Balancing Redundancy and Query Costs in Distributed Data Warehouses

Klaus-Dieter Schewe, Jane Zhao

Massey University, Information Science Research Centre, Department of Information Systems
Private Bag 11222, Palmerston North, New Zealand
Email: [k.d.schewe|j.zhao]@massey.ac.nz

Abstract

Abstract State Machines (ASMs) encourage high-level system specifications without forcing the development into the "formal methods straight-jacket". This makes them an ideal formal method for applications in areas, where otherwise only semi-formal methods are used. One such area is the development of data warehouse and on-line analytical processing (OLAP) applications to which this article contributes. Based on an ASM ground model for data warehouses we show which problems have to be solved in the case of distribution. This mainly amounts to making decisions on materialised views. In this article we develop simple refinement rules for this purpose. Then we develop a cost model that combines the costs of query processing with the maintenance costs arising from redundancy in the local data warehouse fragments. This cost model indicates, whether it is advantageous to apply a refinement rule or not. However, as the refinement process is non-deterministic, there is no guarantee that a global cost optimum will be reached.

Keywords. Abstract State Machine, Data Warehouse, Distribution, Cost Model, Refinement

1 Introduction

ASMs provide a strictly mathematically-founded method for high-level system design, validation and verification (Börger & Stärk 2003). The underlying idea is rather simple: modelling an application by states and transitions, where the states correspond to a structure in (first-order) logic. It has been shown that ASMs capture both sequential and parallel algorithms (Blass & Gurevich 2003, Gurevich 2000).

The advantage of using ASMs is that we obtain a mathematical model already at a very high level of abstraction, which permits starting the formal development process with a rather vague "ground model" of the system. All further development steps then turn out to be refinements. The ground model ASM allows systems requirements to be easily formalised and verified. On the other hand, the general ASM philosophy does not force us to apply any formal verification at such early stages. We could simply follow a philosophy of first designing and analysing and verifying later. This makes them an ideal formal method for applications in areas, where otherwise only semi-formal methods are used. In fact, ASMs have been succesfully applied to various applications including the specification and verification of Java and the Java virtual machine (Börger, Stärk & Schmid 2001), the specification of operational semantics of transactions (Prinz & Thalheim 2003), and the specification, validation and verification of database recovery (Gurevich, Sopokar & Wallace 1997).

In this article we continue our work from (Zhao & Ma 2004, Zhao & Schewe 2004), which addresses the development of data warehouse and on-line analytical processing (OLAP) applications with ASMs. Data warehouses are data-intensive systems that are used for analytical tasks in businesses such analysing sales/profits statistics, cost/benefit relations, customer preferences, etc. The term used for these tasks is "on-line analytical processing" (OLAP) in order to distinguish them from operational data-intensive systems, for which the term "on-line transaction processing" (OLTP) has become common.

The idea of a data warehouse (Inmon 1996, Kimball 1996) is to extract data from operational databases and to store them separately. The justification for this approach is that OLAP largely deals with condensed data, thus does not depend on the latest updates by transactions. Furthermore, OLAP requires only read-access to the data, so the separation of the data for OLAP from OLTP allows time-consuming transaction management to be dispensed with. Thus, the main idea of data warehouses implies a separation of input from operational databases and output to views that contain the data for particular OLAP tasks (also called "data marts").

Data warehouse design is to related to view integration (Kedad & Métais 1999, Widom 1995), but it is not exhausted by this. The work in (Agrawal, Gupta & Sarawagi 1997, Gyssens & Lakshmanan 1996, Thomson 2002) addresses multi-dimensional databases, i.e. schemata that are particular suitable for data warehouses. Maintaining the extraction of data from operational databases and storing them in data warehouses is addressed in (Engström, Chrakravarthy & Lings 2000). The work in (Lewerenz, Schewe & Thalheim 1999) presents a different view on data warehouse design emphasising not just the input, but also the output, i.e. the data marts and OLAP. In doing this, each data mart together with the OLAP functions working on it defines a so-called "dialogue object", a notion introduced in (Schewe & Schewe 2000) as a means for the integration of database systems and dialogue-based user interfaces.

As dialogue objects over a data warehouse lead to views over a view, it may be questioned, whether it makes sense to take a holistic approach to data warehouse design or whether it might be better to replace the data warehouse by a collection of materialised views on the operational databases. This view is also underlying the work in (Theodoratos 1999, Theodor-

Copyright ©2005, Australian Computer Society, Inc. This paper appeared at the Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), University of Newcastle, Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 43. Sven Hartmann and Markus Stumptner, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

atos & Sellis 1998).

The goal of this article is to present a more abstract framework for data warehouse design, in which a possible decision to realise a data warehouse by materialised views appears as a refinement. In the same way we would like to extend the view and consider distributed data warehouses. In order to do so we go back to the very basic idea underlying data warehouses, i.e. the separation of input from operational databases and output to dialogue interfaces for OLAP. We present an ASM ground model that realises this idea.

Then the decision on the distribution should also appear as refinements. Therefore, we discuss how fragmentation (Özsu & Valduriez 1999) of the warehouse schema can be formalised by refinements. It turns out that the case of data warehouses is much simpler than the general case for distributed databases. We present a cost model combining the costs for query processing (Ma & Schewe 2004*a*) with maintenance costs arising from redundancy in the local data warehouse fragments. This cost model indicates, whether it is advantageous to apply a refinement rule or not. However, as the refinement process is non-deterministic, there is no guarantee that a global cost optimum will be reached.

In Section 2 we will first present the general idea of the ASM method. Then we will develop the ground model ASM for data warehouse applications in Section 3. Finally, in Section 4 we discuss the problems for distributed data warehouses. We conclude with a brief summary.

2 Systems Development with Abstract State Machines

Abstract State Machines (ASMs, (Börger & Stärk 2003)) have been developed as means for high-level system design and analysis. The general idea is to provide a through-going uniform formalism with clear mathematical semantics without dropping into the pitfall of the "formal methods straight-jacket". That is, at all stages of system development we use the same formalism, the ASMs, which is flexible enough to capture requirements at a rather vague level and at the same time permits almost executable systems specifications. Thus, the ASM formalism is precise, concise, abstract and complete, yet simple and easy to handle, as only basic mathematics is used.

The systems development method itself just presumes to start with the definition of a *ground model ASM* (or several linked ASMs), while all further system development is done by refining the ASMs using quite a general notion of refinement. So basically the systems development process with ASMs is a refinement-validation-cycle. That is a given ASM is refined and the result is validated against the requirements. Validation may range from critical inspections to the usage of test cases and evaluation of executable ASMs as prototypes. This basic development process may be enriched by rigorous manual or mechanised formal verification techniques. However, the general philosophy is to design first and to postpone rigorous verification to a stage, when requirements have be almost consolidated. In the remainder of this article we will emphasise only the specification of ground model ASMs and suitable refinements (for details see (Börger & Stärk 2003)).

2.1 Simple ASMs

As explained so far, we expect to define for each stage of systems development a collection $M_1, \dots, M_n$ of ASMs. Each ASM M_i consists of a *header* and a *body*. The header of an ASM consists of its name, an import- and export-interface, and a signature. Thus, a basic ASM can be written in the form

ASM M
IMPORT $M_1(r_{11}, \dots, r_{1n_1}), \dots, M_k(r_{k1}, \dots, r_{kn_k})$
EXPORT $q_1, \dots, q_\ell$
SIGNATURE ...

Here r_{ij} are the names of functions and rules imported from the ASM M_i defined elsewhere. These functions and rules will be defined in the body of M_i — not in the body of M — and only used in M. This is only possible for those functions and rules that have explicitly been exported. So only the functions and rules $q_1, \dots, q_\ell$ can be imported and used by ASMs other than M. As in standard modular programming languages this mechanism of import- and export-interface permits ASMs to be developed rather independently from each other leaving the definition of particular functions and rules to "elsewhere".

The *signature* of an ASM is a finite list of function names $f_1, \dots, f_m$, each of which is associated with a non-negative integer ar_i, the *arity* of the function f_i. In ASMs each such function is interpreted as a total function $f_i : \mathcal{U}^{ar_i} \to \mathcal{U} \cup \{\bot\}$ with a not further specified set $\mathcal{U}$ called *super-universe* and a special symbol $\bot \notin \mathcal{U}$. As usual, f_i can be interpreted as a partial function $\mathcal{U}^{ar_i} \nrightarrow \mathcal{U}$ with domain $dom(f_i) = \{\vec{x} \in \mathcal{U}^{ar_i} \mid f_i(\vec{x}) \neq \bot\}$.

The functions defined for an ASM including the static and derived functions, define the set of *states* of the ASM.

In addition, functions can be *dynamic* or not. Only dynamic functions can be updated, either by and only by the ASM, in which case we get a *controlled* function, by the environment, in which case we get a *monitored* function, or by none of both, in which case we get a *derived* function. In particular, a dynamic function of arity 0 is a variable, whereas a static function of arity 0 is a constant.

2.2 States and Transitions

If f_i is a function of arity ar_i and we have $f(x_1, \dots, x_{ar_i}) = v$, we call the pair $\ell = (f, \vec{x})$ with $\vec{x} = (x_1, \dots, x_{ar_i})$ a *location* and v its *value*. Thus, each *state* of an ASM may be considered as a set of location/value pairs.

If the function is dynamic, the values of its locations may be updated. Thus, states can be updated, which can be done by an *update set*, i.e. a set Δ of pairs (ℓ, v), where ℓ is a location and v is a value. Of course, only *consistent* update sets can be taken into account, i.e. we must have

$$(\ell, v_1) \in \Delta \wedge (\ell, v_2) \in \Delta \Rightarrow v_1 = v_2.$$

Each consistent update set Δ defines *state transitions* in the obvious way. If we have $f(x_1, \dots, x_{ar_i}) = v$ in a given state s and $((f, (x_1, \dots, x_{ar_i})), v') \in \Delta$, then in the successor state s' we will get $f(x_1, \dots, x_{ar_i}) = v'$.

In ASMs consistent update sets can be obtained from *update rules*, which can be defined by the following language:

- the skip rule `skip` indicates no change;
- the update rule $f(t_1, \dots, t_n) := t$ with an n-ary function f and terms $t_1, \dots, t_n, t$ indicates that the value of the location determined by f and the terms $t_1, \dots, t_n$ will be updated to the value of term t;

- the sequence rule r_1 `seq` ... `seq` r_n indicates that the rules $r_1, \ldots, r_n$ will be executed sequentially;
- the block rule r_1 `par` ... `par` r_n indicates that the rules $r_1, \ldots, r_n$ will be executed in parallel;
- the conditional rule

  `if` φ_1 `then` r_1 `elsif` $\varphi_2 \ldots$ `then` r_n `endif`

  has the usual meaning that r_1 is executed, if φ_1 evaluates to true, otherwise r_2 is executed, if φ_2 evaluates to true, etc.;
- the let rule `let` $x = t$ `in` r means to assign to the variable x the value defined by the term t and to use this x in the rule r;
- the forall rule `forall` x `with` φ `do` r `enddo` indicates the parallel execution of r for all values of x satisfying φ;
- the choice rule `choose` x `with` φ `do` r `enddo` indicates the execution of r for one value of x satisfying φ;
- the call rule $r(t_1, \ldots, t_n)$ indicates the execution of rule r with parameters $t_1, \ldots, t_n$ (call by name).

Instead of `seq` we simply use ; and instead of `par` we write $\|$. The idea is that the rules of an ASM are evaluated in parallel. If the resulting update set is consistent, we obtain a state transition. Then a *run* of an ASM is a finite or infinite sequence of states $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ such that each s_{i+1} is the successor state of s_i with respect to the update set Δ_i that is defined by evaluating the rules of the ASM in state s_i.

We omit the formal details of the definition of update sets from these rules. These can be found in (Börger & Stärk 2003).

The definition of rules by expressions $r(x_1, \ldots, x_n) = r'$ makes up the body of an ASM. In addition, we assume to be given an *initial state* and that one of these rules is declared as the *main rule*. This rule must not have parameters.

2.3 Refinement of ASMs

The notion of *refinement* relates two ASMs M and M^*. In principle, as the semantics of ASMs is defined by its runs, we would need a correspondence between such runs, i.e.

- a correspondence between the states s of M and the states s^* of M^*, and
- a correspondence between the runs of M and M^* involving states s and s^*, respectively.

However, in contrast to many formal methods the notion of refinement in ASMs does not require all states to be taken into account. We only request to have a correspondence between "states of interest".

Formally, let S and S^* be the sets of states of ASMs M and M^*, respectively. A *correspondence of states* between M and M^* is a one-one binary relation $\equiv \subseteq S \times S^*$ such that $s_0 \equiv s_0^*$ holds for the initial states s_0 and s_0^* of M and M^*, respectively. In particular, we must have

$$s \equiv s_1^* \wedge s \equiv s_2^* \Rightarrow s_1^* = s_2^*$$

and

$$s_1 \equiv s^* \wedge s_2 \equiv s^* \Rightarrow s_1 = s_2.$$

Then we say that an ASM M^* is a *refinement* of the ASM M iff for each run $s_0^* \rightarrow s_1^* \rightarrow \ldots$ of M^* there is a run $s_0 \rightarrow s_1 \rightarrow \ldots$ of M and there are index sequences $0 = i_0 < i_1 < \ldots$ and $0 = j_0 < j_1 < \ldots$ such that $s_{i_x} \equiv s_{j_x}^*$ holds for all x.

3 An ASM Ground Model for Data Warehouses

As we view the basic idea of data warehousing is to separate the output from the input, we get our data warehouse architecture as shown in Figure 1, a three-tier model, consisting of operational database, the data warehouse, and the data mart, with OLAP functions.

At the bottom tier, we have the operational database model, which has the control of the updates to data warehouse. The updates are abstracted as data extraction from the operational database to maintain the freshness of the data warehouse. In the middle tier, we have the data warehouse which defines the data structure from a business point of view. It can either be modelled in multidimensional database(Thomson 2002), or in relational as in our case, as star or snowflake schemata (Kimball 1996). At the top tier, we have the data marts, which are constructed out of dialogue objects with OLAP operations. Based on this three-tier architecture, we end up with three linked ASMs, the DB-ASM, the DW-ASM, and the OLAP-ASM in our ground model.

3.1 The Operational Database ASM

Since we are only interested in the functions for the data warehouse, the operational database model is abstracted to have the data extraction functions for data warehouse. Basically the data warehouse presents a consolidated view of the operational database. The consolidation is achieved through aggregation of the transaction data, which will be realised by the data extraction rules defined in DB-ASM. As data sources other than relational can be wrapped as relational, for simplicity, we assume all data sources are relational, the signature in DB-ASM would just describe these relations. As relations can be seen as boolean-valued functions, each relation with n attributes in one of the operational databases will define an n-ary function in the signature of DB-ASM. The rules on DB-ASM correspond to the extraction of data for each of the relations in the data warehouse schema. DB-ASM will export these rules to refresh the data in the data warehouse.

Of course, it will be preferable to separate each operational database into a single ASM, but we leave this as a refinement step. However, in this case we may have to define an additional integrator ASM.

According to our abstraction, the ASM model for the operational database is defined as follows:

`ASM` DB-ASM
`EXPORT` $\text{extract}_1, \ldots, \text{extract}_n$
`SIGNATURE`
  $R_1(ar_1)$ (controlled),
  $\ldots,$
  $R_k(ar_k)$ (controlled)
`BODY`
  main =
    `if` selected (extract_i)
    `then` $\text{extract}_i(x)$
    `endif`
  $\text{extract}_i(x) = \ldots$

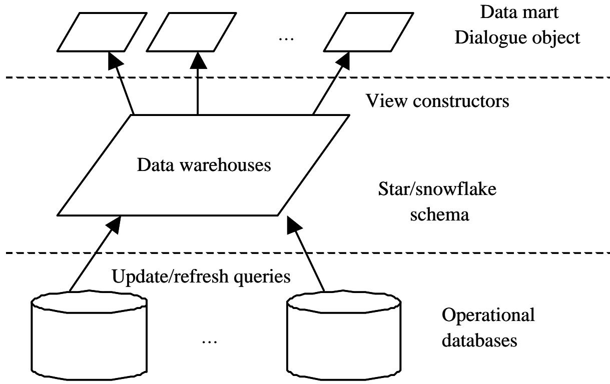

Figure 1: The general architecture of a data warehouse and OLAP

3.2 The Data Warehouse ASM

We follows the same abstraction as in the operational database model, the signature for DW-ASM will contain functions that correspond to the relation schemata used in the data warehouse schema. DW-ASM will import the extraction rules from DB-ASM to maintain the data warehouse data. On the other hand it will export view creation rules specifying view construction operations for use by OLAP-ASM. These rules will be defined in the same way as the rules for the extraction operations in DB-ASM.

Based on our modelling idea, the DW-ASM is defined as such:

ASM DW-ASM
IMPORT DB-ASM(extract$_1$, ..., extract$_n$)
EXPORT create_view$_1$, ..., create_view$_m$
SIGNATURE
  $S_1(ar_1)$ (controlled),
    ...,
  $S_k(ar_k)$ (controlled)
BODY
  main =
    if selected (refresh$_i$)
    then extract$_i(S_i)$
    elsif selected (view$_i$)
    then create_view$_i(x)$
    endif
  create_view$_i(x)$ = ...

Note that DW-ASM does not look significantly different from DB-ASM. The reason for this is that both ASMs specify simple relational databases and view creation operations on them.

If extract$_i(x)$ is the operation for extracting the data for the i-th relation schema in the data warehouse schema writing to variable x, then the operation has to be called with the i-th relation schema as the actual parameter. This guarantees that the desired data is extracted from the operational databases and inserted into the right location in the data warehouse.

Of course, we would like to have efficient refresh-operations. In particular, we would reduce the data warehouse updates to incremental changes. The specification of corresponding rule will be left as a refinement task.

3.3 The OLAP ASM

The top-level ASM dealing with OLAP is a bit more complicated, as it realises the idea of using dialogue

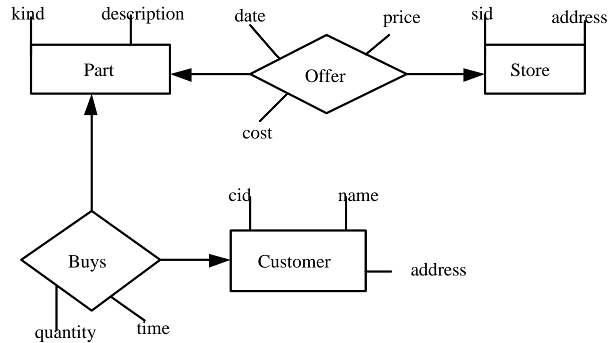

Figure 2: The operational database schema

objects for this purposes. The general idea from (Schewe & Schewe 2000) is that each user has a collection of open dialogue objects, i.e. data marts for our purposes here. At any time we may get new users, or the users may create new dialogue objects without closing the opened ones, or they may close some of the dialogues, or quit when they finish their work with the system. We abstract those functions as user, datamart management function as usr-dm-func. The major functionality, however, deals with running OLAP operations on data marts, OLAP-func, or creating data marts. In the latter case we have to use the view creation rules imported from DW-ASM. As data distribution will have no impacts on the other functions than the view creation in OLAP-ASM model, we will present a simplified version as follows:

ASM OLAP-ASM
IMPORT
  DW-ASM(create_view$_1$, ..., create_view$_m$)
SIGNATURE
  $V_1(ar_1)$, (controlled)
  ...,
  $V_m(ar_m)$, (controlled)
  ...
BODY
  main =
    forall usr, dm
    do if selected (user_dm_management)
      then usr-dm-func
      elsif selected (view$_i$)
      then create_view$_i(V_i)$
      elsif selected (OLAP_function)
      then OLAP-func
    enddo
  usr-dm-func = ...
  OLAP-func = ...

3.4 An Example of Grocery Store Data Warehouse Ground Model

The ground model given in the previous sections are very general. Let us give a more concrete model by look at an example taken from (Lewerenz et al. 1999, p.358). In this case we have a single operational database with five relation schemata as illustrated in the HERM diagram in Figure 2. If we assume a simple star schema for the data warehouse as illustrated by the HERM diagram in Figure 3, we will specify five simple refresh rules. Further we assume that we will create one user view, the total sales for the OLAP model, so we will define one view creation rule in the ASM-DW.

This leads to the following specification of the three linked models:

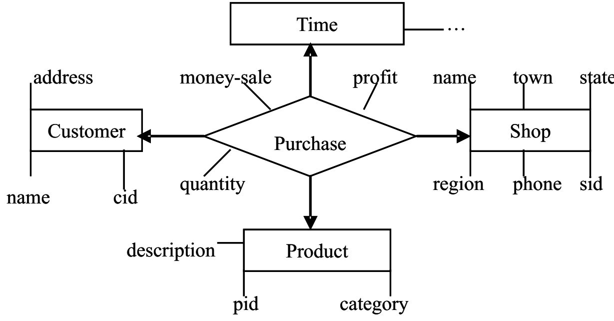

Figure 3: The data warehouse star schema

```
ASM DB-ASM
EXPORT extract_purchase, extract_customer,
extract_shop, extract_product, extract_time
SIGNATURE
  Store(2)(controlled),
  Part(3) (controlled),
  Customer(3) (controlled),
  Buys(4) (controlled),
  Offer(5) (controlled)
BODY
  main =
    if selected (purchase)
    then extract_purchase
    elsif selected (customer)
    then extract_customer
    elsif ...
    endif
  extract_purchase(x) =
    forall i, p, s, t, p′, c
    with ∃q.Buys(i, p, q, t) ≠ ⊥
      ∧∃n, a.Customer(i, n, a) ≠ ⊥
      ∧∃k, d.Part(p, k, d) ≠ ⊥
      ∧∃a′.Store(s, a′) ≠ ⊥
      ∧∃d.(Offer(p, s, p′, c, d) ≠ ⊥
      ∧date(t) = d)
     do let Q = sum(q | Buys(i, p, q, t) ≠
⊥),
        S = Q ∗ p′,
        P = Q ∗ (p′ − c)
      in x(i, p, s, t, Q, S, P) := 1
    enddo
  extract_customer = ...
```

```
ASM DW-ASM
IMPORT DB-ASM(extract_purchase, ...)
EXPORT create_view_sales, ...
SIGNATURE
  Shop(6) (controlled),
  Product(3) (controlled),
  Customer(3) (controlled),
  Time(4) (controlled),
  Purchase(7) (controlled)
BODY
  main =
    if selected (refresh_shop)
    then extract_shop
    elsif selected (refresh_product)
    then extract_product
    elsif ...
    elsif selected (view_sales)
    then create_view_sales(x)
    elsif ...
    endif
  create_view_sales(x) =
  forall s, r, st, m, q, y
  with ∃n, t, ph.Shop(s, n, t, r, st, ph) ≠ ⊥
      ∧∃ ....Time(..., m, q, y, ...) ≠ ⊥
    do let Q = sum(q′ | ∃c, p, t, s′, p′.
              Purchase(c, p, s, t, q′, s′, p′) ≠ ⊥
              ∧month(t) = m
              ∧quarter(t) = q
              ∧year(t) = y),
           S = sum(s′ | ∃c, p, t, q′, p′.
              Purchase(c, p, s, t, q′, s′, p′) ≠ ⊥
              ∧month(t) = m
              ∧quarter(t) = q
              ∧year(t) = y)
      in x(s, r, st, m, q, y, Q, S) := 1
```

```
ASM OLAP-ASM
IMPORT
  DW-ASM(create_view_sales, ...)
SIGNATURE
  view_sales(10), (controlled)
  ...
BODY
  main =
    forall usr, dm
    do if selected (user_dm_management)
      then usr-dm-func
      elsif selected (view_sales)
      then create_view_sales(view_sales)
      elsif selected (OLAP_function)
      then OLAP-func
    enddo
  usr-dm-func = ...
  OLAP-func = ...
```

4 Distribution Design as Refinement

As a data warehouse separates its input from its output, we only have read access to the data warehouse in creating the views, whereas updates are only maintained through data extraction from the operational databases. In particular, refreshing the content of the data warehouse is assumed to be performed off-line. This implies that we may set up a distributed data warehouse simply by replicating the whole data warehouse at each network location.

While this avoids data transport over the network, it is nevertheless not efficient. The warehouse may contain data that is never used at a particular location and thus need not be locally stored. In particular, the existence of superfluous data at a location reduces query performance. However, providing exactly the data that is needed at a location increases the maintenance effort, since data overlapping among the locations are most likely the case, though on the other hand it can exclude redundancy. Therefore, we investigate a cost model that combines query and maintenance costs to guide the distribution process.

4.1 Fragmentation

The standard approach to distribution design for databases is to fragment the logical units, i.e. the relations in case of a relational database (Özsu & Valduriez 1999). In fact, the way we modelled the data warehouse in Section 3 has a relational database schema at its core. We can therefore describe horizontal and vertical fragmentation as refinements of ASMs.

We may start with a fully replicated data warehouse, i.e. for each location ℓ we have a copy of DW-ASM as well as a tailored copy of OLAP-ASM, which only supports those data marts that are used at ℓ. Fragmenting the signature of DW-ASM impacts on the extraction rules defined on DB-ASM and the view creation rules on the copies of DW-ASM, whereas the local versions of OLAP-ASM remain unchanged.

Therefore, without loss of generality we may concentrate on a single machine DW-ASM as the object of refinement.

As shown in the previous section, the signature of DW-ASM consists of a set of controlled functions S_i ($i = 1, \dots, k$), each with an arity ar_i. Horizontal fragmentation of S_i with a selection formula φ leads to two new functions S_{i1} and S_{i2} with the same arity ar_i together with the constraints

$$S_{i1}(x_1, \dots, x_{ar_i}) \neq \bot \Leftrightarrow S_i(x_1, \dots, x_{ar_i}) \neq \bot \wedge \varphi(x_1, \dots, x_{ar_i})$$

and

$$S_{i2}(x_1, \dots, x_{ar_i}) \neq \bot \Leftrightarrow S_i(x_1, \dots, x_{ar_i}) \neq \bot \wedge \neg\varphi(x_1, \dots, x_{ar_i}).$$

In other words, the relation represented by S_i is the disjoint union of the relations represented by S_{i1} and S_{i2}. In DB-ASM we obtain the following additional rule:

```
extract_S_i1(x) =
    extract_S_i(y) ;
    forall x_1, ..., x_ar_i
    with y(x_1, ..., x_ar_i) = 1 ∧
            φ(x_1, ..., x_ar_i) = 1
    do x(x_1, ..., x_ar_i) := 1
    enddo
```

Analogously we obtain a rule extract_$S_{i2}(x)$. If fragment S_{ij} is used in DW-ASM, then this is reflected in an obvious change for refreshing.

Vertical fragmentation of S_i leads to n new functions $S_{i1}, \dots, S_{in}$ of arities $ar_{ij} < ar_i$ and subject to the constraints

$$S_{i1}(x_1, \dots, x_{ar_{ij}}) \neq \bot \Leftrightarrow \exists y_1, \dots, y_{ar_i}. S_i(y_1, \dots, y_{ar_i}) \wedge \bigwedge_{1 \le p \le ar_{ij}} x_p = y_{\sigma_j(p)}$$

for some injective $\sigma_j : \{1, \dots, ar_{ij}\} \to \{1, \dots, ar_i\}$ ($j = 1, \dots, n$). In addition, we must have $\bigcup_{j=1}^{n} \{\sigma_j(1), \dots, \sigma_j(ar_{ij})\} = \{1, \dots, ar_i\}$, and the places $\sigma_j(1), \dots, \sigma_j(ar_{ij})$ must define a key for S_i. Thus, in DB-ASM we obtain the following additional rules:

```
extract_S_ij(x) =
    extract_S_i(y) ;
    forall x_1, ..., x_ar_i
    with y(x_1, ..., x_ar_i) = 1
    do x(x_σj(1), ..., x_σj(ar_ij)) := 1
    enddo
```

4.2 Query and Maintenance Costs

Fragmentation implies changes to the view creation rules in DW-ASM. In fact, these rules define the queries we are interested in. In developing a query cost model as in (Ma & Schewe 2004*a*) we may assume that the queries are rewritten in a way that we first use a selection predicate φ to select tuples in a data warehouse relation S_i, then project them to some of the attributes. That is, each query involves subqueries of this form, which can be written as rules in DW-ASM in the following way

```
subquery(x) =
    forall x_1, ..., x_ar_i
    with S_i(x_1, ..., x_ar_i) = 1 ∧
            φ(x_1, ..., x_ar_i) = 1
    do x(x_σ(1), ..., x_σ(q)) := 1
    enddo
```

for some S_i of arity ar_i in the signature of DW-ASM and an injective $\sigma : \{1, \dots, q\} \to \{1, \dots, ar_i\}$. As the result of such a subquery corresponds to a fragment, first using horizontal fragmentation with φ, then vertical fragmentation with σ. It is common heuristic to consider that some of these fragments may need to be re-combined to reduce the cost of maintenance. The costs of a query q can be composed as $c_1(q) + d \cdot \sum_{f \in \mathcal{F}(q)} s(f)$, where the sum ranges over the set $\mathcal{F}(q)$ of fragments f used in q, d is a constant, $s(f)$ is the size of fragment f, and $c_1(q)$ is one of cost components, which is independent of the fragmentation (Ma & Schewe 2004*b*).

Thus, only $\sum_{f \in \mathcal{F}(q)} s(f)$ is influencing in the query costs of q in the fragmentation case. The query cost for the whole set of the queries at a site can be estimated by

$$\begin{aligned} qcosts &= \sum_q f_q \cdot \sum_{f \in \mathcal{F}(q)} s(f) \\ &= \sum_f \left(\sum_q n_{f,q} \cdot f_q \right) \cdot s(f) \\ &= \sum_f c(f) \cdot s(f) \ , \end{aligned}$$

where the first sum ranges over all queries, f_q denotes the frequency of query q, and $n_{f,q}$ denotes, the number of times the fragment f is used in computing the query q.

On the other hand, the maintenance costs correspond directly to the fragments built, i.e. we have

$$mcosts = d' \cdot \sum_f s(f)$$

with the sum ranging over all fragments used in the local version of DW-ASM and a constant d'. In combining query and maintenance costs we use a weighting factor, so the total costs can be rewritten in the form

$$\sum_f (c(f) + w) \cdot s(f)$$

with a constant w.

4.3 Re-combination of Fragments

Of course, producing all fragments that are suggested by the queries produce minimal query costs. However, as fragments may overlap, the more fragments we use the higher the maintenance costs will be. Therefore, we should also consider the re-combination of fragments into a single fragment for some of the fragments. Suppose S_{i1} and S_{i2} are fragments derived from the function S_i using selection formulae φ and ψ and projections defined by σ and τ, respectively. We may assume without loss of generality that we can write $\sigma(j) = i_j$ and $\tau(j) = i_{x+j}$. Then the combined fragment $S_{i1} \oplus S_{i2}$ can be obtained from S_i by using selection with $\varphi \vee \psi$ followed by projection using $\sigma + \tau$, which is defined by

$$(\sigma + \tau)(j) = \begin{cases} \sigma(j) & \text{for } j = 1, \dots, k \\ \tau(j - x) & \text{else} \end{cases}$$

assuming σ is defined on $\{1, \dots, k\}$ and τ on $\{1, \dots, \ell\}$.

Alternatively, we may re-combine S_{i1} and S_{i2} by an outer-join, which would produce a result different from $S_{i1} \oplus S_{i2}$, in which irrelevant values are replaced by $\bot$. This can be produced by adding to DB-ASM rules of the form

```
extract_S_{i1} ⊕ S_{i2}(x) =
    extract_S_{i1}(y) ;
    extract_S_{i2}(z) ;
    forall x_1, ..., x_{ar_{i1}}, y_1, ..., y_{ar_{i2}}
    with y(x_1, ..., x_{ar_{i1}}) = 1 ∨
                   z(y_1, ..., y_{ar_{i2}}) = 1
    do if ⋀_{1≤j≤m} x_{ar_{i1}-m+j} = y_j
       then
         x(x_1, ..., x_{ar_{i1}-m}, y_1, ..., y_{ar_{i2}}) :=
1
       else
         x(x_1, ..., x_{ar_{i1}}, ⊥, ..., ⊥) := 1 ;
         x(⊥, ..., ⊥, y_1, ..., y_{ar_{i2}}) := 1
       endif
    enddo
```

With respect to the total costs we have $c(f_1 \oplus f_2) = c(f_1) + c(f_2)$ in the worst case, and $\max(c(f_1), c(f_2))$ in the best case. Thus, we have to compare

$$costs = (c(f_1) + w) \cdot s(f_1) + (c(f_2) + w) \cdot s(f_2)$$

with

$$\begin{aligned} costs_{\text{new}} &= (c(f_1 \oplus f_2) + w) \cdot s(f_1 \oplus f_2) \\ &= (c(f_1) + c(f_2) + w) \cdot s(f_1 \oplus f_2) \\ or &= (\max(c(f_1), c(f_2)) + w) \cdot s(f_1 \oplus f_2) \end{aligned}$$

If we have $costs \geq costs_{\text{new}}$, it is advisable to recombine the fragments f_1 and f_2, otherwise prefer to use f_1 and f_2. Obviously, a fragment can be combined with more than one other fragment, but it should appear in only one such combination. Thus, a non-deterministic process which selects pairs of fragments, compares costs and eventually combines them cannot lead to a global cost optimum.

4.4 Example: Distributing a Grocery Store Data Warehouse

In the above section, we have introduced two types of fragmentation, horizontal and vertical, and as well as a cost model for determining if two fragments should be re-combined. Now let us use the example data warehouse, the grocery store, to explain how we apply the cost model in distribution design.

Scenario: Let us assume that there are three locations, A, B, and C, each of them has the analysis work as such: one looks after the sales of expensive products, one monitors the quantity of products sold, and the other tracks the profit made by the sales.

¿From the assumption, it is clear that each location needs only a part of the data warehouse. Tracking of expensive product sales renders it to horizontal fragmentation to the relation Purchase, and tracking the quantity and the profit to vertical fragmentation to the fragments resulted from the horizontal fragmentation.

In this example, we will start to look at the horizontal fragmentation first. Let the predicates φ be defined as S/Q of Purchase ≥ 1000 and ψ as S/Q of Purchase < 1000. Applying the horizontal fragmentation will give us two fragments f_1 and f_2. Then we shall apply vertical fragmentation to f_1 and f_2 by $\sigma_1 : \{1, \ldots, 5\} \rightarrow \{1, \ldots, 7\}$, which maps the attributes 1 to 1, up to 6 to 6, and $\sigma_2 : \{1, \ldots, 5\} \rightarrow \{1, \ldots, 7\}$, which maps 1 to 1, up to 4 to 4, 5 to 6, and 6 to 7. These give use fragments f_{11}, f_{12}, f_{21} and f_{22}.

According to the scenario, the fragments allocation will be made as such: location A will get f_{11} and f_{12}, location B will have f_{11} and f_{21}, and location C will take f_{12} and f_{22}.

To include the above fragmentation in our ground model, we will have a refined model with the following additional rules:

```
extract_f_11 =
    extract_f_1;
    forall c, s, p, t, Q, S, P
    with f_1(c, s, p, t, Q, S, P) = 1
    do x(c, s, p, t, Q, S) := 1
    enddo

extract_f_12 =
    extract_f_1;
    forall c, s, p, t, Q, S, P
    with f_1(c, s, p, t, Q, S, P) = 1
    do x(c, s, p, t, S, P) := 1
    enddo

extract_f_21 =
    extract_f_2;
    forall c, s, p, t, Q, S, P
    with f_2(c, s, p, t, Q, S, P) = 1
    do x(c, s, p, t, Q, S) := 1
    enddo
extract_f_22 =
    extract_f_2;
    forall c, s, p, t, Q, S, P
    with f_2(c, s, p, t, Q, S, P) = 1
    do x(c, s, p, t, S, P) := 1
    enddo
extract_f_1 =
    extract_purchase;
    forall c, s, p, t, Q, S, P
     with (c, s, p, t, Q, S, P) = 1 ∧ S/Q ≥
1000
    do x(c, s, p, t, Q, S, P) := 1
    enddo
extract_f_2 =
    extract_purchase;
    forall c, s, p, t, Q, S, P
     with (c, s, p, t, Q, S, P) = 1 ∧ S/Q <
1000
    do x(c, s, p, t, Q, S, P) := 1
    enddo
```

As we have discussed, for a better balanced query cost and redundancy, we should check if recombination of fragments is advisable. To apply the cost model introduced, we have to make some assumption again about the size of the fragments and the query frequency. Let us have the following:

- The average size of a tuple over Purchase as: $(64 \times 3 + 32 \times 3) = 672$(bits);
- The average number of tuples in Purchase as 400;
- The average size of a tuple over f_{11} is the same as over the other fragments, f_{12}, f_{21}, and f_{22}, as 640 bits;
- The average number of tuples in f_{11} is the same as in f_{12}, as 100;
- The average number of tuples in f_{21} is also the same as in f_{22}, as 300;
- The size of f_{11} and f_{12} as $64,000$, and f_{21} and f_{22} as $192,000$;
- The weighting factor w as 0.4;

- The frequency $c(f_{11})$ and $c(f_{12})$ in location A as 30;
- $c(f_{11})$ and $c(f_{21})$ in location B as 10;
- $c(f_{12})$ and $c(f_{22})$ in location C as 20.

So the total costs, one for the individual fragments case, the other for the recombination case, are calculated as such:

In location A,

$$\begin{aligned} costs &= (c(f_{11}) + w) \cdot s(f_{11}) + (c(f_{12}) + w) \cdot s(f_{12}) \\ &= (30 + 0.4) \cdot 64000 + (30 + 0.4) \cdot 64000 \\ &= 3891200 \end{aligned}$$

The new cost by recombining the fragments f_{11} and f_{12}:

$$\begin{aligned} costs_{\text{new}} &= (c(f_{11} \oplus f_{12}) + w) \cdot s(f_{11} \oplus f_{12}) \\ &= (c(f_{11}) + w) \cdot s(f_{11} \oplus f_{12}) \\ &= (30 + 0.4) \cdot 67200 \\ &= 2042880 \end{aligned}$$

In location B,

$$\begin{aligned} costs &= (c(f_{11}) + w) \cdot s(f_{11}) + (c(f_{21}) + w) \cdot s(f_{21}) \\ &= (10 + 0.4) \cdot 64000 + (10 + 0.4) \cdot 192000 \\ &= 2662400 \end{aligned}$$

The new cost by recombining the fragments f_{11} and f_{21}:

$$\begin{aligned} costs_{\text{new}} &= (c(f_{11} \oplus f_{21}) + w) \cdot s(f_{11} \oplus f_{21}) \\ &= (c(f_{21}) + w) \cdot s(f_{11} \oplus f_{21}) \\ &= (10 + 0.4) \cdot 256000 \\ &= 2662400 \end{aligned}$$

In location C,

$$\begin{aligned} costs &= (c(f_{12}) + w) \cdot s(f_{12}) + (c(f_{22}) + w) \cdot s(f_{22}) \\ &= (20 + 0.4) \cdot 64000 + (20 + 0.4) \cdot 192000 \\ &= 5324800 \end{aligned}$$

The new cost by recombining the fragments f_{12} and f_{22}:

$$\begin{aligned} costs_{\text{new}} &= (c(f_{12} \oplus f_{22}) + w) \cdot s(f_{12} \oplus f_{22}) \\ &= (c(f_{12}) + w) \cdot s(f_{12} \oplus f_{22}) \\ &= (20 + 0.4) \cdot 256000 \\ &= 5324800 \end{aligned}$$

By comparing the two costs, we have $costs \geq costs_{\text{new}}$ in location A. This indicates that recombination of fragments f_{11} and f_{12} will be more cost effective, while in location B and C, recombination makes no difference in cost concern.

5 Conclusion

In this article we continued our work on the application of ASMs to the design of distributed data warehouses and OLAP applications. We started from an ASM ground model that is based on the fundamental idea of separating input from operational databases from output to so-called data marts, which can be understood as views supporting particular analytical tasks. This ground model was already discussed partly in (Zhao & Ma 2004, Zhao & Schewe 2004).

In particular, we only have read access to the data warehouse in creating the views that underly the data marts, while updates are maintained through data extraction from the operational databases. Maintenance, however, is assumed to be performed off-line. This implies that we may set up a distributed data warehouse simply by replicating the whole data warehouse at each network location.

Such a full replication is not advisable, as the local data warehouses will contain lots of data that is never needed. Fully fragmenting the data warehouse and replicating each fragment at all locations where the data is needed provides the best performance for the OLAP application, but may use highly redundant data. In order to find the right balance between this redundancy and the query costs we discussed the recombination of fragments in light of a cost model that takes both query and maintenance costs into account. For each pair of fragments we can determine, whether it is advisable to re-combine them or not. However, as this depends on the choice of fragment pairs, a global cost optimum cannot be reached in general.

Both fragmentation and re-combination of fragments can be described as ASM refinements. Such refinements lead to an acceptable specification of a data warehouse application. Our overall goal is to provide a complete set of refinement rules for data warehouse and OLAP applications. The rationale is that the use of ASMs allows us to verify desirable properties of the application such as consistency, and these properties are preserved by a refinement-based approach. Furthermore, heuristics such as the cost analysis used in this article provide further guidance to whether a refinement rule should be applied or not. We will continue our research by investigating refinement rules for the OLAP functions.

References

Agrawal, R., Gupta, A. & Sarawagi, S. (1997), Modeling multidimensional database, *in* 'Proc. Data Engineering Conference, Birmingham', pp. 232–243.

Blass, A. & Gurevich, J. (2003), 'Abstract state machines capture parallel algorithms', *ACM Transactions on Computational Logic* **4**(4), 578–651.

Börger, E. & Stärk, R. (2003), *Abstract State Machines*, Springer-Verlag, Berlin Heidelberg New York.

Börger, E., Stärk, R. & Schmid, J. (2001), *Java and the Java Virtual Machine: Definition, Verification and Validation*, Springer-Verlag, Berlin Heidelberg New York.

Engström, H., Chrakravarthy, S. & Lings, B. (2000), A holistic approach to the evaluation of data warehouse maintenance policies, Technical Report HS-IDA-TR-00-001, University of Skövde, Sweden.

Gurevich, J. (2000), 'Sequential abstract state machines capture sequential algorithms', *ACM Transactions on Computational Logic* **1**(1), 77–111.

Gurevich, J., Sopokar, N. & Wallace, C. (1997), 'Formalizing database recovery', *Journal of Universal Computer Science* **3**(4), 320–340.

Gyssens, M. & Lakshmanan, L. (1996), A foundation for multidimensional databases, *in* 'Proc. 22nd VLDB Conference, Mumbai (Bombay), India'.

Inmon, W. (1996), *Building the Data Warehouse*, Wiley & Sons, New York.

Kedad, Z. & Métais, E. (1999), Dealing with semantic heterogeneity during data integration, *in* J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau & E. Métais, eds, 'Conceptual Modeling – ER'99', Vol. 1728 of *LNCS*, Springer-Verlag, pp. 325–339.

Kimball, R. (1996), *The Data Warehouse Toolkit*, John Wiley & Sons.

Lewerenz, J., Schewe, K.-D. & Thalheim, B. (1999), Modelling data warehouses and OLAP applications using dialogue objects, *in* J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau & E. Métais, eds, 'Conceptual Modeling – ER'99', Vol. 1728 of *LNCS*, Springer-Verlag, pp. 354–368.

Ma, H. & Schewe, K.-D. (2004*a*), A heuristic approach to horizontal fragmentation in object oriented databases, *in* 'Proceedings of the 2004 Baltic Conference on Databases and Information Systems', Riga, Latvia. to appear.

Ma, H. & Schewe, K.-D. (2004*b*), Query cost analysis for horizontally fragmented complex value databases. submitted for publication.

Özsu, T. & Valduriez, P. (1999), *Principles of Distributed Database Systems*, Prentice-Hall.

Prinz, A. & Thalheim, B. (2003), Operational semantics of transactions, *in* K.-D. Schewe & X. Zhou, eds, 'Database Technologies 2003: Fourteenth Australasian Database Conference', Vol. 17 of *Conferences in Research and Practice of Information Technology*, pp. 169–179.

Schewe, K.-D. & Schewe, B. (2000), 'Integrating database and dialogue design', *Knowledge and Information Systems* **2**(1), 1–32.

Theodoratos, D. (1999), Detecting redundancy in data warehouse evolution, *in* J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau & E. Métais, eds, 'Conceptual Modeling – ER'99', Vol. 1728 of *LNCS*, Springer-Verlag, pp. 340–353.

Theodoratos, D. & Sellis, T. (1998), Data warehouse schema and instance design, *in* 'Conceptual Modeling – ER'98', Vol. 1507 of *LNCS*, Springer-Verlag, pp. 363–376.

Thomson, E. (2002), *OLAP Solutions: Building Multidimensional Information Systems*, John Wiley & Sons, New York.

Widom, J. (1995), Research problems in data warehousing, *in* 'Proceedings of the 4th International Conference on Information and Knowledge Management', ACM.

Zhao, J. & Ma, H. (2004), Quality-assured design of on-line analytical processing systems using abstract state machines, *in* H.-D. Ehrich & K.-D. Schewe, eds, 'Proceedings of the Fourth International Conference on Quality Software (QSIC 2004)', IEEE Computer Society Press, Braunschweig, Germany. to appear.

Zhao, J. & Schewe, K.-D. (2004), Using abstract state machines for distributed data warehouse design, *in* S. Hartmann & J. Roddick, eds, 'Conceptual Modelling 2004 – First Asia-Pacific Conference on Conceptual Modelling', Vol. 31 of *CRPIT*, Australian Computer Society, Dunedin, New Zealand, pp. 49–58.