ASM Ground Model and Refinement for Data Warehouses

Klaus-Dieter Schewe, Jane Zhao

Massey University, Department of Information Systems &
Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand
[k.d.schewe|j.zhao]@massey.ac.nz

Abstract. Data Warehouses and on-line analytical processing (OLAP) systems are a promising area for the application of Abstract State Machines (ASMs). In this paper a ground model specification for data warehouses is sketched that is based on the fundamental idea of separating input from operational databases and output to OLAP systems. On this basis we start defining formal refinement rules for such systems. These refinement rules enable a formal method for the design of data warehouses and OLAP systems that can be applied without knowing details of the ASM formalism.

1 Introduction

Data Warehouses are data-intensive systems that are used for "on-line analytical processing" (OLAP), e.g. analysing sales/profits statistics, cost/benefit relation statistics, customer preferences statistics, etc. Thus, whenever we are confronted with data warehouse design, this includes the design of an OLAP system.

Usually the design of data warehouses and OLAP systems uses rather informal methods, e.g. UML-based approaches [4, 11]. Such approaches have been criticised in general in [7] and [2] due to their lack of solid foundations and semantics and the inappropriateness of layers of abstraction. Our work aims at showing that the use of the ASM method [2, 1] is better suited for the problem.

Applying the ASM method basically consists of developing a "ground model" that is then subject to step-by-step refinement. Adopting a general idea from [6] our goal is to develop application-specific refinement rules that can be applied without knowing details of the ASM formalism, and to glue them together with pragmatic guidelines for applying the refinement rules.

In [14, 13, 9] we started developing an ASM ground model for data warehouses and OLAP system based on the fundamental idea of separating input from operational databases from output to OLAP systems. According to this idea we obtain a model of three interconnected ASMs, one for the operational database(s), one for the data warehouse, and one for the OLAP system.

In this paper we start developing refinement rules for the development of distributed data warehouses and OLAP systems. We roughly classify rules into

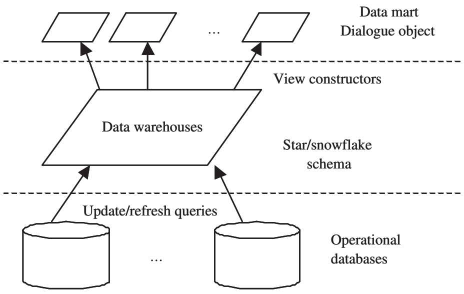

Fig. 1. The general architecture of a data warehouse and OLAP

two groups: those that reflect additional application needs and those that reorganise and improve a specification without adding new requirements. For the first group we have to pick up techniques for view integration, reorganisation of data structures according to a formal data warehouse model [3], and constraint manipulation techniques. For the second group techniques for introducing materialised views [10], distribution design [9] and other more standard operational refinement rules [1] have to be considered. We concentrate only on the first group of rules; the second group is discussed in [12].

2 A Data Warehouse and OLAP System Ground Model

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. This leads to a three-tier architecture as shown in Figure 1. Based on this three-tier architecture, we design an ASM ground model consisting of three linked ASMs, one for each of the tiers. We denote them as DB-ASM, DW-ASM, and OLAP-ASM.

The Operational Database ASM. 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. The schemata of these databases are captured by controlled functions in DB-ASM. Then 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. These operations can be defined by rules in DB-ASM. Thus, the ASM for the operational databases can be defined as follows:

`ASM` DB-ASM
`IMPORT` DW-ASM($S_1, \dots, S_m$)
`EXPORT` *extract*

```
SIGNATURE
  R_1(ar_1) (controlled), ..., R_l(ar_l) (controlled), S(1) (monitored)
BODY main = if selected(extract) then extract(S) endif
  extract(S) = case S of ... S_i : extract_S_i ... endcase
  extract_S_i = ...
```

The Data Warehouse ASM. The second tier of the architecture 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, i.e. we deal again with queries. So, the view construction operations are the only ones that link the middle tier to the top tier, which deals with OLAP.

So, DW-ASM for this tier does not look significantly different from DB-ASM. The controlled functions model the schema, while the rules capture refreshing, i.e. use rules on DB-ASM, and view creation. Thus, DW-ASM looks as follows:

```
ASM DW-ASM
IMPORT DB-ASM(extract), OLAP-ASM(V_1, ..., V_n)
EXPORT create_view, S_1, ..., S_m
SIGNATURE
  S_1(ar_1) (controlled), ..., S_m(ar_m) (controlled),
  S(1) (monitored), V(1) (monitored)
BODY
  main =
    if selected (refresh-schm) then extract(S)
      elsif selected (refresh-view) then refresh_view(V)
    endif
  refresh_view(V) = case V of ... V_i : refresh_V_i ... endcase
  create_view(V) = case V of ... V_i : create_V_i ... endcase
  create_V_i = ...
  refresh_V_i = create_V_i
```

The OLAP ASM. The top tier itself is constructed out of the dialogue objects and the OLAP operations working on them. It realises the idea of describing the OLAP interface by using so-called "dialogue types" [5], i.e. extended views on the data warehouse. So each user has a collection of open dialogue objects. 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. The major functionality, however, deals with running OLAP operations on data marts, or creating data marts. In the latter case we have to use the view creation rules imported from DW-ASM:

ASM OLAP-ASM
IMPORT DW-ASM(*create_view*, *create_view*$_1$, ..., *create_view*$_n$)
EXPORT $V_1, \ldots, V_n$
SIGNATURE
$V_1(ar_1)$ (controlled), ..., $V_n(ar_n)$ (controlled), req_queue(1) (shared),
user(1) (controlled), datamart(1) (controlled), owner(1) (controlled),
in(1) (monitored), dm(1) (monitored), usr(1) (monitored),
op(1) (monitored), view(1) (monitored),
the_view(1) (controlled), the_arity(1) (monitored)
BODY
main =
  forall *req* : req_queue
  do if user(*req.usr*) $= \perp \wedge$ *req.op* = login
    then user(*req.usr*) := 1
    elsif user(*req.usr*) = 1
    then case req.op **of**
      quit: *quit*
      close: *close*
      open: *open*
    else *OLAP_process* **endcase**
    req_queue := req_queue $\backslash$*req* **enddo**
quit =
  forall dm' **with** datamart(dm') $\neq \perp \wedge$ owner(dm') = *usr*
    do datamart (dm') := $\perp$; the_view(dm') := $\perp$ **enddo** ;
  user(req.usr) := $\perp$
close =
  owner(req.dm) := $\perp$; datamart(req.dm) := $\perp$;
  the_view(req.dm) := $\perp$
open =
  choose dm' **with** datamart(dm') = $\perp$
  do datamart(dm') := 1 **enddo**;
  create_view(req.view);
  the_view(dm') :=req.view; r :=the_arity(req.view);
  forall $x_1, \ldots, x_r$ **with** req.view($x_1, \ldots, x_r$) $\neq \perp$
    do $dm'(x_1, \ldots, x_r)$:= 1 **enddo**
OLAP_process = *get(request.in)*; *OLAP_op(req.usr,req.dm,req.op,req.in)*
get(reuest.in) = ...
OLAP_op(req.usr,req.dm,req.op,req.in) = ...

3 Refinements

Let us now approach step-wise refinement concentrating on refinements that are due to capturing additional requirements. We will first discuss informally a refinement-based method, then we will sketch formal refinement ruls.

Requirements Capture. The ASM method assumes that we first set up a ground model, which in our case would be a rather rough sketch of the functionality expressed by the three-layer architecture. In particular, we may assume separate ASMs for the database, the data warehouse and the OLAP level. Each of these ASMs will use separate *controlled functions* to model states of the system by logical structures and *rules* expressing transitions between these states. The ASMs are then linked together via queries that are expressed by these transitions.

Refinements dealing with changes that are due to identified requirements will therefore start on the top level, i.e. the OLAP ASM. We just iterate the following steps each of which corresponds to the application of some formal refinement rules:

1. *Add a new rule to the OLAP ASM*: This is used to model an additional OLAP function. It basically means that we first do not take much care about the existing specification, when we encounter new requirements. Just write down a rule in a rather rough, sketchy way.
2. *Add a new controlled function to the OLAP ASM*: This will be used to model a view that is needed for the support of any new OLAP function, provided the existing view definitions are not yet sufficient.
3. *Integrate controlled functions on the data warehouse ASM*: This will be used whenever the views for the OLAP ASM that are expressed by these controlled functions are extended. As a consequence, the view creation rules on the OLAP ASM must be changed accordingly.
4. *Add additional controlled functions to the data warehouse ASM*: This will be a consequence of the view integration.
5. *Reorganise the controlled functions on the data warehouse ASM*: This will result from the additional controlled functions that are added in the previous step. Accordingly the rules dealing with the refreshing of the data warehouse have to be adapted.
6. *Change the rules on the operational database ASM*: These rules actually model the data extraction from the operational databases and thus will be used by the refresh rules. Thus, each change to a refresh rule has to be reflected in a change to the rules on the operational database ASM.

Formal Refinement Rules. Based on our ASMs model constructs, we introduce the following notation for formalising refinement rules. For simplicity, we combine all different rules, such as add, remove, replace into a single rule as such:

$$\frac{\begin{array}{l} M \rhd \text{IMPORT: } \mathit{impList}, \quad \text{EXPORT: } \mathit{expList}, \\ \qquad \text{SIGNATURE: } \mathit{aFunc}, \ldots, \quad \text{BODY: } \mathit{aRule}, \ldots \end{array}}{\begin{array}{l} M' \rhd \text{IMPORT: } \mathit{newimpList}, \quad \text{EXPORT: } \mathit{newexpList}, \\ \qquad \text{SIGNATURE: } \mathit{newFunc}(\mathit{ar})(\text{controlled/monitored/} \ldots), \ldots, \\ \qquad \text{BODY: } \mathit{newRule}(X) = \ldots, \ldots \end{array}} \; \varphi$$

where M is the current machine, M' is the refined machine, and φ is a side condition. Words in capitals, such as "SIGNATURE", are the constructs in ASMs machines, refer to the ground model for examples. Import list such as $impList$, or export list such as $expList$, or function names, such as $aFnc$, or rule names, such as $aRule$ after the symbol $\rhd$ over the line are the objects of our interest in the current machine for further refinement. If the side condition φ is not satisfied, the current machine will remain unchanged, otherwise, the objects present over the line will be removed, and objects present below the line will be added, and the machine M is transformed to M'.

Following our informal discussion of possible refinement steps we now sketch a set of formal rules.

Adding a New Data Mart. To add a new data mart to the OLAP, we will first add a new controlled function to the OLAP-ASM which represent the new data mart, and insert it onto the export function list, then add the function to the import list and a rule for data refresh in the DW-ASM. The requirement has no impact on the DB-ASM in our design approach.

1. *Add a new controlled function to OLAP-ASM*

$$\frac{}{\text{OLAP-ASM}' \rhd \text{SIGNATURE:}newV(ar)(\text{controlled})}\ \varphi$$

   where φ is defined as, OLAP-ASM(SIGNATURE:$newV$) $= \perp$, that is, the new function name is not present in the current machine.
2. *Add a new function to the export list in OLAP-ASM*

$$\frac{}{\text{OLAP-ASM}' \rhd \text{EXPORT:}(newV(ar)}\ \varphi$$

   where φ is defined as, OLAP-ASM(SIGNATURE: $newV$) $= 1 \wedge$ OLAP-ASM(EXPORT:$newV$) $= \perp$, that is, the added function is defined under SIGNATURE, and not present under EXPORT in the current machine. Of course, it is clear that the sequence of adding the function to SIGNATURE and to EXPORT does not really matter, but an order is needed for ensuring the completeness, since we do not combine the two rules together.
3. *Add a new function to the import list in DW-ASM*

$$\frac{}{\text{DW-ASM}' \rhd \text{IMPORT: OLAP-ASM}(newV)}\ \varphi$$

   where φ is defined as, DW-ASM(IMPORT:DW-ASM($newV$)) $= \perp$.
4. *Add a new rule to DW-ASM*

$$\frac{}{\text{DW-ASM}' \rhd \text{BODY:}newRULE = \ldots}\ \varphi$$

   where φ is defined as, DW-ASM(BODY:$newRule$) $= \perp$.

By now, all the required refinements for adding a new data mart is completed. However, the refinement of adding a new data mart can bring in system optimisation concern, such as functions need to be reorganised, especially if materialised views are present. We have though grouped this type of refinements outside of the requirements capturing but system optimisation.

Adding a New OLAP Operation. To add a new OLAP operation, we will first add the new functions required, and then the operation as rules. If existing rules are affected, we replace them with new rules.

1. *Add new functions to OLAP-ASM*

$$\frac{}{\begin{array}{l}\text{OLAP-ASM}' \rhd \text{SIGNATURE:}\\ \quad newFuncc1(arc1)(\text{controlled}), \dots,\\ \quad newFuncm(arm1)(\text{monitored}, \dots\end{array}}\ \varphi$$

   where φ is defined as, OLAP-ASM(SIGNATURE:$newFuncci$) $= \perp \wedge$ OLAP-ASM(SIGNATURE:$newFuncmi$) $= \perp$, and i ranges through the new function lists for both the controlled and monitored functions.
2. *Add new rules to OLAP-ASM*

$$\frac{}{\text{OLAP-ASM}' \rhd \text{BODY: } newRule1 = \dots, \dots,}\ \varphi$$

   where φ is defined as, OLAP-ASM(BODY:$newRulei$) $= \perp$, i ranges through the new rule list.

3. *Replace a rule in OLAP-ASM*

$$\frac{\text{OLAP-ASM} \rhd \text{BODY:}aRule}{\text{OLAP-ASM}' \rhd \text{BODY: } aRule = \dots}\ \varphi$$

   where φ is defined as, OLAP-ASM(BODY:$aRule$) $= 1$. That is, the rule $aRule$, exists in the current machine.

4 Conclusion

In this paper we present a ground model and its refinements for data warehouses. Based on the general idea of data warehouses separating input from operational databases from output to dialogue-based on-line analytical processing (OLAP) systems, we can model such systems by three interleaved high-level ASMs.

We suggest a pragmatic high-level development method, which starts with requirements arising from OLAP tasks. Adding these requirements first changes a top-tier ASM dealing with the OLAP functionality. It may further propagate to changes in a middle-tier ASM dealing with the data warehouse, and it may even propagate further down to refining an ASM dealing with operational databases.

For the time being the development of formal refinement rules and pragmatic guidelines for their application is still preliminary. Our aim for the future is to complete this set of rules and to identify remaining verification tasks.

However, the lesson we learnt so far is that it seems to be very promising to set up application-dependent refinement rules, as they will enable a formal method for the design of data warehouses and OLAP systems that can be applied by users who do not know the details of the ASM formalism, while the correctness of the rules can be proven in advance.

References

1. Börger, E. The ASM refinement method. *Formal Aspects of Computing 15* (2003), 237–257.
2. Börger, E., and Stärk, R. *Abstract State Machines.* Springer-Verlag, Berlin Heidelberg New York, 2003.
3. Gyssens, M., and Lakshmanan, L. A foundation for multidimensional databases. In *Proc. 22nd VLDB Conference, Mumbai (Bombay), India* (1996).
4. Inmon, W. *Building the Data Warehouse.* Wiley & Sons, New York, 1996.
5. 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.
6. Schewe, K.-D. Specification and development of correct relational database programs. Tech. rep., Clausthal Technical University, Germany, 1997.
7. 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.
8. Schewe, K.-D., and Schewe, B. Integrating database and dialogue design. *Knowledge and Information Systems 2*, 1 (2000), 1–32.
9. Schewe, K.-D., and Zhao, J. Balancing redundancy and query costs in distributed data warehouses – an approach based on abstract state machines. In *Conceptual Modelling 2005 – Second Asia-Pacific Conference on Conceptual Modelling*, S. Hartmann and M. Stumptner, Eds., vol. 43 of *CRPIT*. Australian Computer Society, 2005, pp. 97–105.
10. 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.
11. Thomson, E. *OLAP Solutions: Building Multidimensional Information Systems.* John Wiley & Sons, New York, 2002.
12. Zhao, J., and Ma, H. ASM-based design of data warehouses and on-line analytical processing systems. Tech. Rep. 10/2004, Massey University, Department of Information Systems, 2004. submitted for publication.
13. Zhao, J., and Ma, H. Quality-assured design of on-line analytical processing systems using abstract state machines. In *Proceedings of the Fourth International Conference on Quality Software (QSIC 2004)* (Braunschweig, Germany, 2004), H.-D. Ehrich and K.-D. Schewe, Eds., IEEE Computer Society Press, pp. 224–231.
14. Zhao, J., and Schewe, K.-D. Using abstract state machines for distributed data warehouse design. In *Conceptual Modelling 2004 – First Asia-Pacific Conference on Conceptual Modelling* (Dunedin, New Zealand, 2004), S. Hartmann and J. Roddick, Eds., vol. 31 of *CRPIT*, Australian Computer Society, pp. 49–58.