Striving towards Near Real-Time Data Integration for Data Warehouses

Robert M. Bruckner[1], Beate List[1], and Josef Schiefer[2]

[1]Institute of Software Technology
Vienna University of Technology
Favoritenstr. 9 / 188, A-1040 Vienna, Austria
{bruckner, list}@ifs.tuwien.ac.at
[2]IBM Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
josef.schiefer@us.ibm.com

Abstract. The amount of information available to large-scale enterprises is growing rapidly. While operational systems are designed to meet well-specified (short) response time requirements, the focus of data warehouses is generally the strategic analysis of business data integrated from heterogeneous source systems. The decision making process in traditional data warehouse environments is often delayed because data cannot be propagated from the source system to the data warehouse in time. A real-time data warehouse aims at decreasing the time it takes to make business decisions and tries to attain zero latency between the cause and effect of a business decision. In this paper we present an architecture of an ETL environment for real-time data warehouses, which supports a continual near real-time data propagation. The architecture takes full advantage of existing J2EE (Java 2 Platform, Enterprise Edition) technology and enables the implementation of a distributed, scalable, near real-time ETL environment. Instead of using vendor proprietary ETL (extraction, transformation, loading) solutions, which are often hard to scale and often do not support an optimization of allocated time frames for data extracts, we propose in our approach ETLets (spoken "et-lets") and Enterprise Java Beans (EJB) for the ETL processing tasks.

1. Introduction

The amount of information available to large-scale enterprises is growing rapidly. New information is being generated continuously by operational sources such as order processing, inventory control, and customer service systems. Organizations without instant information delivery capabilities will surrender their competitive advantage. In order to support efficient analysis and mining of such diverse, distributed information, a data warehouse (DWH) collects data from multiple, heterogeneous sources and stores integrated information in a central repository. Traditional DWHs need to be updated periodically to reflect source data updates. The observation of real-world events in operational source systems is characterized by a propagation delay. The update patterns (daily, weekly, etc.) for traditional data warehouses and the data integration process result in increased propagation delays.

While operational systems are among other things designed to meet well-specified (short) response time requirements, the focus of data warehouses is the strategic analysis of data integrated from heterogeneous systems [3]. Traditionally, there is no

Y. Kambayashi, W. Winiwarter, M. Arikawa (Eds.): DaWaK 2002, LNCS 2454, pp. 317-326, 2002.

real-time connection between a DWH and its data sources, because the write-once read-many decision support characteristics would conflict with the continuous update workload of operational systems and result in poor reponse times.

Separated from operational systems, data warehouse and business intelligence applications are used for strategic planning and decision-making. As these applications have matured, it has become apparent that the information and analyses they provide are vital to tactical day-to-day decision-making, and many organizations can no longer operate their businesses effectively without them. Consequently, there is a trend towards integrating decision processing into the overall business process. The advent of e-business is also propelling this integration because organizations need to react much faster to changing business conditions in the e-business world [5].

The goal of a near real-time data warehouse (RTDWH, part of a so called *zero-latency DWH environment* [2]) is to allow organizations to deliver relevant information as fast as possible to knowledge workers or decision systems who rely on it to react in near real-time to new information captured by an organization's computer system. Therefore, it supports an immediate discovery of abnormal data conditions in a manner not supported by an OLTP system. Up till recently, *timeliness* requirements (the relative availability of data to support a given process within the time frame required to perform the process) were postponed to mid-term or long-term. While a total real-time enterprise DWH might still be the panacea, we will present an approach to enable DWHs to integrate particular data *"just-in-time"* to changing customer needs, supply chain demands, and financial concerns. As e-business erodes switching costs and brand loyalty, customers will consider instant service fulfillment and information access the primary relationship differentiator.

The remainder of this paper is organized as follows. In section 2, we discuss the contribution of this paper and related work. In section 3, we give an overview of requirements for real-time DWHs and discuss differences of operational data stores. In section 4 we discuss the ETL process of real-time DWHs and introduce a J2EE architecture for an ETL environment which supports a near real-time data propagation. Finally, we discuss our future work and give a conclusion in section 5.

2. Contribution and Related Work

The authors in [1] describe an approach which clearly separates the data warehouse refreshment process from its traditional handling as a view maintenance or bulk loading process. They provide a conceptual model of the process, which is treated as a composite workflow, but they do not describe how to efficiently propagate the data. Theodoratos and Bouzeghoub discuss in [11] data currency quality factors in data warehouses and propose a DWH design that takes these factors into account.

Temporal data warehouses address the issue of supporting temporal information efficiently in data warehousing systems [14]. Keeping them up-to-date is complex, because temporal views may need to be updated not only when source data is updated, but also as time progresses, and these two dimensions of change interact in subtle ways. In [13], the authors present efficient techniques (e.g. temporal view self-maintenance) for maintaining DWHs without disturbing source operations.

An important issue for near real-time data integration is the accommodation of delays, which has been investigated for (business) transactions in temporal active databases [8]. The conclusion is that temporal faithfulness for transactions has to be provided, which preserves the serialization order of a set of business transactions.

Although possibly lagging behind real-time, a system that behaves in a temporally faithful manner guarantees the expected serialization order.

In [12], the authors describe the ETL (extract-transform-load) tool ARKTOS, which is capable of modeling and executing practical ETL scenarios by providing explicit primitives for the capturing of common tasks (like data cleaning, scheduling, and data transformations) using a declarative language. ARKTOS offers graphical and declarative facilities for the definition of DWH transformations and tries to optimize the execution of complex sequences of transformation and cleansing tasks.

Our contribution in this paper is the characterization of RTDWHs, the identification of technologies that can support it, and the composition of these technologies in an overall architecture. We provide an in-depth discussion of using the J2EE platform for ETL environments of RTDWHs. The prerequisite for a RTDWH is a continual, near real-time data propagation. As a solution for this problem, we are proposing a J2EE architecture with ETL container and ETLets, which provide an efficient way of performing, controlling and monitoring the ETL processing tasks. To the best of our knowledge, there are no other approaches, which use container managed Java components for continual data propagating.

3. Real-Time Data Warehouse Requirements

A real-time data warehouse (RTDWH) aims at decreasing the time it takes to make the business decisions. In fact, there should be minimized latency between the cause and effect of a business decision. A real-time data warehouse enables analysis across corporate data sources and notifies the business of actionable recommendations, effectively closing the gap between business intelligence systems and the business processes. Business requirements may appear to be different across the various industries, but the underlying information requirements are similar – integrated, current, detailed, and immediately accessible.

Transforming a standard DWH using batch loading during update windows (where analytical access is not allowed) to an analytical environment providing current data involves various issues to be addressed in order to enable (near) real-time dissemination of new information across an organization. The business needs for this kind of analytical environment introduce a set of service level agreements that exceed what is typical of a traditional DWH. These service levels focus on three basic characteristics, which are described below.

- *Continuous data integration*, which enables (near) real-time capturing and loading from different operational sources.
- *Active decision engines* that can make recommendations or even (rule-driven) tactical decisions for routine, analytical decision tasks encountered [9, 10].
- *Highly available* analytical environments based on an analysis engine that is able to consistently generate and provide access to current business analyses at any time not restricted by loading windows typical for the common batch approach.

An in-depth discussion of these characteristics from the analytical viewpoint (in order to enable timely consistent analyses) is given in [2]. Near real-time integration for data warehouses does not minimize capturing and propagation delays of the operational source systems of an organization, which are responsible for capturing real world events. Data warehouses (in particular real-time DWHs) try to represent the history as accurately as possible (to enable tactical decision support). Therefore, late-arriving records are welcome because they make the information more complete.

However, those facts and dimension records are bothersome because they are difficult to integrate (e.g. when using surrogate keys for slowly changing dimensions).

The RTDWH provides access to an accurate, integrated, consolidated view of the organizations' information and helps to deliver near real-time information to its users. This requires efficient ETL (extract-transform-load) techniques enabling continuous data integration, which is the focus of this paper. Combining highly available systems with active decision engines allows near real-time information dissemination for DWHs. Cummulatively, this is the basis for *zero latency* analytical environments [2].

3.1. Continuous Data Integration

In e-business environments, information velocity is a key determinant of overall business growth capacity, or scalability. In an environment of growing volume, velocity increases as well. Greater volumes of data must be exchanged and moved in shorter time frames. For performing transactions faster and more efficient, operational systems are supported by business intelligence tools, which provide valuable information for decision makers. Real-time data warehouses help the operational systems deliver this information in near real-time.

ETL environments of RTDWHs must support the same processing layers as found in traditional data warehouse systems. Critical for RTDWHs is an end-to-end automation of the ETL processes. The ETL environment must be able to complete the data extracts and transformations in allocated time frames and it must meet the service-level requirements for the data warehouse users. Therefore, a continuous data integration environment facilitates better and faster decision making, resulting in streamlined operations, increased velocity in the business processes, improved customer relationships and enhanced e-business capabilities.

3.2. Real-Time DWH vs. Operational Data Stores

For handling real-time data propagations, companies often consider building an operational data store (ODS). An ODS bridges the information gap between operational systems and the traditional data warehouse. It contains data at the event detail level, coordinated across all relevant source systems and maintained in a current state. W.H. Inmon defines an ODS as an architectural construct, which is subject oriented, integrated, volatile, current valued, and contains detailed data [4].

But there are noteworthy differences between ODSs and RTDWHs. An ODS serves the needs of an operational environment while real-time data warehouses serve the needs of the informational community. The ODS and the RTDWH are identical when it comes to being *subject oriented* and *integrated*. There are no discernible differences between the two constructs with regard to those characteristics. However, when it comes to transaction support, volatility, currency of information, history and detail, the ODS and the real-time data warehouse differ significantly.

- **History of Data.** A data warehouse contains data that is rich in history. ODSs generally do not maintain a rich history of data, because they are used within operational environments and have strict requirements for the query response times. Consequently, an ODS is highly volatile in order to reflect the current status from operational sources.
- **Data Propagation.** For an ODS it is common that data records are updated. An ODS must stay in sync with the operational environments to be able to consistently operate within its environment. An ODS has predictable arrival rates for data and includes sync points or checkpoints for the loading process. RTDWH systems are generally read-only and track data changes by creating data snapshots. The absence

of frequent record-level update processing makes the load-and-access processing more efficient. Furthermore, RTDWHs have a higher variability of the size of the transactions and the variability of the rate at which the transaction arrive.

- **Transaction Support.**. Data in an ODS is subject to change every time one of its underlying details changes. An advantage of the ODS is that it is integrated and that is can support both decision support and operational transaction processing. An ODS often requires a physical organization which is optimal for the flexible update processing of data while the RTDWH system requires a physical organization which is optimal for the flexible access and analysis of data. RTDWH systems are not interwoven with operational environments and do not support operational transaction processing.
- **Aggregated Data.** The ODS contains data that is very detailed, while a RTDWH contains a lot of summary data. Data in the ODS can also be summarized, and a summarized value can be calculated. But, because the summarized value is subject to immediate change (ODSs are highly volatile), it has a short effective life. Summary data of a RTDWH is less dynamic, because aggregated values from history data are often not affected by new data.

A real-time DWH is not a replacement for an ODS. An ODS is an architectural construct for a decision support system where collective integrated operational data is stored. It can complement a RTDWH with the information needs for knowledge workers. The ODS contains very detailed current data and is designed to work in an operational environment.

4. ETL Environment for RTDWHs

We propose an architecture, which facilitates streamlining and accelerating the ETL process by moving data between the different layers without any intermediate file storage. We achieve this streamlining of the ETL process by using *ETLets* (explained in detail in section 4.1) and EJB components [17] for 1) extracting the data with high-speed J2EE connectors [15], 2) immediately parsing and converting the source data into the XML format, and 3) converting and assembling the data for the target format. This way, each layer of the ETL environment can process the source data and forward it to other layers for further processing. ETLets and EJB components are configurable Java components which perform the ETL processing tasks, such as data extraction, validation, transformation, or data assembly. They can be deployed on any Java application server and can be reused for several ETL processes.

Figure 1 shows the levels of a typical ETL process. Please note, that the processing steps for each layer may be executed in a distributed environment at different locations. The data propagation from the source system and the data movement within the ETL environment is managed in our approach by containers. We use containers, which ensure that the ETL components get instantiated for the processing and that the data is passed from one component to the next.

Because RTDWHs try to minimize the data latency and the time for propagating data from the source systems, the number of data extracts increases and the amount of data per data extract normally decreases. One of the most critical ETL processing tasks is an efficient data extraction. An ETL environment of a RTDWH must be able to establish high-performance connections to the source systems and optimize these connections. For instance, connections to databases or enterprise systems must be automatically pooled to optimize the resources and throughput times. In our approach,

containers manage the connection pools to provide an efficient access to the data sources. Containers also create, destroy, and swap in and out of memory the ETL components to optimize the ETL processing in a structured and efficient way.

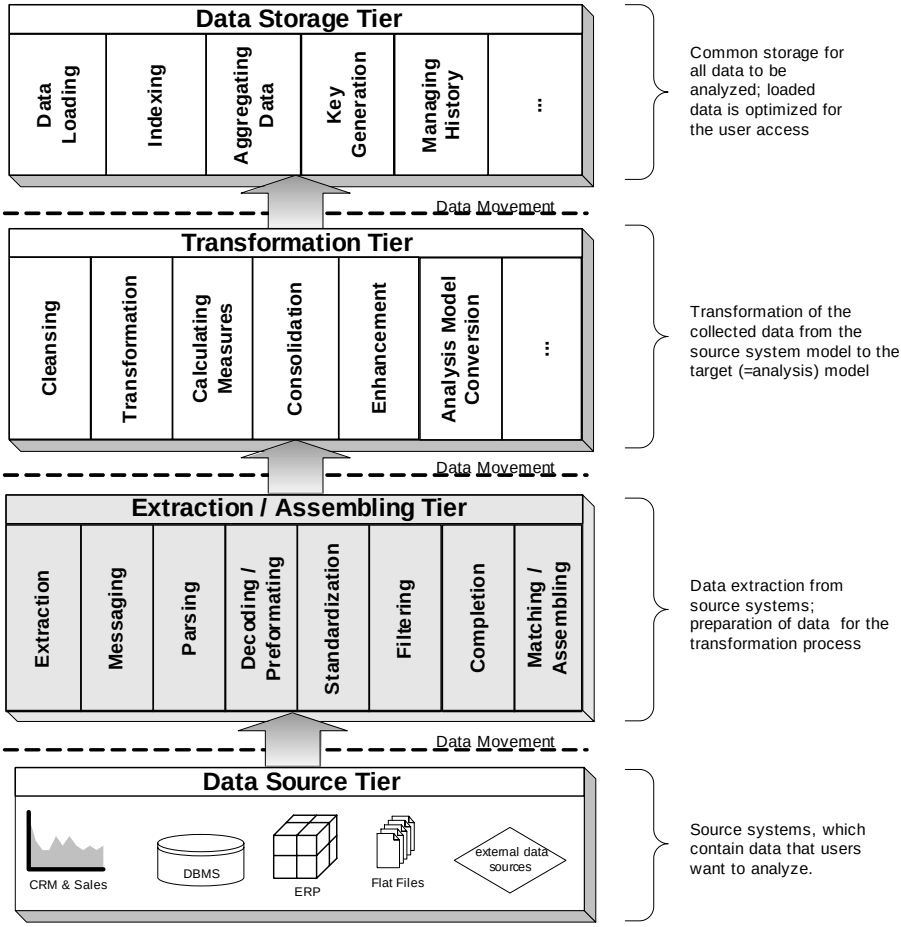

Fig. 1. ETL processing layers of a (near) real-time data warehouse

4.1 J2EE Architecture for RTDWHs

In this section we introduce a J2EE (Java 2 Platform, Enterprise Edition) architecture for an ETL environment of RTDWHs. Our architecture includes the familiar layers of a traditional data warehouse system, but also fulfills the requirements and characteristics of a RTDWH. For real-time data warehouses, a robust, scalable and high-performance data staging environment, which is able to handle a large number of near real-time data propagations from the source systems is essential.

We are proposing a J2EE environment for the extraction, transformation, and movement of the data, which fulfills these additional requirements. J2EE is a Java

platform designed for the mainframe-scale computing typical of large enterprises. Sun Microsystems (together with industry partners such as IBM) designed J2EE to simplify application development in a thin client tiered environment and to decrease the need for programming and programmer training by creating standardized, reusable modular components and by enabling the tier to handle many aspects of programming automatically [16]. J2EE environments have a multi-tiered architecture, which provides natural access points for integration with existing and future systems and for the deployment of new sources systems and interfaces as needs arise. In J2EE environments, the container takes responsibility for system-level services (such as threading, resource management, transactions, security, persistence, and so on). This arrangement leaves the component developer with the simplified task of developing business functionality. It also allows the implementation details of the system services to be reconfigured without changing the component code, making components useful in a wide range of contexts. Instead of developing ETL scripts, which are hard to maintain, scale, and reuse, ETL developers are able to implement components for critical parts of the ETL process. In our approach, we extend this concept by adding new container services, which are useful for ETL developers. A *container service* is responsible for the monitoring of the data extracts and ensures that resources, workload and time-constraints are optimized. ETL developers are able to specify data propagation parameters (e.g. schedule and time constraints) in a deployment descriptor and the container will try to optimize these settings.

Most important for an ETL environment is the *integration tier* (in J2EE environments also called *enterprise information tier*). The J2EE integration tier contains data and services implemented by non-J2EE resources. Databases, legacy systems, ERP (Enterprise Resource Planning) and EAI (Enterprise Application Integration) systems, process schedulers, and other existing or purchased packages reside in the integration tier. The integration tier allows the designers of an ETL environment to choose efficient mechanisms and resources for the data propagation that is best suited for their needs, and still interoperate seamlessly with J2EE.

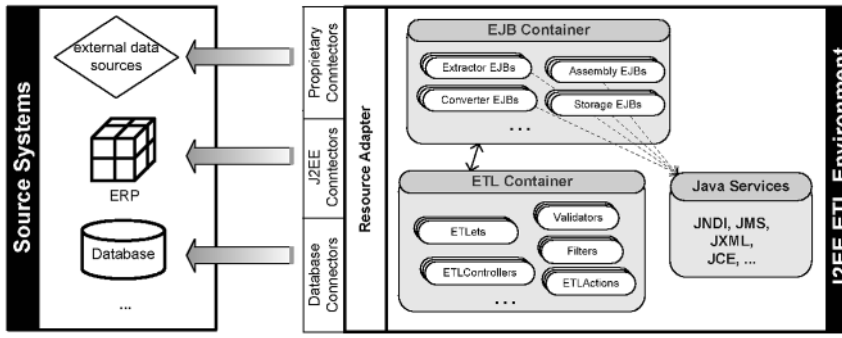

Fig. 2. J2EE ETL environment.

Figure 2 shows a J2EE ETL environment with resource adapters for the data extraction, which are available for the data propagation. Source systems for the ETL environment may require different types of access connectors. Note, that J2EE environments include a standard API for high-performance connectors. Many vendors of ERP or CRM systems (e.g. SAP, PeopleSoft) offer a J2EE connector interface for

their systems. With our architecture, ETL developers can use existing high-performance connectors with a standard API without worrying about issues like physical data formats, performance or concurrency. ETL developers can implement reusable components, which run in a container that optimizes the processing, resources and data access. The J2EE platform includes synchronous and asynchronous communication mechanisms via J2EE connectors. Furthermore, the J2EE platform includes a standard API for accessing databases (JDBC) and for messaging (JMS) which enables the ETL developers to access queue-based source systems, which can propagate data via a messaging system.

An *ETL container* is a part of a Java application server that provides services for the execution and monitoring of ETL tasks. It manages the lifecycle of ETLets and provides default implementations for some of the interfaces shown in Figure 3. There are three possibilities to implement an ETL container: 1) implementing a new ETL container for an exiting Java application servers, or extending the EJB container of an Java application server with ETL functionality, 2) extending a Java-enable database management system that includes a Java virtual machine, or 3) writing an own Java application server for the ETL container. By choosing option 1 or 2, the existing functionality of a Java application server or a database management system can be reused and extended. Furthermore, the ETL processing performance can significantly benefit from the architecture and infrastructure of the Java application server or the database management system.

Like other Java-based components, ETLets are platform independent Java classes that are compiled to platform neutral bytecode that can be loaded dynamically into and run by a Java application server. ETLets interact with schedulers or ETL clients via a predefined interface of the ETL container. The ETL container supports filters and validators, which allow on-the-fly transformations and validations of data being processed. ETL filters can globally modify or adapt the data being processed. ETL validators are used to ensure data consistency and can be used to check business rules. Filters and validators are pluggable Java components that can be specified in the deployment descriptor for the ETL container. ETL controllers are Java classes that act as intermediaries between ETLets and the underlying ETL processing model. ETL controllers coordinate the ETL tasks by processing events with ETL actions, which use the EJB components (ExtractionBean, ConversionBean, AssemblyBean etc.) to perform the processing.

Figure 3 shows the Java interfaces, which are supported by the ETL container. ETLets of the same type share an ETLet context, which is accessible to all instances of an ETLet class and can be used to find out initialization parameters or to share attributes among all ETLet instances. For instance, the ETLet context can be used for implementing caching strategies. The *ETLetSession* interface is used by ETLets to maintain the business data for one run of the ETL process. The method *getETLData()* makes the extracted data accessible to all components used for the ETL process. The *ETLetConfig* interface provides information about the initialization parameters for the ETLet. The *ETLValidator* and *ETLFilter* interfaces must be implemented by validator and filter components, which can be used in several ETL processes by specifying them in the deployment descriptor. ETL developers can use the *ETLController* and *ETLAction* interfaces for implementing complex control logic for the ETL process. For simple ETL processes developers can simply implement the *runETL()* method of the ETLet. For more complex scenarios, developers can implement ETLActions to 1) encapsulate processing steps, 2) instantiate EJB components, and 3) to invoke methods from these EJB components. The ETLActions are invoked by an ETLController, which contains the centrally managed control logic.

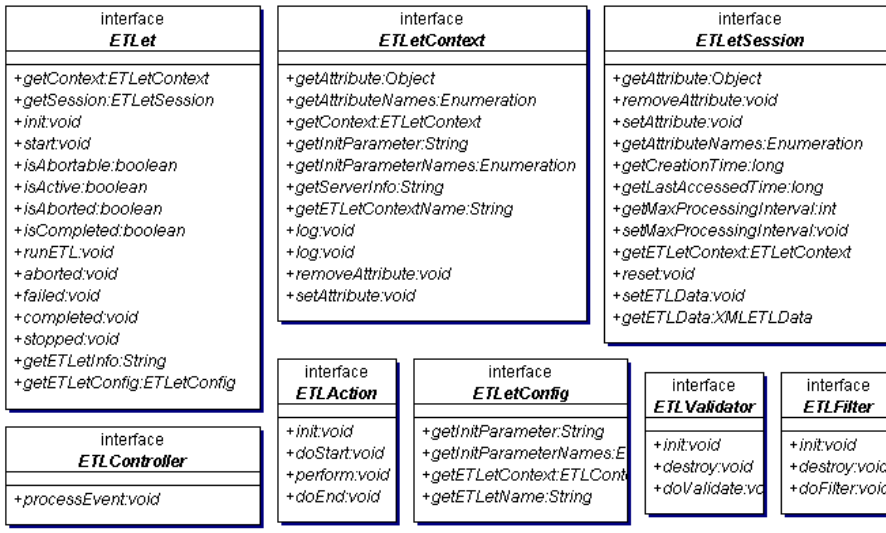

Fig. 3. Java interface supported by the ETL container.

The EJB container of the J2EE environment provides portable, scalable, available, and high-performance access to data and ETL processing. The Java application server manages the efficient access to instances of the EJB components regardless of whether the components are used locally or remotely. The environment automatically manages the implementation details of multithreading, persistence, security, transactions, and concurrent data extracts. ETL developers can implement EJBs for any type of processing of the source data.

5. Conclusion and Future Work

In this paper we investigated and characterized real-time data warehouses (RTDWH). We further examined technologies necessary to enable near real-time data integration, and described the composition of these technologies in an overall architecture. The ETL architecture uses *ETL container* and *ETLets*, which provide an efficient way of performing, controlling and monitoring the ETL processing tasks. The main advantages of ETLets are that they are lightweight Java components and that they allow a continuous ETL processing of the source data without using immediate file storage. Therefore ETLets are well suited for a near real-time data propagation. To our knowledge, we have seen no other approach, which uses container managed Java components for a continual and near real-time data propagating.

Presently, we use ETL controllers for managing complex ETL process. In our future work, we want to develop a model to formally describe and execute ETL processes. Besides managing the lifecycle of ETLets, we want to add a set of services including caching, concurrency, security, messaging, and transformation engines.

We want to use ETLets for propagating *workflow audit trail data* and related business data from various source systems. Workflows are typically executed on top of several business applications and are controlled by a workflow management system, which also tracks the execution. We want to give knowledge workers and decision makers an in-depth analysis of the executed business processes by providing valuable, process-oriented business metrics (cycle time, costs, quality, etc.) [6]. We

utilize ETLets to prepare and process the audit trail and business data. A near real-time propagation of the data is very critical, because it allows knowledge workers and decision makers an early discovery of weaknesses and problems in the process execution [7].

References

1. Bouzeghoub, M., Fabret, F., Matulovic, M.: *Modeling Data Warehouse Refreshment Process as a Workflow Application.* Intl. Workshop DMDW'99, Heidelberg, Germany, June 1999.
2. Bruckner, R.M., Tjoa, A M.: *Capturing Delays and Valid Times in Data Warehouses – Towards Timely Consistent Analyses*. To appear: Journal of Intelligent Information Systems (JIIS), forthcoming, 2002.
3. Inmon, W.H.: *Building the Data Warehouse*. 2nd ed., J.Wiley & Sons, New York, 1996.
4. Inmon, W.H.: *Building the Operational Data Store.* 2nd ed., J.Wiley & Sons, NY, 1999.
5. Inmon, W.H., Terdeman, R.H., Norris-Montanari J., Meers, D.: *Data Warehousing for E-Business*. J.Wiley & Sons, New York, 2001.
6. List, B., Schiefer, J., Bruckner, R.M.: *Measuring Knowledge with Workflow Management Systems.* TAKMA Workshop, in Proc. of 12th Intl. Workshop DEXA'01, IEEE CS Press, pp. 467–471, Munich, Germany, September 2001.
7. Kueng, P., Wettstein, T., List, B.: *A Holistic Process Performance Analysis through a Performance Data Warehouse*. Proc. AMCIS 2001, Boston, USA, pp. 349–356, Aug. 2001.
8. Roddick, J.F., Schrefl, M.: *Towards an Accommodation of Delay in Temporal Active Databases*. Proc. of 11th ADC2000, IEEE CS Press, pp. 115–119, Canberra, Australia, 2000.
9. Schrefl, M., Thalhammer, T.: *On Making Data Warehouses Active*. Proc. of the 2nd Intl. Conf. DaWaK, Springer, LNCS 1874, pp. 34–46, London, UK, 2000.
10. Thalhammer, T., Schrefl, M., Mohania, M.: *Active Data Warehouses: Complementing OLAP with Analysis Rules.* Data & Knowledge Engineering, Vol. 39(3), pp. 241–269, 2001.
11. Theodoratos, D., Bouzeghoub, M.: *Data Currency Quality Factors in Data Warehouse Design.* Intl. Workshop DMDW'99, Heidelberg, Germany, June 1999.
12. Vassiliadis, P., Vagena, Z., Skiadopoulos, S., Karayannidis, N., Sellis, T.: *ARKTOS: towards the Modeling, Design, Control and Execution of ETL Processes*. Information Systems, Vol. 26(8), pp. 537–561, 2001.
13. Yang, J., Widom, J.: *Temporal View Self-Maintenance.* Proc. of the 7th Intl. Conf. EDBT2000, Springer, LNCS 1777, pp. 395–412, Konstanz, Germany, 2000.
14. Yang, J.: *Temporal Data Warehousing.* Ph.D. Thesis, Department of Computer Science, Stanford University, 2001.
15. Sun Microsystems, *J2EE Connector Specification 1.0*, 2001.
16. Sun Microsystems, *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*, Second Edition, 2001.
17. Sun Microsystems, *Enterprise JavaBeans Specification*, Version 2.0, 2001.