A Framework for Tool–Independent Modeling of Data Acquisition Processes for Data Warehousing

Arne Harren and Heiko Tapken

Oldenburg Research and Development Institute for Computer Science Tools and Systems (OFFIS)
Escherweg 2, 26121 Oldenburg, Germany
{arne.harren,heiko.tapken}@offis.de
http://www.offis.de

Abstract. Due to their integrated and unified view over data of various operational and external systems, data warehouse systems nowadays are well established to serve as a technical fundament for strategic data analyses. Unfortunately, data acquisition which is responsible for introducing new or changed data from source systems into the data warehouse is not static and new requirements may be identified as time walks by. Therefore, comprehensibility and maintainability of data acquisition processes are crucial to the long-time success of a data warehouse.
Within the scope of this paper we sketch some aspects of our framework for tool-independent data acquisition design including the framework's architecture, the underlying design process model, and the management of metadata.

1 Introduction

Data warehouses which usually integrate cleansed data from operational and external data sources, provide a reliable platform for decision support. Besides the data warehouse database, enterprise-wide data warehouse environments typically comprise multitudes of components which store data, e.g. data sources, staging areas, operational data stores, and online analytical processing systems. Often, these environments contain more than one physical data warehouse or data mart that are fed by the source systems. For further complexity, these components may be spread across various platforms and may store their metadata in vendor-dependent repositories.

With regard to the overall coordination of the data flow, maintainability and comprehensibility are key factors for sound data acquisition. Therefore, our research project TODAY (Toolsuite for Managing a Data Warehouse's Data Supply) aims at providing a framework that supports an iterative definition of data acquisition processes from data sources to target systems, enforces a clear separation of process descriptions from their corresponding, optimized implementations, and employs commercial extraction, transformation and loading tools for process execution.

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 733–736, 2002.

2 Architecture

The framework enables modeling of the overall data flow within data acquisition processes on an extensible, tool-independent *design layer* that permits hiding tool-specific details of process implementations. After process design is done, (semi-)automatic derivation of optimized, tool-specific implementations is provided by the framework's optimization component in conjunction with tool-dependent deployment components. Thus, processes on the design layer may be modeled intuitively and do not necessarily have to be specified in an optimized way from a tool's point of view. This enables easy adoption of new functionality within a process without having the designer worrying about the impacts on later optimization.

Subsequent to the deployment of process implementations into the tools' proprietary metadata repositories, data acquisition processes can be set to production at the *execution layer*. At this layer, process execution is scheduled and monitored by a dedicated controller component.

Figure 1 sketches the framework's architecture and the relationship between the design layer and execution layer.

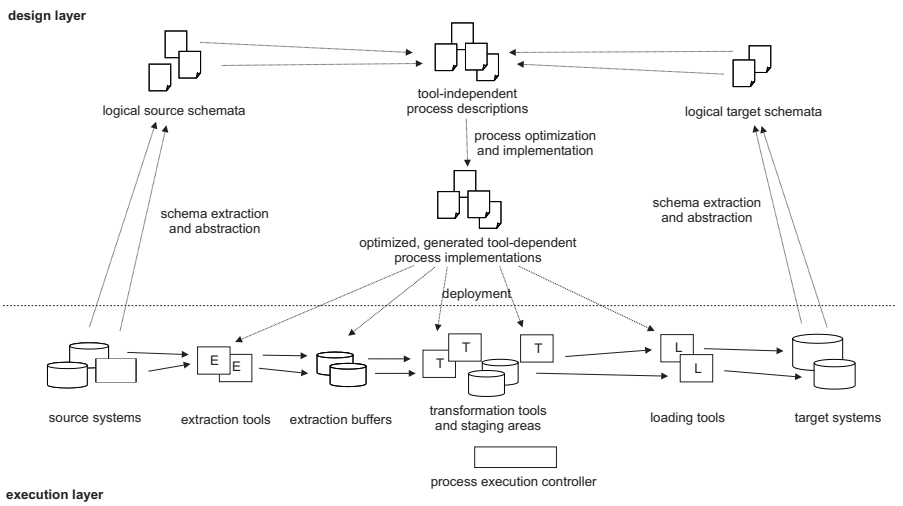

Fig. 1. Framework architecture

3 Design Process Model

For process design at the design layer, we have developed a process model which allows simple, flexible, comprehensive design of data acquisition [3]. This process model comprises the following six phases which are traversed iteratively:

1. As a prerequisite for tool-independent modeling, counterparts of physical source and target schemata have to be described on the basis of our framework's logical data model. These logical schemata hide details of the corresponding data structures, e.g. relational databases or XML files.
2. Data transformations can be modeled by means of data flow graphs in which each node describes a data processing step. Each step is associated with an operator, e.g. for providing data from a source system, deriving new attribute values, or filtering data streams. Data can be manipulated using our tool-independent transformation language TL^2.
3. In order to be able to generate process implementations in subsequent phases, appropriate tools have to be selected from a given set of available tools.
4. As a first part of process optimization, processes are partitioned and restructured based on tool capabilities and operator dependencies.
5. The previous phase results in process segments consisting of unary and binary operators that are now converted to tool-dependent operators. As a second part of process optimization, each segment is locally optimized with regard to the specific tool.
6. All generated, optimized, tool-dependent process segments are now deployed into the tools' private repositories using the required, vendor specific metadata format. Functions for data manipulation described in TL^2 are mapped to equivalent functions in tool-specific programming languages.

4 Metadata Management

Within our framework we are using a hybrid metadata architecture (cp. [1,4]). Therefore, each software component involved in the environment may use its own repository for local metadata while shared metadata are primarily stored in a central repository.

For central metadata storage, we have developed the prototyped, extensible repository system TOY (TODAY Open Repository) [2] which is capable of handling metadata based on metadata structures using the Meta Object Facility (MOF) [6]. We implemented the Common Warehouse Metamodel (CWM) [5] and use extensions to it in order to meet all needs at the design and execution layer. Access to a repository instance is possible using the system's object-oriented API or using text streams which comply to the metadata interchange format XMI [7]. By combining API and stream-based metadata exchange with XSL [8] and additional wrappers for accessing proprietary repository systems, we can reduce time and effort which needs to be spent for developing mappings between different metadata repositories.

5 Current Status and Future Work

We are currently implementing a prototype system of our framework. In this prototype we support a set of base operators for data transformations (cp. [3]) and integrate a commercial tool for extraction, transformation, and loading of data.

Completing the prototype shall show the soundness of our concepts. Evaluation will be done in cooperation with an enterprise within the telecommunication industry.

References

1. H. H. Do, and E. Rahm: *On Metadata Interoperability in Data Warehouses*, Technical report 1-2000, Institute for Informatics, University of Leipzig, 2000. 735
2. A. Harren, H. Tapken: *TODAY Open Repository: An Extensible, MOF-Based Metadata Repository System*, technical report, Oldenburg Research and Development Institute for Computer Science Tools and Systems (OFFIS), 2001. 735
3. A. Harren, H. Tapken: *A Process Model for Enterprise-Wide Design of Data Acquisition for Data Warehousing*, Proc. of the 4th Intl. Conference on Enterprise Information Systems, Ciudad Real, Spain, ICEIS Press, 2002. 734, 735
4. D. Marco: *Building and Managing the Meta Data Repository, A Full Lifecycle Guide*, Wiley Computer Publishing, 2000. 735
5. Object Management Group: *Common Warehouse Metamodel (CWM) Specification. Version 1.0*, 2001. 735
6. Object Management Group:*OMG Meta Object Facility (MOF), Specification, Version 1.3*, 1999. 735
7. Object Management Group: *OMG XML Metadata Interchange (XMI) Specification, Version 1.1*, 2000. 735
8. World Wide Web Consortium: *Extensible Stylesheet Language (XSL) Version 1.0*, W3C Recommendation, 2001. 735