimensional change events also builds an advantage compared with historical (periodic) snapshots.

In some cases daily snapshots [19] 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 not primarily depend on the change fluctuation and volatility of the instances. One can apply in a second step some compression algorithms to overcome these disadvantages.

Kimball [10] has introduced three slowly changing dimension (SCD) types to track changes in dimension attributes. In the SCD Type 1, the changed attribute is simply overwritten to reflect the most current value thus does not keep the historical changes. The 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 which is not an appropriate method for unpredictable changes.

Bliujute et al. [4] suggested the temporal star schema to overcome the SCD problems with event and state-oriented data. They tackled the SCD type 2 in fixed attributes with timestamp. We propose a more general event model where the target dimensional object can be fine tailored depending on business requirements.

Research in temporal databases [5] has identified two orthogonal dimensions of time in databases valid time, modeling changes in the real world and transaction time, modeling the update activity in the database [18]. In data warehouse, there is another step to refresh the warehouse data from the operational data sources. Therefore, the third time dimension – processing time – is used to model the activity of loading new data into the warehouse. 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 in [20].

The authors in [6,23] proposes a temporal multi-dimensional data model to cope with the changes of dimension via multi versions and valid time. Our purpose is not only to keep track the versions of instances but also their relationships within the dimensional data.

Inmon recommends the usage of normalized dimensions [8]. This is very important as the event-based maintenance of denormalized dimensions although possible is not very practical [3].

3. State-oriented data versus state-change or event-oriented data

In systems theory, the notation of *state* is introduced in order to separate the past from the future [1]. A *state* is something that has extent in time. Something is true about an object for a period of time, but was neither true before nor after. An *event* is instantaneous [5]; it is something that "happens", rather than being true over a period of time. Events delimit states. The occurrence of an event results in a fact becoming true; later, the occurrence of another event renders that fact no longer valid. Hence, events and states are duals: states can be represented by their delimiting events, and events are implied by states [17].

In a data warehouse, two types of data exist: (1) state-oriented data and (2) state-change or event-oriented data. Examples of state-oriented data include, e.g., address, prices, account balances, and inventory levels. Examples of event-oriented data are sales, inventory transfers, and financial transactions.

Since *every event is of significant* importance in the event-oriented data approach, the loss of a single event can lead to a loss of synchronization in the state between a source system and the data warehouse. Datasets containing event information must be queued and removed on reading to ensure that every event is processed just once and no event gets loss.

If the system processes *state information*, e.g., the current address, it is reasonable to integrate it with the old version of the state value stored. In such a system there is no need to guarantee that every dataset is processed just once. In addition, if a single state change is lost during data integration, the data warehouse is automatically resynchronized by processing the next state information.

Event-oriented data in a data warehouse uses an *event representation*, which means that each row in the fact table represents some event and has a timestamp, capturing the event occurrence time. For state-oriented data, we obviously make use of the *state representation*. Every row in the table describes some state and has two timestamps, i.e. the beginning and ending times of the period throughout which the state persisted (see Fig. 1).

Transaction Date	Transaction Amount
2005-05-05	100
2005-05-07	+500

Begin Date	End Date	Account Balance
2005-04-30	2005-05-05	1000
2005-05-05	2005-05-07	900
2005-05-07	*until changed*	1400

Fig. 1. Event representation vs. state representations of data.

4. Slowly changing dimension

In a data warehouse the information storage is typically divided into fact tables and dimension tables [10]. Fact table data, by its nature, represents a time series of measurements and is always augmented with an explicit time dimension. Analyzing old fact-table data is one of the standard queries in a data warehouse. Dimensional data requires a more specialised treatment. Dimensions do not change in a predictable manner, i.e. individual entities (e.g. customers, products) evolve slowly. Some of the changes are true physical changes (e.g. customer's new address). Other changes are actually corrections of mistakes in the data. One of the major contributors to the development of solutions in the area of changes in dimension attributes is Ralph Kimball. For these kinds of changes he introduced the notion of *slowly changing dimensions* (SCDs) [11]. The purpose of the SCD solution is to maintain the un-altered relationship between the facts and dimension table without updating the fact tables when the dimension data is changing. In the following sections, we will briefly introduce the SCD Types 1–3 as proposed by Kimball and discuss their limitations before suggesting an enhanced SCD Type 2 which we have used for our prototype implementation at T-Mobile Austria.

4.1. SCD Type 1

"Overwrite the old values in the dimension record with the new values"

This is the simplest and fastest SCD solution because it simply overrides the old dimensional value with the new information without keeping any trace. Obviously, it does not maintain any past history and we loose the ability to track the old history. The technique thus does not address the implications of evolving data, because a fact is only associated with the current value of a dimension column. Nevertheless, overwriting is frequently used when the data warehouse team legitimately decides that the old value of the changed dimension attribute is not of further interest.

Fig. 2 illustrates an example using SCD Type 1 to keep the information about customer Robert changing his address from 20 Rennweg to 25 Favoritenstr on 02-16-2005. As we can see, the old value of address (20 Rennweg) is overridden by the new one.

4.2. SCD Type 2

"Create an additional dimension record using a new value of the surrogate key"

This technique partitions the history between the old and the new value. A new surrogate key is created when a true physical change occurs in a dimension entity at a specific point in time, such as a customer's new address.

A fact is always associated with the value of a dimension column before it changes (via the surrogate key). A sequence of facts describes the evolving data. This technique automatically partitions history and therefore analytical applications are not required to place any time constraints on effective dates in the dimension. However, finding out the whole sequence of changes from the data repository (in order to do analysis across history) is difficult and involves rather complex and expensive queries.

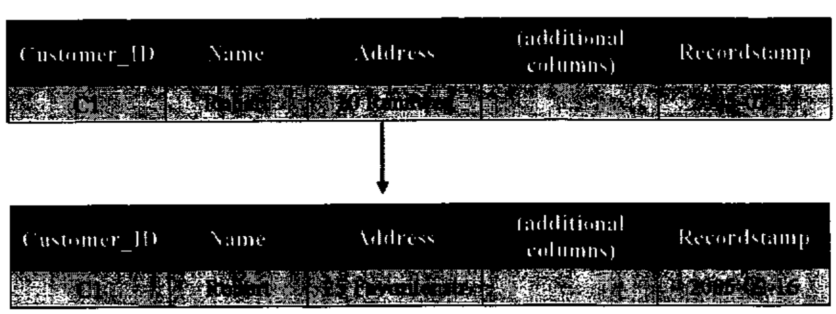


Fig. 2. Example using SCD Type 1.

Surrogate key	Customer_ID	Name	Address	(additional columns)	Recordstamp
C11	C1	Robert	20 Rennweg	...	2005-02-14

Surrogate key	Customer_ID	Name	Address	(additional columns)	Recordstamp
C11	C1	Robert	20 Rennweg		2005-02-14
C19	C1	Robert	25 Favoritenstr		2005-02-16

Fig. 3. Example using SCD Type 2.

The use of SCD Type 2 requires that the dimension key is generalized (*surrogate key*) to ensure the uniqueness property of the primary key of the dimension table when adding the new record with the same key with the previous record. It may be sufficient to take the underlying production key and add some version digits to the end of the key to simplify the surrogate key generation process. Future insertions to the fact table will be related to the new identifying codes, and so the segmentation will remain consistent with respect to time. Fig. 3 illustrates the example of using SCD Type 2 to keep trace the address information after the updating.

4.3. SCD Type 3

*"Create an **old** field in the dimension record to store the immediate previous attribute value."*

In an application which requires comparisons across transitions, SCD Type 3 is an appropriate solution. In an SCD Type 3, there will be two columns to indicate the particular attribute of interest, one indicating the original value, and one indicating the current value. There will also be a column that indicates when the current value becomes active. A fact is associated with both the original value and with the current value of a dimension column.

Compared to SCD Type 2, the SCD Type 3 does not increase the size of the table, since new information is updated while it still keeps part of history. However, SCD Type 3 is rarely used in actual practice because it modifies the structure of the dimension tables (adding more columns). Further more, SCD Type 3 is not be able to keep the entire history where an attribute is changed more than once because it keeps only the original and the current values of the changed attribute. Intermediate values are lost (see Fig. 4).

4.4. Limitations of the existing SCD approaches

Among the above three SCD solution approaches, SCD Type 2 is the most widely used in Data Warehousing projects [13]. While SCD Type 1 and Type 3 update the existing data and destroy the old information which implicate a conflict with the non-volatile Warehouse design criteria [8], SCD Type 2 inserts a new record without deleting or updating anything. However, SCD Type 2 partitions the history in a strict manner and does not allow the overlapping of valid times. Due to the absence of dates, it is impossible to determine precisely when

Customer ID	Name	Original Address	Current Address	Effective Date	(add columns)
C1	Robert	20 Rennweg	20 Rennweg	2005-02-14	

Customer ID	Name	Original Address	Current Address	Effective Date	(add columns)
C1	Robert	20 Rennweg	25 Favoritenstr	2005-02-16	

Fig. 4. Example using SCD Type 3.

changes occur. The only way to extract the time is via a join to the fact table. This will give an approximate time for the change. The degree of accuracy depends on the frequency of fact table entries relating to the dimensional entry concerned. The more frequent the entries in the fact table, the more accurate the traceability of the history of the dimension is, and vice versa.

The following three main issues give an explanation why the existing SCD approaches are not sufficient:

1. Comparing one dimension row with another to determine what has changed involves performing analysis across rows; something where SQL is notoriously bad [12].
2. Even when the next row in the dimension correctly tracks a change, it is impossible to examine the row in isolation to determine exactly what has changed, especially if more than one attribute has been changed.
3. If the data warehouse has to tie to the books, then it is not allowed to change e.g. an old monthly sales total, even if the old sales total was incorrect [11]. Late-arriving fact and dimension data cannot be integrated in such data warehouses using traditional SCD techniques.

4.5. *Comprehensive enhanced SCD solution (cSCD)*

As discussed in Sections 4.2 and 4.3, SCD Type 2 can keep traces of multiple changes of a dimensional attribute, while SCD Type 3 can keep the current value and previous value of a dimensional attribute. One can think about the possibility of mixing SCD Type 2 and SCD Type 3 to support increased analytical application complexity requirements. We propose an enhanced SCD which is a mix between SCD Type 2 and SCD Type 3. For each dimensional attribute change, as in case of SCD Type 2, we create a new record with the new surrogate key into a table, i.e. the TRANS table. The fact table records are linked to the relevant TRANS record via the surrogate key. However, for the attribute value chain tracing purpose, additional columns are added into the TRANS table. "Previous_Value" keeps the previous value of the attribute before it changes to current value, "Valid_from" and "Valid_to" determine the valid duration of the dimension attribute during the whole history of dimension entity. "Version" is the counter to keep the version of dimension attribute in time. "Last_version" is the redundant attribute which indicates the newest version (which means that the correspondent record stores the current attribute value and still be valid at the moment) (see also [22]). "Recordstamp" points out the processing time of the change of dimension attribute. "Change_key" is the attribute indicates which operation has effect the attribute (i.e., insert or delete or update). This column is used for checking the validation of the transaction which will be discussed later in this paper.

Please note that each record in the TRANS table corresponds to a change value of one dimension attribute in which we are interested in called *traced attribute*. In case we do not consider the change of dimension attributes, these attributes are classified as *flat attributes* and the TRANS table only keeps the current value (overridden as in SCD Type 1). For example, consider the case of *customer* dimension table which contains the following columns: ID, Name, and Address. ID is the dimension key, Name is the *flat attribute* (i.e., the change of name value is not of some interest and thus is not traced). Address is a *traced attribute*. Fig. 5 gives us an example how TRANS keeps the records when customer Robert changes his address from 20 Rennweg to 25 Favoritenstr. The Valid_to the very big date value: 31-12-9999 00:00:00 implies *until change*.

Surr Key	Cust ID	Valid from	Valid to	Name	Address	Address from	Recordstamp	Version	Last version	Change key
C11	C1	[illegible]	31-12-9999 00:00:00	Robert	20 Rennweg		[illegible]	1	Y	[illegible]

Surr Key	Cust ID	Valid from	Valid to	Name	Address	Address from	Recordstamp	Version	Last_version	Change key
C11	C1	10-02-2005 07:08:00	14-02-2005 07:09:59	Robert	20 Rennweg		14-02-2005 07:15:00	1	N	I
C12	C1	14-02-2005 07:10:00	31-12-9999 00:00:00	Robert	25 Favoritenstr	20 Rennweg	14-02-2005 07:15:00	2	Y	U

Fig. 5. Example using enhanced SCD (mix Type 2 and Type 3).

5. Data warehousing at T-mobile: a case study

The data warehouse at T-Mobile Austria runs on an Oracle 9.1.3 relational database and has a data volume of about six terabytes (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. One of the most important operational sources for the data warehouse is the *BSCS* system (*Business Support and Control System*). BSCS is the billing system and as such stores customer relevant data. However, the customer care system at T-Mobile Austria does not directly use BSCS functionality for performance reasons. It has an independent data repository and reads/updates relevant data by using BSCS interfaces.

5.1. *Snapshot-based SCD approach*

Since 2001, the data warehouse at T-Mobile Austria is built and refreshed using the batch approach. The dimensional data loaded from legacies like the billing system (BSCS), SAP or the CRM Systems is received via daily snapshots. Although the data is currently batch loaded, the data freshness requirement for transactional data is very high, e.g. CDRs (call detail records) are usually loaded every 4 h.

The Data Warehousing team at T-Mobile Austria has implemented a flexible *snapshot comparison* **approach. Fig. 6 gives an overview of the model. Applying the snapshot comparison technique to a particular** data warehouse dimension involves three tables:

5.2. *Table SNAP*

This table contains full snapshots of relevant datasets and tables of an operational source system. A snapshot describes datasets, which are already cleansed and transformed. However, a snapshot only contains current data, i.e. if a dataset/key is deleted within an operational source system, it is not part of the snapshot. A sequence of snapshots is distinguished by their snapshot date.

5.3. *Table TRANS*

This table contains the transactions, which were derived from two arbitrary (in general succeeding) snapshots taken from the table SNAP. The snapshot comparison approach distinguishes between flat and traced attributes.

- If a change affects only flat attributes, it does not cause the generation of a new version of the according dimension dataset in the data warehouse.
- If a change affects a traced attribute, the according transactions (insert, update, or delete) in the operational system are reconstructed by comparing the snapshots.

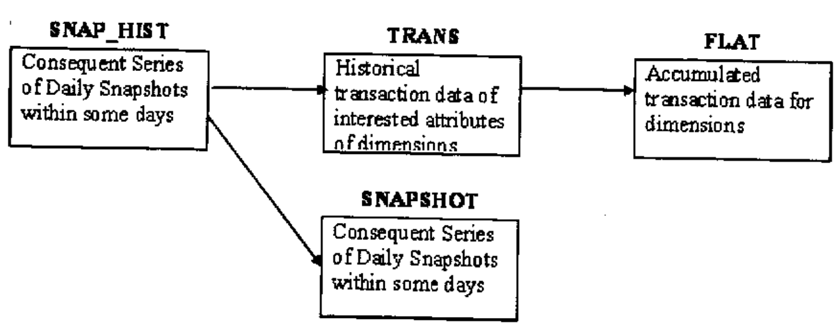


Fig. 6. Snapshot-based comparison SCD approach.

5.4. Table FLAT

This table contains the full history of the life-cycle of dimension datasets. If an update or delete transaction was identified by the snapshot comparison, a new version of the involved dimension dataset is inserted into the data warehouse using an enhanced SCD (mix Types 2 and 3) technique. The effective date of the changed attribute is derived from the snapshot date.

With this approach, a series of complete snapshots are stored chronologically to trace the complete history of dimensions without considering the number of actual data changes. This demand has been implemented via a PL/SQL package namely UTL_SCD. This functionality has often been reused within the data warehouses ETL processes (which are performed with the Informatica tool. at T-Mobile).

However, some issues remained unsolved. It is possible that more than one change on the same attribute occurs between two succeeding snapshots. If the data is collected by the snapshot comparison method, only those values will be captured, which exist at the time of the snapshot. All intermediate changes are completely missed in this case. Additionally, with the data warehouse side of six terabytes (TB), the tremendous daily snapshot versions are kept in the data warehouse for a specified time-period and significantly increase the data size while the amount of data changes is actually relatively small. For each snapshot comparison the number of records to process is high, requiring also high computing-resources and -time.

5.5. Event-based SCD approach

Due to the many problems of the snapshot-based approach, a more efficient near-real time approach is considered. Data changes in the operational sources are captured and provided in near real time as event messages via the event-based infrastructure of TIBCO [7]. This approach implies also the validation of the message content, which is necessary because of the highly differing quality provided by the message sources. Mainly three quality aspects, which are independent from the event-based infrastructure, 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 which is inspired by Kimball's three SCD types [11,6] as discussed in Section 4.5 and we propose a valid alternative to the snapshot-based information transfer. This solution is applicable especially in cases, where the information requirements of the receiving system is focused on complete and detailed historical information for all instances enhanced with a minimal latency demand. Especially in cases, where a dimension contains a large number of instances compared with the number of instance-changes within the time interval, which represents the time-latency requirements of the receiving system, the advantages of this method are obvious.

The proposed method is in fact a kind of logical information replication which can be successfully implemented with quite realistic and reasonable efforts. But the target of this logical replication is not to build a mirror of the dimensional physical object but to provide all necessary views on this objects to fulfil a much higher variety of demands, than the original source object can support. The event-fed 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".

Fig. 7 depicts the overall view of our event-based SCD process. Compared to the snapshot-based approach, the event-based SCD process does not require to keep all consequent snapshots data, instead it requires the event data which reflects all changing data since the last refresh cycle. The legacy systems thus do not need to send the full snapshot every night as they have to do at the moment. However, they need to provide the event data as frequently as required in the near real time refresh cycle.

The event-based SCD process will access the event data, filters those which happened since the last refresh (those records which do not appear in TRANS or those which are different with current records in TRANS) and then update the TRANS table.

The new event-based approach requires a very high level of truthfulness of the event. Therefore, the event-based SCD approach must provide as a necessary feature of event validation which validates the events with automatic correction options to override invalid events before applying these events to refresh the TRANS table. The Table Event_Protocol keeps the invalid events as an operational log.

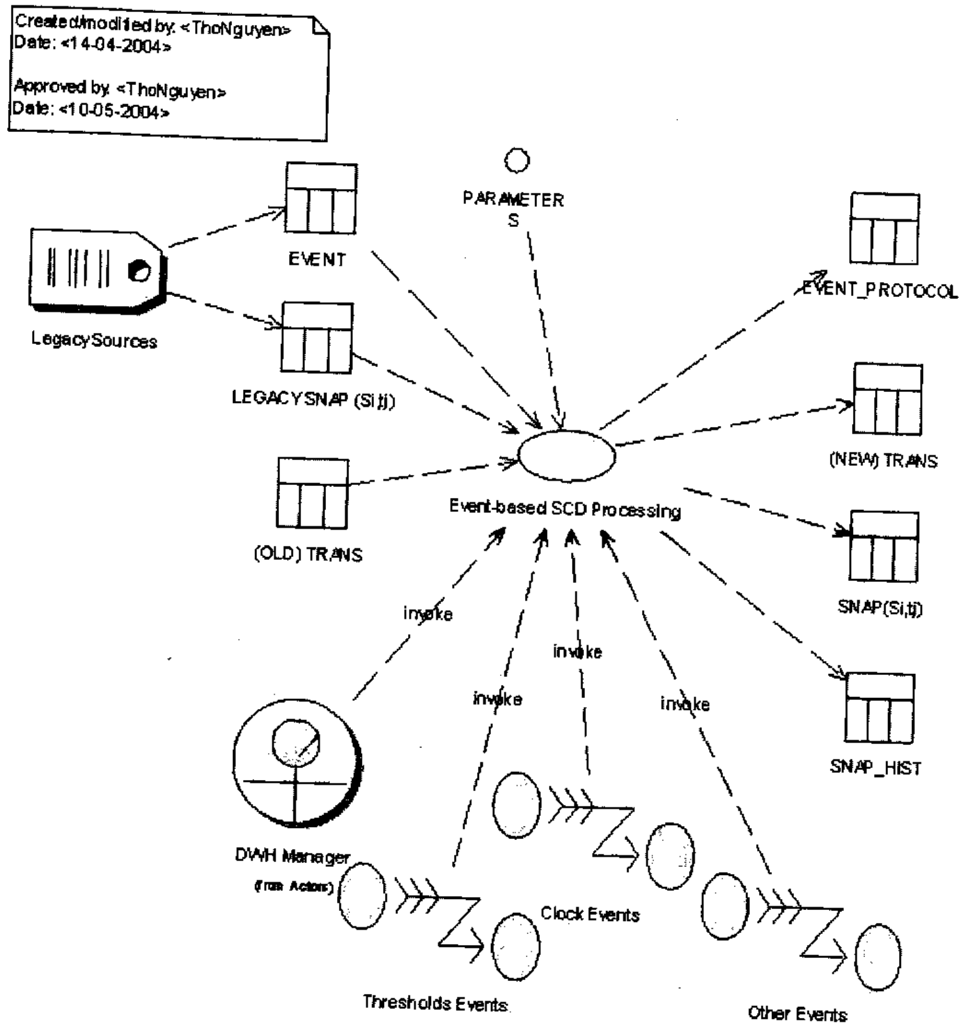


Fig. 7. Global view of event-based SCD process.

Since we do not keep all snapshots, in the case that there is requirement to have the snapshot at one point in time of any subset of entity instances, it must be made possible to rebuild such snapshot data. From the TRANS table, we can rebuild the snapshot of any subset instance at any point in time. The snapshot generation process can be totally based on TRANS or it could receive a truthful snapshot as the based snapshot to generate the on demand snapshot of any subset S_i(i indicates the entity instance) at any time point t_j (j denotes the point of time where the snapshot is built).

Such on demand Snapshot could be inconsistent with the Legacy Snapshot at some time- point, we thus have to check our TRANS to be consistent with the Legacy SNAP at these time points. When the inconsistency status is detected, the inconsistent data will be applied as the incoming events to re-establish the consistency. This process will also store the truthful Legacy Snapshot table in the periodic SNAP_HIST.

6. 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 a UML profile describing the event meta-model. Additionally, the notion of event is defined and shown by a simple example. Possible strategies of event interpretations are discussed.

Based on the defined model and the interpretation of the event we broader discuss and indicate that the traditional distinction between fact and dimension in an event-based DWH environment can be regarded as a different specialization of our event-based model.

6.1. *Profile*

To describe a general event [9] it is necessary to raise the model to the meta-level M2 [14] as the structure of each event type is very proprietary based on the transferred business information. The simplified profile definition is depicted in Fig. 8.

The key concepts of the event profile are as follows:

- Stereotype ⟨⟨Event⟩⟩ describes the object containing the event data.
- Stereotype ⟨⟨Efd⟩⟩ (abbreviation for event-fed 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. This stereotype attribute is use to determine the order of concurrent events (which have the same timestamp).
- Stereotype ⟨⟨Timestamp⟩⟩ identifies an attribute containing the timestamp information of an event. The transaction time (the timestamp when the transaction happened), event creation time (the timestamp when event is created), event processing time (the timestamp when the event is processed) 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.

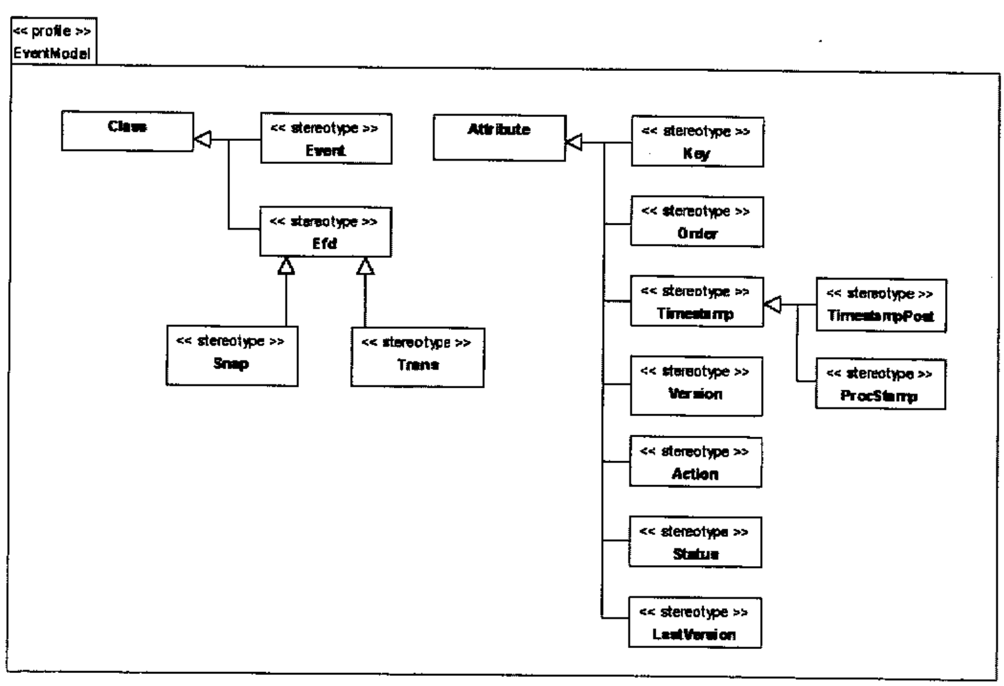


Fig. 8. Simplified profile event definition.

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 coincide, 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).

6.2. Example

To illustrate the usage of the event profile let us 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 its 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. 9 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 Section 5.3). The meaning of the additional attribute is discussed below.

6.3. 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 abbreviation for dimension snapshot.

The *Trans* object is maintained cumulatively, each event is added to their 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.

Another option is defined on the level of attribute; an attribute noted as *TimestampPost* is applicable for *Trans* object only. It is filled with the value of the corresponding *Timestamp* attribute of the successor version

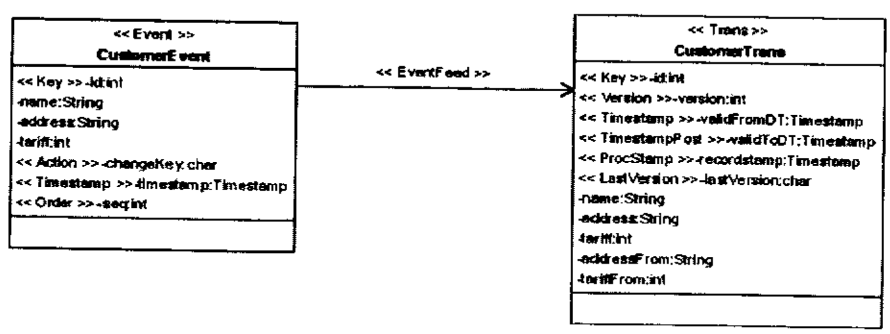


Fig. 9. Customer event profile example.

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 attributes is established with naming conventions.

A different semantic aspect is the validation of the event model, i.e. 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 checking in the 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 [16].

7. Event-fed cSCD implementation

The described implementation represents a particular instantiation of the presented event model in Section 6. The target object is implemented as a Trans table; natural key option is used; *Action* and *from attributes* are supported.

7.1. 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 easily call the functionality from ETL (e.g. Informatica Powercenter) mappings.

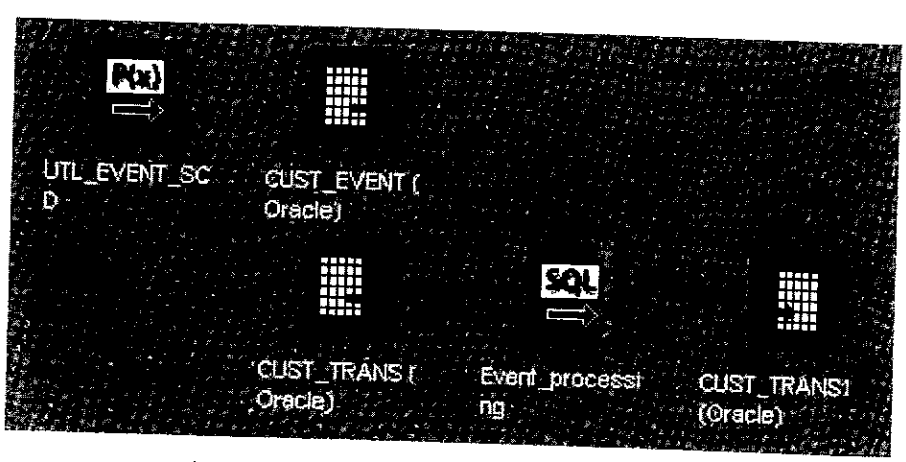


Fig. 10. Informatica mapping diagram of event processing.

The example mapping shown in Fig. 10 illustrates the source–target dependency (Note: the event transformation is done in UTL_EVENT_SCD). The source tables are the EVENT table (which contains only the new arriving events) and the old TRANS table, the target table is the new (updated) TRANS table (which keeps the full history of the dimension life-cycle). With this approach, the requirement of retrieving the state of each entity at any time point is still supported (generated on demand) without keeping a series of state-based snapshots.

7.2. The UTL_EVENT_SCD package

The package is 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*.

The package (Fig. 11) contains three 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.

7.2.1. Event processing (EP)

The event processing (EP) module is the main module of the package; it processes event data and refreshes the TRANS table as follows. It first accesses the event data, filters those which occurred 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. Table 1 enumerates the invalid cases and how to make the corrections or throw exceptions as invalid events.

Only the valid events are used to refresh the TRANS table. For each event data related to an entity instance, an equivalent transaction record in the TRANS table is created. If there are other events related

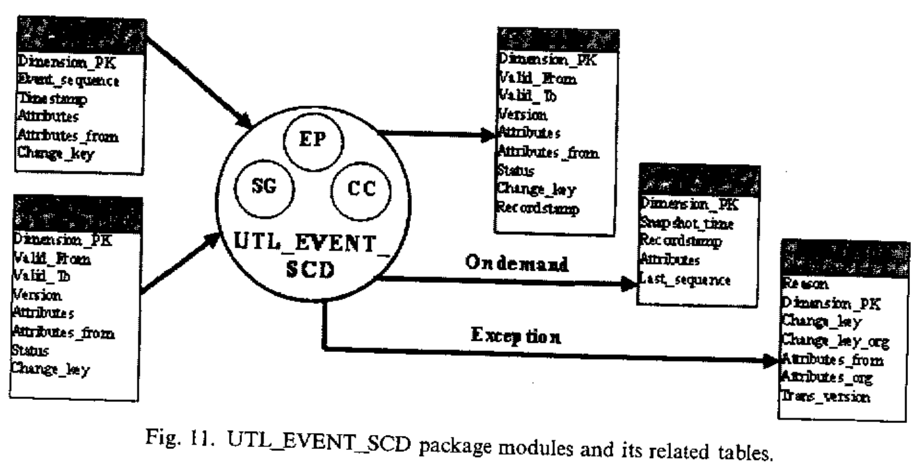


Fig. 11. UTL_EVENT_SCD package modules and its related tables.

Table 1
EFD SCD Package Corrections and Exceptions

Case	Description	Remark
Duplicated events	More than one events received with identical PK, timestamp and sequence number	Only one of those events is processed, all other are moved to protocol table (as there is no distinction, the processed event is randomly selected)
Duplicated time	More than one event have identical PK and time, but they are distinct in their sequence numbering	All events are processed in the order of sequence numbers. As the time of all events is identical, all but the last event are stored in the TRANS table as "semi invalid", i.e. validfrom > validto
Insert after insert	Two (or more) events with the same PK are inserted (change_key = 'I') subsequently with different timestamp	**Correction**: The first event is "normally" processed. The following ones are corrected to Update (i.e., change_key is set to 'U'). The change is written in the protocol table
Update non-existing PK	An update event is received without preceding insert event	**Correction**: The update event is interpreted as an insert event (i.e., change_key is set to 'I') The change is written in the protocol table
Delete non-existing PK	A delete event is received without preceding insert event	**Invalid**: The delete event is moved to protocol table
Change key NULL or invalid	The event interface contains change key. An event is received with invalid or NULL change key	**Invalid**: The event is moved to protocol table

to the same entity, the enhanced SCD (Section 4.5) is applied to keep trace over all transactions (with versions). The TRANS table thus contains the complete transaction history of dimension changes.

The invalid cases (i.e., those events in the protocol table) will be manually investigated. It is possible to discard those events which are really invalid ones (due to error in the system). If they are some missing events, they are re-applied as the new events (inserted as new record in EVENT table) to correct the inconsistent status of the DWHs.

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 a.m., 14/02/2005. At 7:10, Robert informs that he changes his address from 20 Rennweg to 25 Favoritenstr. 7:12 a.m., 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. 12)

The investigation of the performance behaviour 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 20,000 events), the more events accumulated, the less efficient of the event-SCD approach compared to the snapshot-based SCD approach (see Fig. 13).

With the current activities at T-Mobile which process daily about 15,000–20,000 events, the ideal refresh policy could be applied event-based SCD processing at every 3 h. The snapshot-based SCD could be applied as mid night if necessary.

7.2.2. 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 (Fig. 14) and (2) based on an existing snapshot, further referenced as based snapshot (Fig. 15). The generated Customer snapshots at 7:00 and 7:15 are shown in Fig. 16.

CUST_TRANS (Before Event applying)												
Surr_key	Cust_ID	Valid_from	Valid_to	Name	Address	Tariff	Address_from	Tariff_from	Record_stamp	version	Last_version	Change_key
C11	C1	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
C21	C2	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									
Surr_key	Cust_ID	Timestamp	Seq	Name	Address	Tariff	Address_from	Tariff_from	Change_key
C12	C1	14-02-2005 07:10:00	1	Robert	25 Favoriten	T1	20 Rennweg		U
C31	C3	14-02-2005 07:12:00	2	Micheal	10 Rathaus	T2			I
C13	C1	14-02-2005 07:13:00	3	Robert	25 Favoriten	T2		T1	U
C22	C2	14-02-2005 07:14:00	4	Sonja	15 Kargan	T1		T2	U

CUST_TRANS (After Event applying)												
Surr_key	Cust_ID	Valid_from	Valid_to	Name	Address	Tariff	Address_from	Tariff_from	Record_stamp	version	Last_version	Change_key
C11	C1	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
C12	C1	**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
C13	C1	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
C21	C2	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
C22	C2	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
C31	C3	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. 12. TRANS table refresh after UTL_EVENT_SCD package execution.

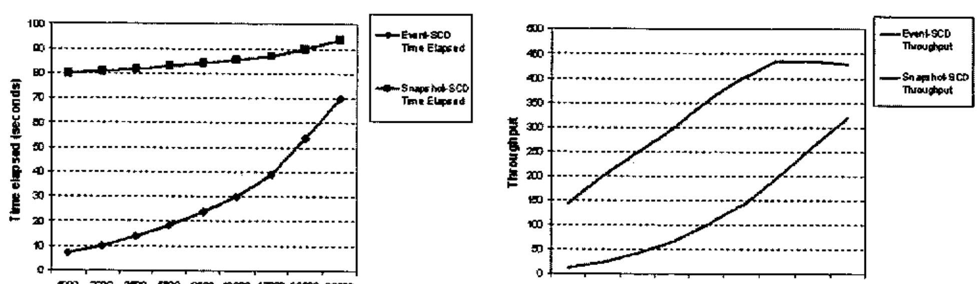


Fig. 13. Elapsed processing time and performance throughput (no. of transactions/s) comparison between event-SCD and snapshot-based SCD approach.

```
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. 14. Create snapshot from scratch (i_timepoint is the time point of the snapshot data).

7.2.3. *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

```
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. 15. 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						SNAPSHOT at 14-02-2005 7:15				
Cust_ID	Snaptime	Name	Address	Tariff		Cust_ID	Snaptime	Name	Address	Tariff
C1	14-02-2005 07:00:00	Robert	20 Rennweg	T1		C1	14-02-2005 07:15:00	Robert	25 Favoriten	T2
C2	14-02-2005 07:00:00	Sonja	15 Kargan	T2		C2	14-02-2005 07:15:00	Sonja	15 Kargan	T1
						C3	14-02-2005 07:15:00	Micheal	10 Rathaus	T2

Fig. 16. SNAPSHOT tables generated at 7:00 and 7:15.

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 7.2.2) 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.

8. Summary

In this paper, we introduced the event-fed comprehensive slowly changing dimension approach to overcome the limitation of existing SCD approaches and the snapshot-based solution. The event-fed cSCD approach significantly reduces the number of records to be processed compared to the snapshot-based approach. Besides, compared with the Kimball's classification of SCD [11] 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.

Furthermore by extending our model with summarizing stereotypes (e.g. add the actual value of the attribute to the previous value) the way is paved 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 investigations.

Acknowledgements

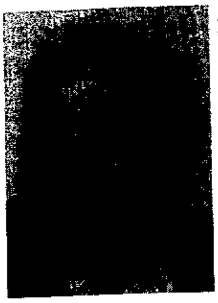
This research is performed at T-Mobile Austria and especially supported by the IT department providing the needed infrastructure and environment. This research is also supported by the Austrian Nationalbank

(Jubiläumsprojekt Nr 11806) "ZELESSA: Ein Instrument für intelligentes Monitoring von Geschäftsprozessen mit minimaler Verzögerung".

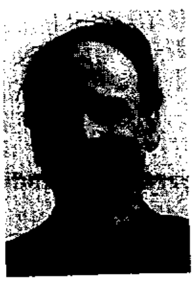
References

[1] H. Kopetz, G. Bauer, The time-triggered architecture, Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, vol. 91, Springer-Verlag LNCS 2114, 2003, pp. 112–126 (No. 1).
[2] S. Brobst, Enterprise application integration and active data warehousing, in: Proceedings of Data Warehousing 2002, Physica-Verlag, 2002, pp. 15–22.
[3] B. Rieger, K. Brodmann, Mastering time variances of dimension tables in the data warehouse, Osnabrueck University. http://andromeda.oec.uni-osnabrueck.de/uwdstift/DMOS-Res.pdf, 1999.
[4] R. Bliujute, S. Saltenis, G. Slivinskas, C.S.Jensen, Systematic change management in dimensional data warehousing, in: Proceedings of the Third International Baltic Workshop on Databases and Information Systems, Riga, Latvia, 1998, pp. 27–41.
[5] C.S. Jensen, C.E. Dyreson, The consensus glossary of temporal database concepts, in: Proceedings of Temporal Databases: Research and Practice, Springer LNCS 1399, 1998, pp. 367–405.
[6] C. Koncilia, J. Eder, Changes of dimension data in temporal data warehouses, in: Proceedings of Data Warehousing and Knowledge Discovery, DaWaK 2001, Springer-Verlag LNCS 2114, 2001, pp. 284–293.
[7] TIBCO Software Inc. Tibco. http://www.tibco.com, 2002.
[8] W.H. Inmon, Building the Data Warehouse, second ed., Wiley Inc, 1996.
[9] W. Rjaibi, K. Dittrich, D. Jaepel, Event matching in symmetric subscription systems, in: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, 2002, pp. 9–10.
[10] R. Kimball, The Data Warehouse Toolkit, Wiley Inc, 1996.
[11] R. Kimball, The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, second ed., Wiley Inc, 2002.
[12] R. Kimball, Realtime partitions, Intelligent Enterprise Magazine 5 (3) (2002).
[13] R. Kimball, Tricky time spans, Intelligent Enterprise Magazine 5 (10) (2002).
[14] MOF, Meta object facility (MOF) specification. http://www.omg.org/docs/formal/00-04-03.pdf, 2003.
[15] A. Karakasidis, P. Vassiliadis, E. Pitoura, Etl queues for active data warehousing, in: Proceedings of the Second International Workshop on Information Quality in Information Systems IQIS 2005, 2005, pp. 28–39.
[16] G. Hohpe, B. Woolf, K. Brown, C. D'Cruz, M. Fowler, S. Neville, M.J. Rettig, J. Simon, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Systems, Pearson Education, 2004.
[17] C.S. Jensen, R.T. Snodgrass, Semantics of time-varying information, Information Systems 21 (4) (1996) 311–352.
[18] D. Gao, S. Jensen, T. Snodgrass, D. Soo, Join operations in temporal databases, The VLDB Journal The International Journal on Very Large Data Bases 14 (1) (2006) 2–29.
[19] R. Rocha, F. Cardoso, M. Souza, Performance tests in data warehousing ETLM process for detection of changes in data origin, in: Proceedings of Data Warehousing and Knowledge Discovery, DaWaK 2003, LNCS 2737, pp. 129–139, 2003.
[20] R. Bruckner, A. Tjoa, Managing time consistency for active data warehouse environments, in: Proceedings of Data Warehousing and Knowledge Discovery, DaWaK 2001, Springer-Verlag LNCS 2114, 2001, pp. 254–263.
[21] J. Vandermay, Considerations for building a real-time oracle data warehouse, datamirror corporation white paper. http://datawarehouse.ittoolbox.com/white-papers/considerations-for- building- a-realtime-data-warehouse-1188, 2000.
[22] J. Eder, G. Kappel, A. Tjoa, R. Wagner, Bier – the behaviour integrated entity relationship approach, in: Proceedings of the Fifth International Conference on Entity-Relationship Approach, Dijon, North-Holland, 1996, pp. 147–166.
[23] B. Babel, J. Eder, C. Koncilia, T. Morzy, R. Wrembel, Creation and management of versions in multiversion data warehouse. in: Proceedings of the 2004 ACM symposium on Applied computingm, 2004, pp. 717–723.

Tho Manh Nguyen received his PhD in Information Systems from the Vienna University of Technology in 2005 and currently keeps a postdoctoral research fellowship at the Institute of Software Technology and Interactive Systems. He has been awarded Microsoft Student Travel Awards, IBM Europe Student Event Recognition, and Vienna University of Technology Outstanding Students Award. He is PC member and organizer of several international conferences and workshops such as DaWaK 2005, DaWaK 2006, ARES 2006, CONFENIS 2006, DAWAM 2006, iiWAS 2006. He has several publications in International Conferences and Journals in the field of Data Warehousing and Knowledge Discovery like IJDWM, IJBIDM, DaWaK, DOLAP, SCC. His research areas of interest include Data Warehousing, Data Mining and Knowledge Discovery, Business Intelligence System, Grid-based Knowledge Discovery, Service Oriented Computing, Ontology and Semantic Management.



A. Min Tjoa is since 1994 director of the Institute of Software Technology and Interactive Systems at the Vienna University of Technology. He is currently also the head of the Austrian Competence Center for Security Research. He received his PhD in Engineering from the University of Linz in 1979. He was Visiting Professor at the Universities of Zurich, Kyushu and Wroclaw (Poland) and at the Technical Universities of Prague and Lausanne (Switzerland). He was the president of the Austrian Computer Society from 1999 to 2003. He is member of the IFIP Technical Committee for Information Systems. His current research focus areas are Data Warehousing, Grid Computing, Semantic Web, Security, and Personal Information Management Systems. He has published more than 150 peer reviewed articles in journals and conferences. He is author and editor of 15 books.



Jaromir Nemec graduated in computer science from Charles University in Prague. He started his career at the Research Institute for Mathematical Machines Prague, working on the development of hierarchical database management system. Later Mr. Nemec worked with Siemens Austria developing an integrated operational system for insurance companies. Today Mr. Nemec is an independent IT architect focusing on system integration and data warehouse systems. He is currently working for a leading Austrian mobile communication provider. He is a speaker on the Oracle conferences.

Martin Windisch is since 2000 a senior architect and manager at the Department of Information Technology at T-Mobile Austria. He received his PhD in Physics from the Vienna University of Technology in 1995 and started his postdoctoral research at the Institute of Theoretical Physics in the field of Computational Physics, Condensed Matter Theory and Quasicrystals. He received a MSc in Medicine Physics at the University of Vienna in 1998. From 1996 to 2000 he designed and developed a laboratory information management system (LIMS) for about ten institutes of the Austrian Ministry of Agriculture with over 500 clients. His current working areas at T-Mobile Austria include Business Analysis, System Design, Enterprise Application Integration, Data Warehousing and Organization and Management of Information Systems with the focus on the business of mobile telecommunication.