Using Abstract State Machines for Distributed Data Warehouse Design

Jane Zhao **Klaus-Dieter Schewe**

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

Abstract

Data Warehouses are data-intensive systems that are used for analytical tasks. As these tasks do not depend on the latest updates by transactions, data warehouses can be set up in a way that input of data from operational databases and output to dialogue interfaces for on-line analytical processes (OLAP) can be separated. In the paper we describe how abstract state machines (ASMs) can be used to design distributed data warehouses. We formalise the ground idea of data warehouses by a ground model ASM and discuss refinement steps, which can be applied in a step-by-step design methodology. Distribution will appear as such a refinement step.

1 Introduction

Data Warehouses are data-intensive systems that are used for analytical tasks in businesses such analysing sales/profits statistics, cost/benefit relation statistics, customer preferences statistics, 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. OLAP tasks are usually included in Management Information and Decision Support Systems.

The idea of a data warehouse (see (Inmon 1996) and (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, a first problem in data warehouse design is to integrate views from various source databases. This point of view of data warehouse design as a view integration problem has been strongly promoted in (Widom 1995) and (Kedad and Métais 1999). On the other hand it has been observed that the structure of the data needed for OLAP, i.e. the structure of data warehouse schemata, is somehow simpler than the structure of operational databases. This has led to the notion of multi-dimensional databases, in which "facts" needed for OLAP such as number of sales, prices, etc. are separated from "dimensions" such as time, location, product, etc., i.e. parameters that characterise the facts. Formally, we still obtain relations, in which the dimensions form a key, but the multi-dimensional (relational) database schemata usually have the form of star or snowflake schemata (Inmon 1996).

The work in (Gyssens and Lakshmanan 1996) presents a formal model for multi-dimensional databases. Various examples of star schemata for data warehouses can be found in (Kimball 1996). The work in (Agrawal et al. 1997) and (Thomson 1997) concentrates the conceptual modelling of multi-dimensional databases. In (Thalheim 2003) it has been observed that star and snowflake schemata dominate major components of database schemata. On this basis a design approach based on the integration of local schemata of this simple form is proposed and substantiated by some logical theory. A problem arising from the separation of data warehouses from operational databases is that the extraction functions have to be maintained. This problem is addressed in (Engström et al. 2000).

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. In the data warehouse literature these views are often called "data marts" (see e.g. (Inmon 1996)). The work in (Lewerenz et al. 1999) presented 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 and Schewe 1996, Schewe and Schewe 2000) as a means for the integration of database systems and dialogue-based user interfaces. Following this idea it is astonishing that a lot of work is put into the design of data warehouses, whereas the major emphasis should be on the OLAP functions that are based on views over the warehouse. This motivates to take a closer look into the systems dynamics, not just the static data structures. The work in (Schewe 1995) contains a sophisticated formal approach to specify, analyse and verify database applications including application programs. Such kind of work is even more needed for data warehouses and OLAP.

Furthermore, 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 and Sellis 1998) and (Theodoratos 1999).

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. Then the decision on the distribution should also appear as refinements. 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. Then we use Abstract State Machines (ASMs) to model this idea. ASMs (see (Börger and Stärk 2003)) provide a strictly mathematically founded method for high-level system design, validation and verification. We will show that it is already possible to use ASMs just to model the basic idea of data warehouses in a precise, simple, abstract and complete way. All further design steps then turn out to be refinements.

The advantage of using ASMs is that we obtain a somehow mathematical model of the data warehouse and OLAP from the very beginning. The ground model ASM allows systems requirements such as the use of dialogue objects for modelling the OLAP functions to be easily formalised as system invariants. Then we can verify that the specification is consistent with these invariants. 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 is called the absence of a formal methods straight-jacket in (Börger and Stärk 2003). Furthermore, we will also show that we can reason about architectural principles on the level of an ASM ground model.

We have chosen ASMs because of these features, and because they have been successfully applied to various applications including the specification and verification of Java and the Java virtual machine (Börger et al. 2001), the specification of operational semantics of transactions (Prinz and Thalheim 2003), and the specification, validation and verification of database recovery (Gurevich et al. 1997).

In this article we will first present the general idea of the ASM method in Section 2. Then we will develop a "ground model" ASM, which provides a very high-level abstract design of a data warehouse. In fact, it just reflects the basic idea of input-output separation. This idea leads not just to a single ASM, but to three inter-linked ASMs, one for the data warehouse as such, one for the operational database(s), and one for the dialogue objects and OLAP. We present this ground model in Section 3, and discuss advantages of the ASM approach on this level.

Following this we discuss refinements in Section 4. These refinements may comprise standard refinement steps for database and dialogue design as discussed in (Schewe and Thalheim 1994) and (Schewe and Schewe 1996), the approach to the design of star and snowflake schemata from (Thalheim 2003) and the formal refinement primitives for database and dialogue design from (Schewe 1995). In addition, we will discuss refinements that will lead to distributed data warehouses.

2 Systems Development with Abstract State Machines

Abstract State Machines (ASMs, (Börger and Stärk 2003)) have been developed as means for high-level system design and analysis. The general idea — which also underlies previous work of the second author in (Schewe 1995) and (Schewe and Thalheim 1994) — 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), which mainly captures requirements in a way similar to scenarios in (Schewe 2001) or use-cases in (Jacobson et al. 1992, Rumbaugh et al. 1999). All further system development is done by refining the ASMs using quite a general notion of refinement, which is the same as the one used in (Schewe and Thalheim 1994).

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 and Stärk 2003)).

2.1 Simple ASMs

As explained so far, we expect to define for each stage of systems development a collection $M_1, \ldots, M_n$ 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, ..., r_1n_1), ..., M_k(r_k1, ..., r_kn_k)
EXPORT q_1, ..., q_ℓ
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 $q_1, \ldots, q_\ell$ 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 $f_1, \ldots, f_m$, 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 $f(x_1, \ldots, x_{ar_i}) = v$, we call the pair $\ell = (f, \vec{x})$ with $\vec{x} = (x_1, \ldots, x_{ar_i})$ 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 $f(x_1, \ldots, x_{ar_i}) = v$ in a given state s and $((f, (x_1, \ldots, x_{ar_i})), v') \in \Delta$, then in the successor state s' we will get $f(x_1, \ldots, x_{ar_i}) = v'$.

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 $f(t_1, \ldots, t_n) := t$ with an n-ary function f and terms $t_1, \ldots, t_n, t$ indicates that the value of the location determined by f and the terms $t_1, \ldots, t_n$ 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).

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 and 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.

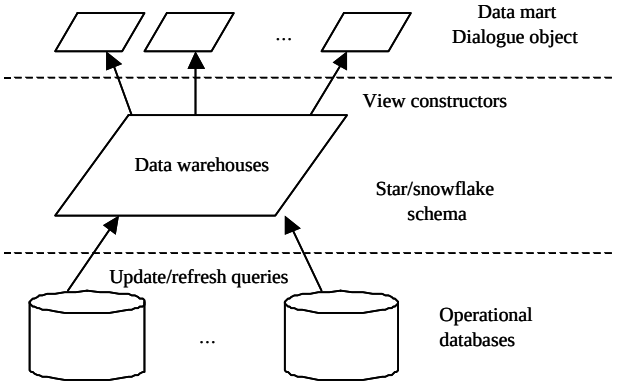

Figure 1: The general architecture of a data warehouse and OLAP

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

In order to define an ASM ground model for data warehouses let us go back to the basic idea of data warehouses and OLAP, which is basically a separation of input and output as explained in the introduction. The idea is illustrated by Figure 1.

In this figure we see a clear three-tier architecture. On the bottom tier we have operational databases set up for purposes that are of no particular interest for the data warehouse. However, we assume that the data stored in the data warehouse is extracted from these operational databases. In other words, the input of data into the data warehouse is defined by update- or refresh operations, which take data from these operational databases and insert them into the data warehouse.

In general, if there is some conflict between these operational data, i.e. data from different operational

databases has to be integrated before it can be inserted into the data warehouse, we need an integrator component (see (Widom 1995)). However, we may assume that this integrator is part of the extraction functions.

The second tier of the architecture in Figure 1 is made up by the data warehouse itself. In principle, we just have a database system here with the only differences that we may assume simpler schemata, i.e. star or snowflake schemata, and we do not have to consider complex transactions. The only write-operations are the refresh operations that connect the data warehouse to the operational databases. All other operations only read data from the data warehouse. In fact, we just build views for the "data marts" or dialogue objects that are used as the OLAP interface. So, the view construction operations are the only ones that link the middle tier to the top tier, which deals with OLAP.

The top tier itself is constructed out of the dialogue objects and the OLAP operations working on them.

3.1 The Idea of the ASM Ground Model

In our approach we take the general architecture from Figure 1 as a direct guideline for the definition of a ground ASM model, which will give us three linked ASMs DB-ASM, DW-ASM and OLAP-ASM.

Assuming (for simplicity) that the operational databases are all 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. If we assume a simple star schema, there will be only one such relation. These extraction rules can be combined into a single refresh rule. DB-ASM will export these refresh rules.

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.

Similarly, the signature for DW-ASM will contain functions that correspond to the relation schemata used in the data warehouse schema. If this is a star schema, we may have only one such relation schema. If it is a snowflake schema, we will have more than one controlled function. DW-ASM will have to import the refresh rules from DB-ASM. On the other hand it will export view rules specifying view construction operations for use by OLAP-ASM. These rules will be defined in the same way as the rules for the refresh operations in DB-ASM.

The signature of OLAP-ASM will be significantly more complicated, as we have to take into account that the data structure of dialogue object requires complex value objects, not just tuples. Furthermore, there will be significantly more operations for OLAP, and the flexibility of dialogue objects will require branching, parallelism and synchronisation. However, the imported rules are just those specifying the view creation operations of the data warehouse, i.e. the operations exported by DW-ASM.

3.2 The Operational Database ASM

According to our general idea the definition of DB-ASM will look as follows:

```
ASM DB-ASM
EXPORT extract_1, ..., extract_n
SIGNATURE
  R_1(ar_1) (controlled),
  ...,
  R_k(ar_k) (controlled)
BODY
  main =
    if selected (op_transaction_i)
    then run_transaction (op_transaction_i)
      seq if successful
          then commit (op_transaction_i)
          else abort (op_transaction_i)
          endif
    elsif selected (extract_i)
    then extract_i(x_i)
    endif
  extract_i(x_i) = ...
```

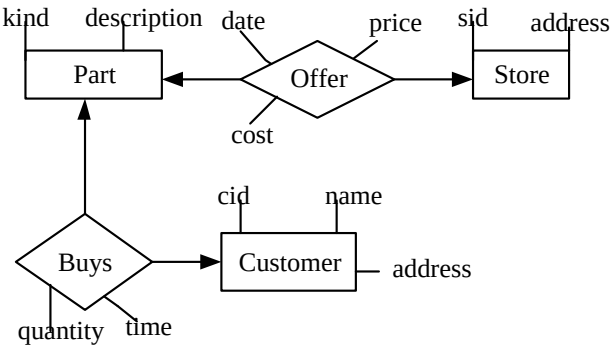

Figure 2: Grocery Store: Operative Sales Schema

Note that the specification of the main rule includes the execution of operational transactions. For our purposes we are not really interested in specifying these executions, as we may assume that at least some of the operational databases are already realised.

This means that the refinement of DB-ASM should lead to a specification that is equivalent to an already realised database machine. On the other hand, having a realisation of a machine at hand, we may use this to verify the correctness of the ASM specification.

Let us now 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 have to specify five simple refresh rules.

This leads to the following specification of DB-ASM, in which we omit the main rule, which has already been defined above.

```
ASM DB-ASM
EXPORT extract_purchase, extract_customer
  extract_shop, extract_product
SIGNATURE
  Store(2) (controlled),
  Part(3) (controlled),
  Customer(3) (controlled),
  Buys(4) (controlled),
  Offer(5) (controlled)
BODY
  main = ...
  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)
```

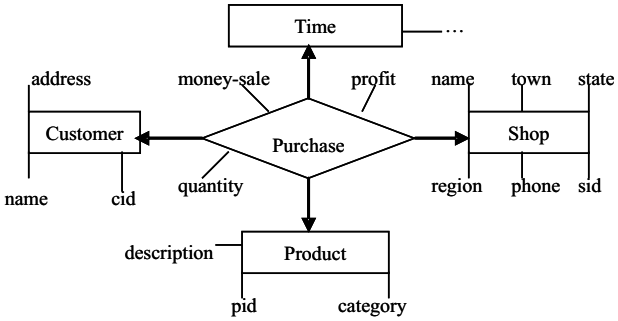

Figure 3: Grocery Store: Star Schema

```
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 = ...
```

Note that the rule extract_purchase realises a view. Using SQL the defining query could have been defined equivalently as

```
select C.cid, P.pid, S.sid, B.time,
  Sum(O.quantity) as quantity,
  Sum(O.quantity) * O.price as money_sales,
  Sum(O.quantity) * )(O.price - O.cost) as profit
from Customer C, Part P, Store S,
  Buys B, Offer O
where B.time.(day,month,year) = O.date
group by C.cid, P.pid, S.sid, B.time
```

3.3 The Data Warehouse ASM

According to our general idea the definition of DW-ASM will look as follows:

```
ASM DW-ASM
IMPORT DB-ASM(extract_1, ..., extract_n)
EXPORT create_view_1, ..., create_view_m
SIGNATURE
  R_1(ar_1) (controlled),
  ...,
  R_k(ar_k) (controlled)
BODY
  main =
    if selected (refresh_i)
    then extract_i(R_i)
    elsif selected (view_i)
    then create_view_i(x_i)
    endif
  create_view_i(x_i) = ...
```

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 $\text{extract}_i(x_i)$ is the operation for extracting the data for the i-th relation schema in the data warehouse schema writing to variable x_i, 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.

For instance, extract_purchase extracts all purchases of a certain product by a customer at a certain time together with the total quantity, price and profit. Executing the rule refresh_purchase(Purchase) has the effect that these data will be stored in the relation Purchase of the data warehouse.

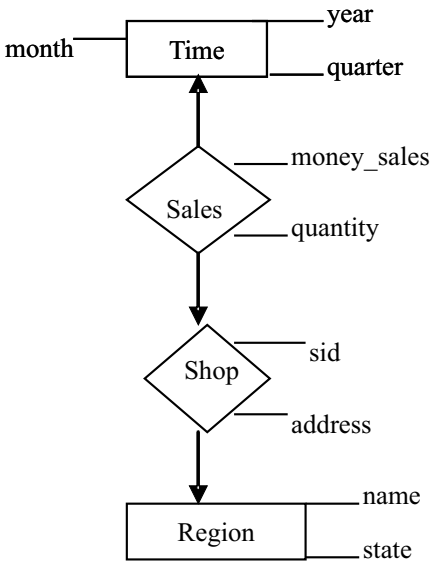

Figure 4: Schema underlying a data mart

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.

Let us now continue our grocery store example, in which case the controlled functions R_i correspond to the relation schemata arising from the star schema in Figure 3. We omit the main rule, which has already been specified.

```
ASM DW-ASM
IMPORT DB-ASM(extract_purchase, extract_shop
  extract_customer, extract_product)
EXPORT create_view_sales, ...
SIGNATURE
  Shop(6) (controlled),
  Product(3) (controlled),
  Customer(3) (controlled),
  Time(...) (controlled),
  Purchase(7) (controlled)
BODY
  main = ...
  create_view_sales(x) = ...
```

The rule create_view_sales creates a view with the underlying schema from Figure 4. Using SQL we can write the defining query as follows:

```
select S.sid, S.region, S.state,
    T.month, T.quarter, T.year,
    Sum(P.quantity) as quantity,
    Sum(P.money_sale) as money_sale
from Shop S, Time T, Purchase P
group by S.sid, T.month
```

Using the rule specification language of ASMs we obtain the following definition for create_view_sales:

```
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,s,p,t,q',s',p') ≠ ⊥
       ∧month(t) = m
       ∧quarter(t) = q
```

$\wedge$year$(t) = y$),
$S = sum(s' \mid \exists c, p, t, q', p'.$
Purchase$(c, s, p, t, q', s', p') \neq \bot$
$\wedge$month$(t) = m$
$\wedge$quarter$(t) = q$
$\wedge$year$(t) = y$)
in $x(s, r, st, m, q, y, Q, S) := 1$

3.4 The OLAP ASM

The top-level ASM dealing with OLAP is a bit more complicated, as it realises the idea of using dialogue objects for this purposes. The general idea from (Schewe and 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, and each user may create new dialogue objects without closing the existing ones. Thus, we maintain a controlled function user with user$(u) \neq \bot$ iff u is the id of a user in the system. Analogously, we use a controlled function datamart with datamart$(dm) \neq \bot$ iff dm is the id of a data mart in the system. In addition, we need another controlled function owner with owner$(dm) = u$ indicating that user u owns the data mart dm. Then part of the functionality of OLAP-ASM deals with adding and removing users and data marts. In particular, if a user leaves the system, all data marts owned by him must be removed as well.

The major functionality, however, deals with running operations on existing data marts or creating new data marts. In the latter case we have to use the view creation rules that have to be imported from DW-ASM. In this case we choose a new identifier for this data mart / dialogue object, and initialise a function representing its data content. Therefore, the signatures representing the output of the views will have to take the identifier of the data mart as a parameter, and they will have to consider a hidden part, from which all possible presentations can be derived, and a visible part that will be displayed.

If the user selects some of the data of the dialogue object and an operation other than quit, i.e. the user does not want to leave the system, or close, i.e. the user does not want to finish work on the current data mart, or open, i.e. no new data mart is to be created, then we request to receive additional input from the user, before the selected operation will be executed.

In order to control input from users, and data and operations selected by a user on a data mart we use three more monitored functions in, data_sel and op_sel.

Figure 5 illustrates this processing of the main rule of OLAP-ASM. Using the language from Section 2 we obtain the following specification:

```
   ASM OLAP-ASM
IMPORT
  DW-ASM(create_view_1, ..., create_view_m)
SIGNATURE
  V_1(ar_1), ..., V_m(ar_m), (controlled)
  user(1), (controlled)
  owner(1), (controlled)
  datamart(1), (controlled)
  in(1), (monitored)
  data_sel(2), (monitored)
  op_sel(2) (monitored)
BODY
  main =
    forall usr, dm
    with datamart(dm) ≠ ⊥
    do if user(usr) = ⊥
       then if in(usr) = new
          then user(usr) := 1
          endif
       elsif data_sel(usr, dm) = d
          ∧op_sel(usr, dm) = op
       then if op = quit
          then user(usr) := ⊥
             seq forall dm'
                with datamart(dm') ≠ ⊥
                   ∧owner(dm') = usr
                do datamart(dm') := ⊥
                enddo
          elsif op = close
          then datamart(dm) := ⊥
          elsif true
          then request_input(op, d_in)
             seq op(usr, dm, d, d_in)
          endif
       endif
    enddo
```

Of course, OLAP-ASM has to provide additional rules for all operations on data marts including their initialisation. Such an initialisation does not depend on the actual datamart dm nor does it request input data. Basically, it has to create a view on the data warehouse using one of the rules imported from DW-ASM. However, in order to support the OLAP operations on the data mart the view has to be duplicated resulting in a hidden and a visible part. Only the hidden part contains the full information. In addition, we have to create a new identifier for the data mart and associate ad owner to it.

Thus, a typical initialisation operation in OLAP-ASM will have the following form:

```
open_i(usr, dm, d, d_in) =
  choose dm'
  with datamart(dm') = ⊥
  do create_view_i(x_i)
    seq let ..., v_ij = f_j(x_i), v'_ij = v_ij, ...
       in V_i(dm', v_i1, ..., v_iki, v'_i1, ..., v'_iki) := 1
    seq datamart(dm') := 1
    seq owner(dm') := usr
  enddo
```

Now look again at our grocery store example. In this example we import the view creation rule create_view_sales, which can be used to initialise a data mart dealing with sales statistics. Obviously, there may be more than one user working on a sales statistics at the same time. Figure 6 shows different ways of presentation the data involved in such a data mart.

In fact, we consider just two dimensions: location and time. A location may be either a shop indicated by its identifier sid and (optionally) its region and state, or a region indicated by the region name and (optionally) its state, or a state. Analogously, for time we may have a month, a quarter or a year. The facts are quantity, i.e. the total number of sales, and money_sales, i.e. the sum of sales values. Therefore, we choose to represent the data mart by a 9-ary relation. That is, in the signature of OLAP-ASM we must have

$$V_{\text{sales}}(9) \text{ (controlled)}.$$

Thus, we obtain the following rule in OLAP-ASM for the initialisation of a data mart for the sales statistics:

```
open_sales(usr, dm, d, d_in) =
choose dm'
with datamart(dm') = ⊥
do create_view_sales(x)
  seq forall s, r, st, m, q, y, Q, S
    with x(s, r, st, m, q, y, Q, S) ≠ ⊥
    do let ℓ = (s, r, st), ℓ' = (s, r, st),
```

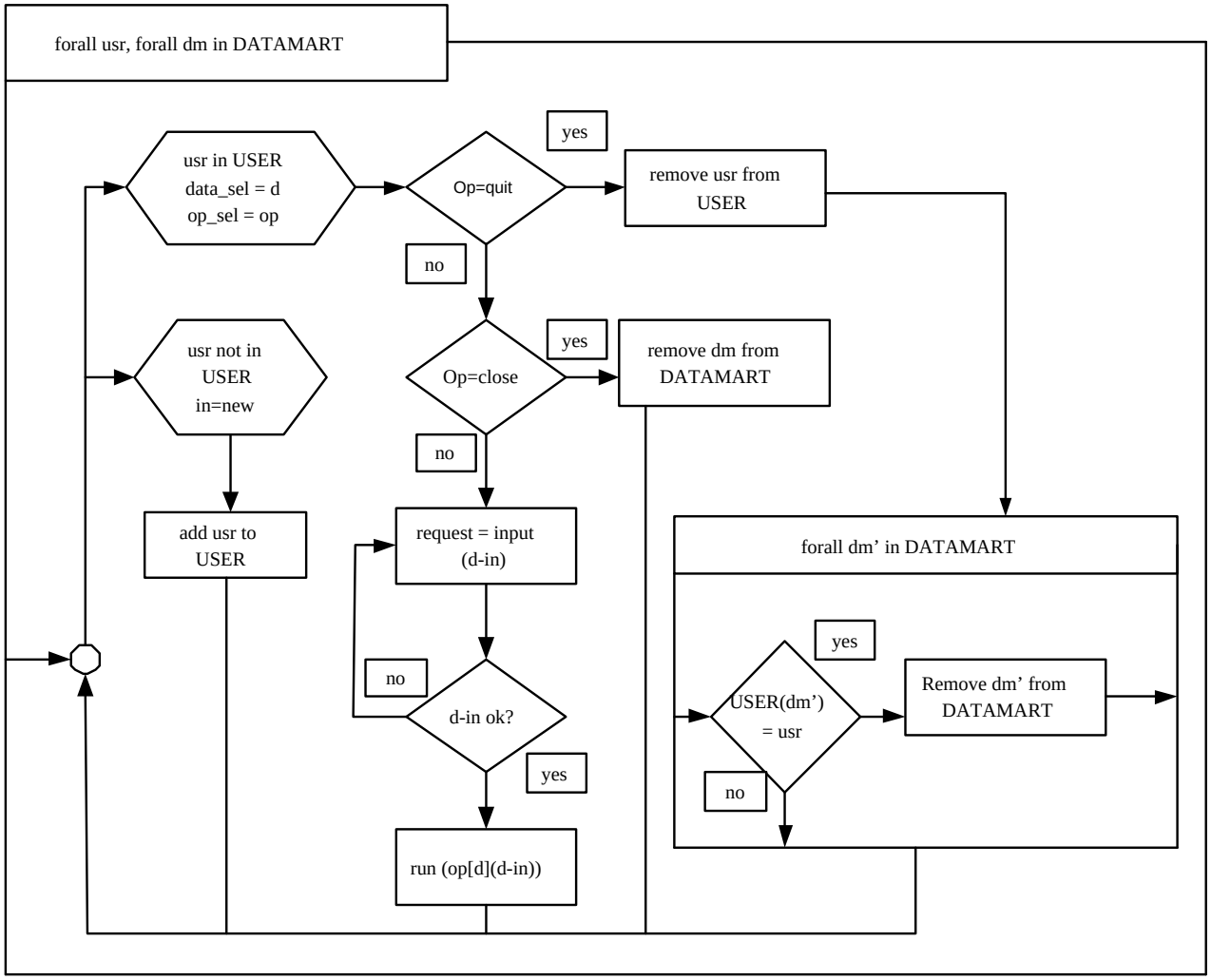

Figure 5: The main rule in OLAP-ASM

$t = (m, q, y)$, $t' = (m, q, y)$,
$Q' = Q$, $S' = S$
in $V_{\text{sales}}(dm', \ell, t, Q, S, \ell', t', Q', S') := 1$
enddo

Other typical OLAP operations include roll-up and drill-down. These operations rearrange the presented data by moving up and down the dimension hierarchies. For instance, in the sales statistics data mart of our grocery store example we have two dimensions location and time. Figure 6 illustrates the effects of roll-up and drill-down for the location dimension: shop $\leq$ region $\leq$ state.

The following rule in OLAP-ASM specifies the roll-up from shop to region:

roll-up$_{\text{shop}}(dm)$ =
forall r
with $\exists \ell_1, t_1, Q_1, S_1, \ell_2, t_2, Q_2, S_2.$
$V_{\text{sales}}(dm, \ell_1, t_1, Q_1, S_1, \ell_2, t_2, Q_2, S_2) \neq \bot$
$\wedge \exists s, st. \ell_1 = (s, r, st)$
do let $\ell_{\text{new}} = (r, st)$
$Q_{\text{new}} = sum(Q_1 \mid$
$\exists \ell_1, t_1, Q_1, S_1, \ell_2, t_2, Q_2, S_2, s, st.$
$V_{\text{sales}}(dm, \ell_1, t_1, Q_1, S_1, \ell_2, t_2, Q_2, S_2) \neq \bot$
$\wedge \ell_1 = (s, r, st))$,
$S_{\text{new}} = sum(S_1 \mid$
$\exists \ell_1, t_1, Q_1, S_1, \ell_2, t_2, Q_2, S_2, s, st.$
$V_{\text{sales}}(dm, \ell_1, t_1, Q_1, S_1, \ell_2, t_2, Q_2, S_2) \neq \bot$
$\wedge \ell_1 = (s, r, st))$
in $V_{\text{sales}}(dm, \ell_1, t_1, Q_1, S_1,$
$\ell_{\text{new}}, t_2, Q_{\text{new}}, S_{\text{new}}) := 1$
enddo

3.5 Reasoning about the ASM Ground Model

The ASM ground model developed so far is still rather vague in the sense that lots of details are missing. These will be added later via refinements. Furthermore, an advantage of ASMs discussed in (Börger and Stärk 2003) is that designers are not forced to verify their specification at immature stages. This leaves the designer the freedom to design first and to analyse and verify later.

However, the ASM ground level for data warehouses and OLAP that we have developed so far already allows us to start some first quality checks with respect to the satisfaction of requirements. As we adopted the dialogue object approach to data warehouses and OLAP, we will have the following general requirements:

1. At each time a user is either logged in, i.e. known to the system, or not. With respect to the functions we used in the main rule in OLAP-ASM, we can formalise this requirement by the invariant

$$\forall usr. \text{user}(usr) = \bot \vee \text{user}(usr) = 1 \ .$$

2. If a user is not logged in, the only available op-

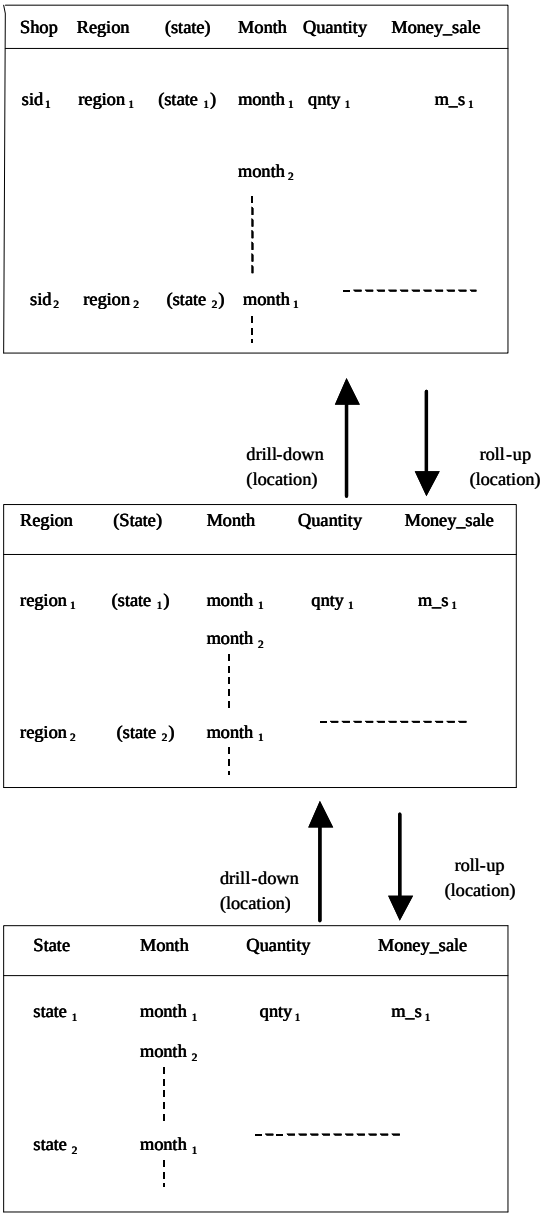

Figure 6: The effect of roll-up and drill-down operations on sales statistics

eration is 'new', which is formalised by

$$\forall usr.\mathrm{user}(usr) = \bot \wedge \mathrm{in}(usr) \neq \bot \Rightarrow \mathrm{in}(usr) = \mathrm{new}\ .$$

Without going into details of formal verification techniques — we are confident that the techniques used in (Schewe 1995) can be adapted to ASMs — it is easy to see that the main rule in OLAP-ASM is consistent with these two invariants.

Furthermore, we obtain requirements on datamarts that are realised by dialogue objects:

3. Each datamart always belongs to a user who must be logged in to the system, which is formally expressed by the invariant

$$\forall dm.\mathrm{datamart}(dm) \neq \bot \Rightarrow \exists usr.\mathrm{user}(usr) = 1 \wedge \mathrm{owner}(dm) = usr$$

For this invariant it is again easy to see the consistency of the main rule in OLAP-ASM.

4. Furthermore, we have requirements regarding the effect of operations such as 'quit' and 'close'. The former one logs off a user, the latter one deletes an open datamart. Moreover, if a user quits, all his datamarts will be deleted. We dispense formalising these requirements using transition constraints. Again, the main rule in OLAP-ASM is consistent with these requirements, too.

5. Finally, operations on a datamart depend on the data selected on the visual presentation of the dialogue object content, the selected operation and further input that has to be requested from the user. This is what the main rule in OLAP-ASM expresses. However, the major part concerning the execution of the selected operation is left for refinement.

Besides considering the consistency of the main rule in OLAP-ASM we have requirements regarding the data content of the datamarts, the data warehouse as such and the underlying operational databases. The chosen rules for building datamarts render their realisation as views over the data warehouse explicit. In fact, we have a one-to-one correspondence between view creation rules in DW-ASM and open-operations in OLAP-ASM. Similarly, there is a one-to-one correspondence between the extraction rules in DW-ASM and the extraction rules in DB-ASM.

However, if during refinement the database machine will be replaced by several heterogeneous machines the extraction rule in DW-ASM will become more complicated. In particular, it will be necessary to call more than one extraction rule in the different modules of DB-ASM.

4 Refinements

The general development method with ASMs is centered around the idea of stepwise refinement. Thus, the specification of the data warehouse ground model in the previous section is only the starting point for this process. Our goal is to provide a complete set of standard refinement rules for data warehouse and OLAP design similar to the refinement rules in (Schewe 1995) and the rules in (Schewe and Thalheim 1994).

Roughly speaking such refinements can be classified into three classes:

- The first class of refinement rules deals with extensions of a specification, i.e. further functions and rules are added in order to achieve completeness in the sense that all requirements will be captured. In the context of data warehouses and OLAP this mainly means to add functions to OLAP-ASM, which represent additional data marts, and to add rules to OLAP-ASM, which represent additional OLAP functionality. These refinements are almost straightforward.

- The second class of refinement rules deals with detailisation. That is, we will add details to the rules as such. This may involve defining new rules that are called in the existing rules. It may also lead to a less declarative structure of the specification.

- The third and probably most difficult class of refinements deals with restructuring. This includes the restructuring of controlled functions, the extension of modularity by "outsourcing" parts of an ASM into a new ASM, the distribution of the data warehouse, and the sequentialisation of rule

definitions. In the context of data warehouses and OLAP we may replace DW-ASM by several ASMs in order to achieve distribution. Similarly, we may provide a particular OLAP ASM for each user or separate ASMs for the different data marts.

For instance, in our grocery store example assume that analytical OLAP functions on the level of shops are only executed locally, whereas analysis of regions is done in regional head-quarters. In this case the star schema in Figure 3 would be replaced by a new star schema without Shop — and reduced arity for Purchase — reflecting a horizontal fragmentation (Özsu and Valduriez 1999) along the different values of sid.

The corresponding rules in corresponding OLAP ASMs would consequently be restricted to those with a fixed location dimension. For instance, the rule extract_purchase in DB-ASM would change to:

$$
\begin{array}{l}
\text{extract\_purchase}(s, x) = \\
\quad \texttt{forall } i,\ p,\ t,\ p',\ c \\
\quad \texttt{with } \exists q.\text{Buys}(i, p, q, t) \neq \bot \\
\qquad \wedge \exists n, a.\text{Customer}(i, n, a) \neq \bot \\
\qquad \wedge \exists k, d.\text{Part}(p, k, d) \neq \bot \\
\qquad \wedge \exists d.(\text{Offer}(p, s, p', c, d) \neq \bot \wedge \mathit{date}(t) = d) \\
\quad \texttt{do let } Q = \mathit{sum}(q \mid \text{Buys}(i, p, q, t) \neq \bot), \\
\qquad S = Q * p', \\
\qquad P = Q * (p' - c) \\
\qquad \texttt{in } x(i, p, t, Q, S, P) := 1 \\
\quad \texttt{enddo}
\end{array}
$$

Consequently, the rule create_view_sales would change to:

$$
\begin{array}{l}
\text{create\_view\_sales}(x) = \\
\quad \texttt{forall } m, q, y \\
\quad \texttt{with } \exists \ldots.\text{Time}(\ldots, m, q, y, \ldots) \neq \bot \\
\quad \texttt{do let } Q = \mathit{sum}(q' \mid \exists c, p, t, s', p'. \\
\qquad\qquad \text{Purchase}(c, p, t, q', s', p') \neq \bot \\
\qquad\qquad \wedge \text{month}(t) = m \\
\qquad\qquad \wedge \text{quarter}(t) = q \\
\qquad\qquad \wedge \text{year}(t) = y), \\
\qquad S = \mathit{sum}(s' \mid \exists c, p, t, q', p'. \\
\qquad\qquad \text{Purchase}(c, p, t, q', s', p') \neq \bot \\
\qquad\qquad \wedge \text{month}(t) = m \\
\qquad\qquad \wedge \text{quarter}(t) = q \\
\qquad\qquad \wedge \text{year}(t) = y) \\
\quad \texttt{in } x(m, q, y, Q, S) := 1
\end{array}
$$

Furthermore, we would obtain another fragment, in which Shop in Figure 3 would have to be replaced by Region with attributes region and state. Consequently, all view creation and OLAP rules in the corresponding DW-ASM and OLAP-ASM would be simplified by ignoring the Shop dimension. We omit the details here.

5 Conclusion

In this paper we presented first bits of our work on the use of Abstract State Machines (ASMs) for Distributed Data Warehouse Design. We showed that the general idea of data warehouses is in fact a separation of input from operational databases from output to dialogue-based on-line analytical processing (OLAP). This separation allows the basic model of a data warehouse to be specified by three interleaved high-level ASMs. Further development of the data warehouse and the OLAP interfaces can be based on step-wise refinement of such a "ground model" ASM.

The approach covers requirements at a very high level of abstraction, but nevertheless provides the rigorous mathematical semantics of ASMs from the very beginning of systems development. We demonstrated that already on the level of an ASM ground model it is possible to reason about requirements satisfaction, though the general ASM philosophy — first design, then analyse and verify — does not require formal verification at this stage. We argued that it is possible to verify some quality criteria already at the very beginning of system development.

Furthermore, there is no switch in terminology. This will ease the validation of requirements, and enable even formal verification when this becomes suitable. In this paper we could only sketch the idea of ASM refinement in data warehouse and OLAP design. For a full development method we will have to add a sophisticated collection of standard refinement rules (as in (Schewe 1995) for database application systems). Furthermore, we will explore the full range of OLAP functions that may be present in such systems.

Another problem left open for future work is adding ad-hoc creation of data marts to the specification. This would mean to include some form of linguistic reflection allowing textual input to be analysed and transferred into macros that realise further OLAP functionality. It may well be the case that this ambitious extension will require also extensions to the ASM method itself.

References

AGRAWAL, R., GUPTA, A., AND SARAWAGI, S. Modeling multidimensional database. In *Proc. Data Engineering Conference, Birmingham* (1997), pp. 232–243.

BÖRGER, E., AND STÄRK, R. *Abstract State Machines.* Springer-Verlag, Berlin Heidelberg New York, 2003.

BÖRGER, E., STÄRK, R., AND SCHMID, J. *Java and the Java Virtual Machine: Definition, Verification and Validation.* Springer-Verlag, Berlin Heidelberg New York, 2001.

ENGSTRÖM, H., CHRAKRAVARTHY, S., AND LINGS, B. A holistic approach to the evaluation of data warehouse maintenance policies. Tech. Rep. HS-IDA-TR-00-001, University of Skövde, Sweden, 2000.

GUREVICH, J., SOPOKAR, N., AND WALLACE, C. Formalizing database recovery. *Journal of Universal Computer Science 3*, 4 (1997), 320–340.

GYSSENS, M., AND LAKSHMANAN, L. A foundation for multidimensional databases. In *Proc. 22nd VLDB Conference, Mumbai (Bombay), India* (1996).

INMON, W. *Building the Data Warehouse.* Wiley & Sons, New York, 1996.

JACOBSON, I., CHRISTERSON, M., JONSSON, P., AND ÖVERGAARD, G. *Object-oriented Software Engineering: A Use-Case Driven Approach.* Addison-Wesley, 1992.

KEDAD, Z., AND MÉTAIS, E. Dealing with semantic heterogeneity during data integration. In *Conceptual Modeling – ER'99* (1999), J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, and E. Métais, Eds., vol. 1728 of *LNCS*, Springer-Verlag, pp. 325–339.

KIMBALL, R. *The Data Warehouse Toolkit.* John Wiley & Sons, 1996.

LEWERENZ, J., SCHEWE, K.-D., AND THALHEIM, B. Modelling data warehouses and OLAP applications using dialogue objects. In *Conceptual Modeling – ER'99* (1999), J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, and E. Métais, Eds., vol. 1728 of *LNCS*, Springer-Verlag, pp. 354–368.

ÖZSU, T., AND VALDURIEZ, P. *Principles of Distributed Database Systems.* Prentice-Hall, 1999.

PRINZ, A., AND THALHEIM, B. Operational semantics of transactions. In *Database Technologies 2003: Fourteenth Australasian Database Conference* (2003), K.-D. Schewe and X. Zhou, Eds., vol. 17 of *Conferences in Research and Practice of Information Technology*, pp. 169–179.

RUMBAUGH, J., JACOBSON, I., AND BOOCH, G. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.

SCHEWE, K.-D. Specification and development of correct relational database programs. Tech. rep., Clausthal Technical University, Germany, 1995.

SCHEWE, K.-D. UML: A modern dinosaur? – a critical analysis of the unified modelling language. In *Information Modelling and Knowledge Bases XII*, H. Jaakkola, H. Kangassalo, and E. Kawaguchi, Eds., Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, 2001, pp. 185–202.

SCHEWE, K.-D., AND SCHEWE, B. View centered conceptual modelling – an object oriented approach. In *Conceptual Modeling – ER '96* (1996), B. Thalheim, Ed., LNCS 1157, Springer, pp. 357–371.

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

SCHEWE, K.-D., AND THALHEIM, B. Principles of object oriented database design. In *Information Modelling and Knowledge Bases V*, H. Jaakkola, H. Kangassalo, T. Kitahashi, and A. Márkus, Eds., Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, 1994, pp. 227–242.

THALHEIM, B. Database component ware. In *Database Technologies 2003: Fourteenth Australasian Database Conference* (2003), K.-D. Schewe and X. Zhou, Eds., vol. 17 of *Conferences in Research and Practice of Information Technology*, pp. 13–26.

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

THEODORATOS, D., AND SELLIS, T. Data warehouse schema and instance design. In *Conceptual Modeling – ER'98* (1998), vol. 1507 of *LNCS*, Springer-Verlag, pp. 363–376.

THOMSON, E. *OLAP Solutions: Building Multidimensional Information Systems.* John Wiley & Sons, New York, 1997.

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