Real-time Workflow Audit Data Integration into Data Warehouse Systems

Josef Schiefer[1], Jun-Jang Jeng[1], Robert M. Bruckner[2]

[1] IBM Watson Research Center
30 Saw Mill River Rd.
Hawthorne, NY, 10532
{josef.schiefer, jjjeng}@us.ibm.com

[2] Institute of Software Technology
Vienna University of Technology
Favoritenstr. 9-11 /188
A-1040 Vienna, Austria
bruckner@ifs.tuwien.ac.at

Abstract

Workflow management systems are being increasingly used by many organizations to automate business processes and decrease costs. Audit trails from workflow management systems include significant amounts of information that can be used to analyze and monitor the performance of business processes in order to improve the efficiency. Traditional approaches for using workflow audit trail for decision support purposes are lacking in transforming the workflow audit trail in real-time to valuable business information. In this paper, we introduce a data warehousing approach for extracting, transforming, and evaluating workflow audit trail data with the aim of providing a solid data foundation for process-oriented decision support. We introduce an architecture that allows the propagation and transformation of audit data from workflow management systems in near real-time to a data warehouse system. The architecture takes full advantage of existing J2EE (Java 2 Platform, Enterprise Edition) technology and uses an ETL container for providing a scalable, near real-time ETL environment for workflow audit trails.

Keywords

Workflow Management, Continuous Data Integration, Business Activity Monitoring, J2EE Containers

1. Introduction

Business Process Management Systems (BPMSs) are software solutions that support the management of the lifecycle of a business process. This includes the definition, execution and monitoring of business processes. For the execution of business processes, many organizations are increasingly using Workflow Management Systems (WFMSs) to improve the efficiency of their processes and to reduce costs. During the execution of the business process, WFMSs record many types of events, including the start and completion time of each activity, its input and output data, the assigned resources, and the outcome of the

execution. Major BPMSs and WFMSs provide comprehensive support for the early stages of the business process lifecycle but often lack capabilities for providing feedback and transparency about the process performance during execution time. Although WFMSs often log detailed information during the process execution, many BPMSs and WFMSs have difficulties in using this information for monitoring and analysis purposes. One reason for the limited capabilities is the lack of a broadly supported (industry) standard for workflow audit trail information, which is implemented by major Workflow Management Systems. Although there is a standard specification for the workflow audit trail in the reference model of the Workflow Management Coalition (1998), it is not support by most workflow management products. Therefore, up till recently, it has been very difficult for process analysts to use the workflow audit information to get a clear picture regarding the status and performance of business processes.

Most WFMSs and BPMSs offer only very basic monitoring and analysis capabilities, such as the retrieval of status information about process instances or summary information about cycle times. For a more comprehensive analysis, users have to use reporting tools from third-party vendors and write queries to retrieve data of interest. While this approach does provide basic reporting, it requires considerable configuration effort and assumes the existence of comprehensive knowledge of the process analysts to write reasonable queries.

In order to overcome these limitations, we introduce in this paper a data warehousing approach for extracting and transforming workflow audit trail data with the aim of providing a solid data foundation for process-oriented decision support. We introduce an architecture that allows the propagation and transformation of data from WFMSs in near real-time to a data warehouse system.

For a long time it was assumed that data in the data warehouse can lag at least a day if not a week or a month behind the actual operational data. That was based on the assumption that business decisions do not require up-to-date information but very rich historical data. Existing ETL (Extraction, Transformation, Loading) tools often rely on this assumption and achieve high efficiency in loading large amounts of data periodically into the data warehouse system. While this still holds true for a wide range of data warehouse applications, the new desire for monitoring information about business processes in real-time is breaking the long-standing rule that data in a data warehouse is static except during the downtime for data loading. Workflow audit trail information is a good example for this trend because it provides the most value to the users and process analysts if it is available with minimal delays. A traditional data warehouse focuses on strategic decision support. A well-developed strategy is critical, but the ultimate value to an organization is only as good as its execution. Therefore, also tactical decision support is becoming increasingly important. A variety of architectural frameworks – such as Active Data Warehousing (Brobst & Ballinger 2000), the Corporate Information Factory (Inmon et al. 2001), and Zero-Latency Enterprises (Gartner Group 1998) – have emerged recognizing the importance of tactical decision support as an extension of traditional data warehouse capabilities.

In this paper, we introduce an architecture that supports real-time integration of workflow audit trail information with the aim of propagating it continuously with minimal delay into a data warehouse system. The propagation of workflow audit trail involves several challenges:

Real-time Data Propagation. Data propagation delays can significantly decrease the value of workflow metrics that are calculated from the workflow audit trail. The ultimate goal for the dissemination of workflow events should be a zero-latency between the time the moment the workflow has been recorded and the moment it is required for monitoring and analytical

purposes. A minimal latency for workflow metrics gives process analysts more accurate and detailed information about current business situations and exceptions and allows them to respond quicker to inefficiencies in the process. It further enables automated intelligent business operations that can use the information from the data warehouse system.

Correlation of the Artifacts in the Workflow Audit Trail. The workflow audit trail includes event information about workitems (including information about resource assignments), activity instances and process instances. These runtime artifacts depend on each other and must be correlated accordingly during the data propagation when they are integrated into the data warehouse system.

Multiple Aggregation Levels. During the integration of workflow audit trails, business metrics like cycle times, costs or other quality metrics are calculated. These workflow metrics can have dependencies on other low-level metrics. An example would be *costs*: the costs of workitems can be aggregated to activity costs that can be further aggregated to process costs. The ETL process must be able to coordinate the calculations of these metrics that belong to various aggregation levels.

Adding Business Process Context. Every workflow event entails a business process context during its creation. When workflow events are propagated from WFMSs to data warehouse systems, they often include only key information of the business process context. During the calculation of workflow metrics, it is often necessary to enhance the workflow event data with additional information from other data sources. The integration process for workflow audit trails must be able to merge key information in the workflow audit trail with additional information from other data sources.

Automated Response Mechanisms. The integration process for workflow audit trails can also be utilized for instant feedback to operational systems. Notifying the business of recommendations or automatically triggering the appropriate business operations based on the evaluation of calculated workflow metrics, can change the state of a business process.

Adaptive Environment. New metrics and an additional business process context can result from changes of an existing business process. The data staging environment for workflow audit trail must be agile and adaptive to modifications of the operational environment.

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 present an architecture for integrating workflow audit trail into data warehouse systems. In section 4, we introduce an ETL subsystem for processing workflow audit trail information. Section 5 discusses in detail the ETL container which is the core component for the workflow integration. In section 6, we show an ETL processing example for calculating cycle times. Finally, in section 7 we present our conclusion and discuss our future work.

2. Contribution and Related Work

Workflow controlling and workflow history management for business process monitoring have not been extensively studied in the research literature. Some approaches emphasize the need for integrating audit trail into data warehouse systems (e.g. the process data warehouse in Sayal et al. (2002)), others are limited to a smaller set of workflow history that is managed within the WFMS. To our knowledge there has been no work that thoroughly discusses the issues of propagating and transforming audit trail in near real-time to a data warehouse system to gain valuable business information.

Sayal et al. (2002) present a set of integrated tools that support business and IT users in managing process execution quality. These tools are able to understand and process the workflow audit trail from HP Process Manager (HPPM), and can load via a loader component into the process data warehouse. Sayal et al. provide a high-level architecture and a data model for the process data warehouse, but they do not describe the data loader in detail. They show the data loader as "black-box" and do not further discuss the issues of aggregating workflow data and feeding it into the process data warehouse.

Zur Muehlen, M. (2002) provides a comprehensive overview about workflow controlling, but only discusses briefly the challenges of processing audit trail information. He discusses in detail audit trail formats from various workflow management systems, but does not show a solution for propagating and transforming these different audit trail formats.

Bouzeghoub et al. (1999) discuss an approach for modeling the data refreshment processes as a workflow. They distinguish three types of data refreshments that are triggered by a timer or data conditions in the data warehouse system: 1) client-driven refreshment, when a user causes an update of the data warehouse information, 2) source-driven refreshment, when changes to the source data trigger a refreshment, and 3) ODS-driven refreshment, which is triggered by data changes in the operational data store. They further show a design and an implementation for data refreshments from various sources as a workflow. Although they use workflow technology for modeling and executing the ETL process, they do not address the challenges of processing workflow audit trail.

An approach for history management of audit trail data from a distributed workflow system is discussed in Koksal et al. (1998). The paper describes the structure of the history objects determined according to the nature of the data and the processing needs, and the possible query processing strategies on these objects. These strategies show how to write queries for retrieving audit trail information. Unlike our approach, neither the transformation and aggregation of audit trail data nor its propagation into a data warehouse system are considered.

Muth et al. (1999) describe an architecture of a light-weight workflow management system, consisting of a small system kernel with extensions for history management. The history management consists of two components: a database system for storing the history data and a library of sub-workflows handling the access to the history database. History management sub-workflows are specified as state and activity charts. The approach resolves aggregations of historical data during workflow execution and supports complex queries on this data. The workflow history serves simple monitoring as well as business process improvement purposes. Unlike our approach, which uses a data warehouse approach, Muth et al. select and aggregate historical data at runtime and the system is only designed for a limited amount of data.

Kueng P. (1998) argues that workflow-based controlling is mainly technology-driven. The selection of process performance indicators is primarily influenced by the data, which can be gathered through the automated or semi-automated execution of activities by a WFMS and is therefore lacking the qualitative performance data and performance data about activities that are carried out manually. Our architecture addresses this issue and is able to merge audit trail data with other data sources, e.g. data from the human resource department, employee surveys etc.

Our contribution in this paper is the introduction of an architecture that allows real-time workflow audit trail integration into an existing data warehouse environment. As a solution

for this problem, we are proposing an ETL container for managing Java components that provide an efficient way for performing, controlling and monitoring the steps required for the integration of a workflow audit trail. We present a detailed discussion about the issues that have to be considered for the processing and transforming of audit trail data and how to generate workflow metrics. We further introduce an active mechanism that can evaluate calculated workflow metrics and can instantly trigger business operations based on the evaluation results. To the best of our knowledge, there are no other approaches, which use container managed Java components for a continual and real-time workflow data propagation.

3. Architecture – Process Information Factory

In this section, we present our data warehouse architecture for monitoring and analyzing business processes. Please note, that the focus of this paper is the integration of workflow audit trail data. Data modeling, application for monitoring, workflow controlling issues are out of the scope of this paper and are discussed in other publications (Zur Muehlen, M. 2002, List et al. 2000).

Our approach is based on the process information factory architecture that is shown in Figure 1. The main goal of the process information factory is to provide a data foundation for a process-driven decision support to monitor and improve the business process continuously. The process information factory consist of three major components: 1) the ETL container which is used for propagating and transforming workflow audit trail information, 2) the process warehouse which is part of the enterprise data warehouse system and is used for storing a rich set of historical workflow data for the strategic decision support and 3) the process data store, which includes very detailed up-to-date workflow data of currently running processes and can be used for tactical and operational decision support.
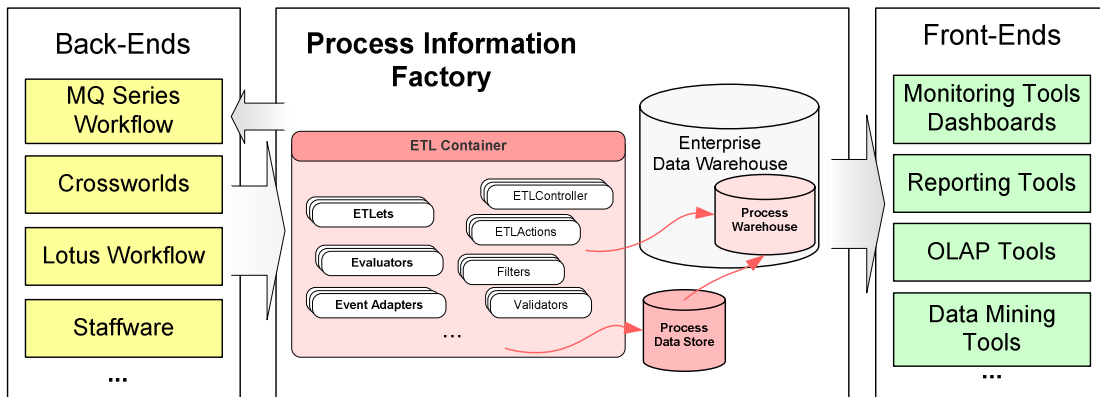

Figure 1. Process Information Factory with ETL Container

Note that process warehouse and process data store are conceptually equivalent to traditional data warehouses and operational data stores (ODSs) with the only difference that they are used to store workflow data and any process related information. Therefore, the process information factory adds a process perspective to an analytical environment. The process information factory has an open architecture that supports various source systems (open back-ends) and various applications for a process-oriented decision support (open front-ends).

In this paper, we only want to focus our discussion on the ETL container component of the process information factory. The ETL container ultimately prepares the workflow audit trail for loading it into the data warehouse. A detailed description of the ETL container and the components that are managed by the ETL container can be found in section 5.

4. ETL Subsystem for Workflow Audit Trail

Figure 2 shows the ETL process for workflow audit trails that is supported by the ETL container. The figure shows the extraction, transformation and loading steps. Additionally, we included an evaluation step for workflow metrics that allows to evaluate workflow metrics immediately after they have been calculated. Based on the result of the evaluation, the ETL container can either send out notifications or trigger any kind of business operations.
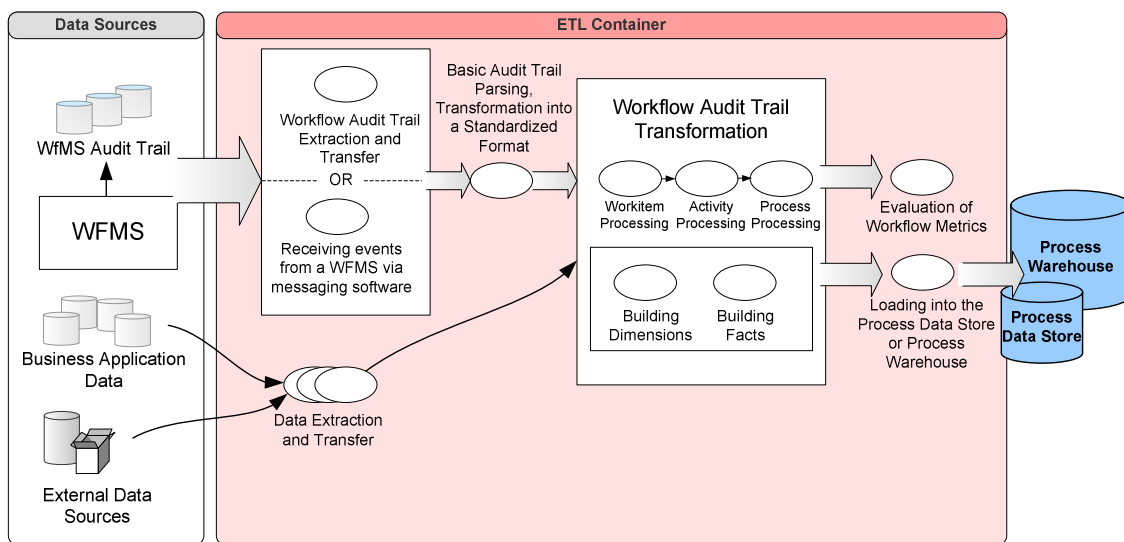

Figure 2. ETL Process for Workflow Audit Trail

For the extraction step, we distinguish two ways of receiving audit trail data from the WFMS: 1) reading the logs of the WFMS that are available in database tables or log files, or 2) receiving workflow events that have been sent from the WFMS via messaging software. The major difference between the two options is that option 1 is a pull strategy for receiving the audit trail data, and option 2 is a push strategy. Sending audit trail data via messaging middleware has many advantages and is in our opinion the better option for implementing a real-time ETL solution for the workflow audit trail.

Once the audit trail data is parsed and standardized, it is passed through a number of components that store all relevant information about workitems, activity instances and process instances, calculate and evaluate workflow metrics, and load it into dimension and fact tables of the data warehouse system. For the calculation of workflow metrics, the ETL container might also use information from other data sources. For instance, information about the customer satisfaction or performance data from business applications that have been invoked by the WFMS can significantly increase the business value of workflow metrics. Depending on the requirements, there may be additional steps to validate and filter audit trail data within the ETL process.

Workflow events (e.g. workitem events, activity events, process events) can require a very different kind of ETL processing. The processing of the same event type (e.g. activity events) might vary significantly between various processes. For example, a human resource process (e.g. hiring process) will include other workflow metrics than a typical order process. Therefore, the processing in the ETL layer can vary for the same event type although the events might come from the same workflow engine.

Storing all details of the workflow audit trail in a historical database will result in huge databases and execution times for complex queries against this database. Therefore, it is very critical that the ETL process includes a selection and consolidation of the key business data within the workflow audit trail in order to avoid storing useless data for the monitoring or analysis.

Furthermore, the ETL process must be able to handle complex metric calculations that can include aggregations of already calculated workflow metrics or merging the workflow audit trail with data from external data sources (e.g. customer survey information for computing the customer satisfaction). For instance, the costs of a business case (represented by a workflow instance) can be calculated by aggregating all costs that were caused by the activities of this business case. When a business case is completed, the ETL container can immediately start calculating the workflow metrics for the completed business case.

Figure 3 shows the processing stages for populating the workflow metrics into the various fact tables. Some of the ETL processing steps can receive data from previous processing steps to be able to do some aggregations for this data. It is very critical that the ETL environment can effectively manage the dependencies of these processing steps.
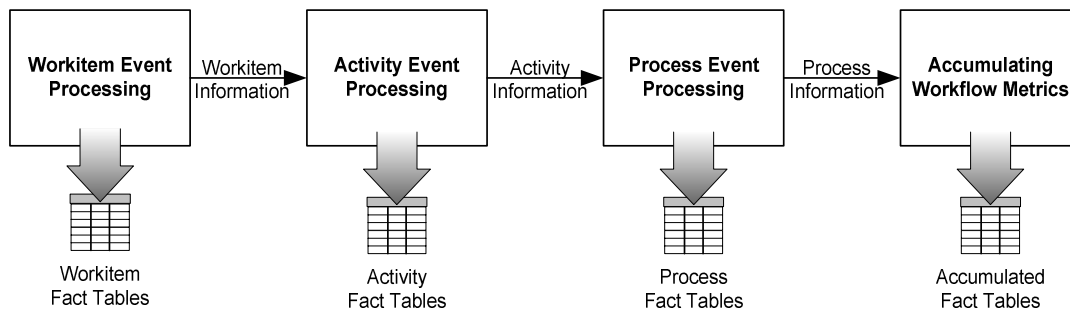

Figure 3. Populating Fact Tables

In our approach, the ETL container automatically triggers the ETL processing logic via Java components, called *ETLets*, which are dedicated to the processing of a specific type of workflow event. The ETL container also ensures that the ETLets can access data from previous processing steps.

During the processing of workflow audit trail, dimensional data will also be resolved or populated. All fact tables in Figure 3 are referencing a set of dimension tables. The foreign keys for referencing these dimension tables have to be resolved by either creating a reference to an existing record in a dimension table or by first populating a dimension record and then referencing this record. Which option to use depends on the type of dimension table. Dimension tables for resources and business objects often have to be populated during the processing of a workflow event. For instance, if a customer dimension table is referenced by a

fact table shown in Figure 3, and a new customer is part of the workflow transactions, a new customer record has to be created in the customer dimension table during the processing of the workflow event to avoid inconsistencies and non-valid references. Figure 4 shows different types of dimension tables that are populated during the workflow audit trail processing. Please note that the different types of resolvers can include complex logic to correctly reference or populate dimensional data and to maintain the history of this data. A detailed discussion about modeling and managing dimensions can be found in Kimball et al. (1998).
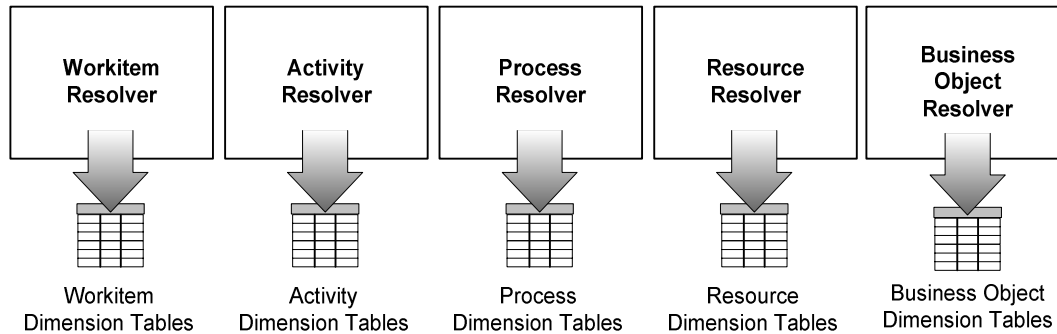

Figure 4. Resolving and Populating Dimensional Data

A detailed discussion of the schema for fact tables and dimension tables for workflow data is out of the scope of this paper. Zur Muehlen, M. (2002) and List et al. (2000) give a comprehensive description of data warehouse models for workflow audit trail data.

5. ETL Container

The ETL container is a part of a Java application server that provides services for the execution and monitoring of the ETL tasks. The ETL container complies Sun's J2EE (Java 2 Platform, Enterprise Edition) architecture and provides a robust, scalable and high-performance ETL environment, which is able to handle a large number of small data extracts or events that can require a complex processing logic. Similar to Java technology for web applications, where servlets and JSPs took the place of traditional CGI scripts, our approach uses Event Adapter, ETLet, and Evaluator components (see Figure 5) that replace traditional ETL scripts, which are often hard to maintain, scale, and reuse.

ETL scripts are not suitable for an event-driven environment where data extracts and data transformations are very small and frequent, because the overhead for starting the processes and combining the processing steps can dominate the execution time. Because of this constraint in the traditional approach, we use the ETL container to manage and optimize the ETL processing. The ETL container handles each workflow event by a lightweight Java thread, rather than a heavyweight operating system process. This approach also simplifies the programming tasks for developers who have to implement the logic for the workflow event processing, since the ETL container takes responsibility for various system-level services (such as threading, resource management, transactions, caching, persistence, and so on). In our approach, we extend this concept by adding new container services, which are useful for the audit trail processing and can be leveraged by the developers. An example of a container services is the evaluation service, which significantly reduces the effort for evaluating

calculated workflow metrics. This arrangement leaves the developer with a simplified development task and allows the implementation details of the system and container services to be reconfigured without changing the components.
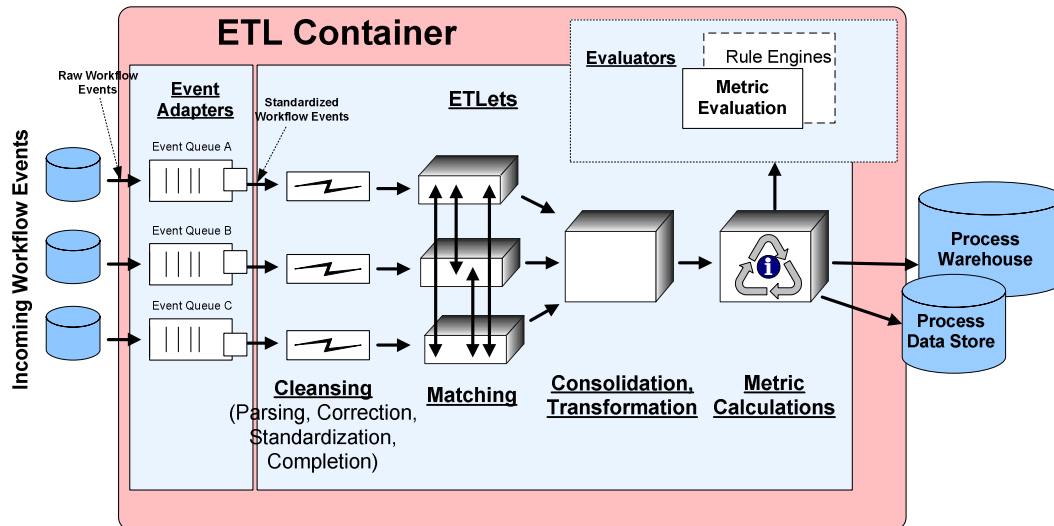

Figure 5. ETL Container for Processing Workflow Events

Figure 5 shows the core ETL components that are managed by the ETL container: 1) Event Adapters, 2) ETLets, and 3) Evaluators. Each of these components must implement a certain interface that is used by the ETL container in order to manage the component. The ETL container automatically instantiates these components and calls the interface methods during the components' lifetime. Figure 6 shows the component interfaces with some sample implementations.

Every incoming event source requires an Event Adapter, which receives and dequeues the messages and dispatches them in a standardized event format for a further ETL processing. The purpose of Event Adapters is to unify the different event formats from various WFMSs. After the dispatching of the workflow events in the Event Adapter the ETL container assigns each event a separate thread for the ETL processing. After assigning the thread, the ETL container calls the ETLets that do the ETL processing. For one event type (e.g. activity events), there might be several ETLets that are executed. Please note, that for the execution of all ETL processing steps no intermediary storage is required. The ETLet components must implement the ETLet interface which includes the *runETL()* method. The ETL developers have to implement this method with the ETL processing logic for the workflow events. The ETLet processing can include the calculation of workflow metrics and storing these metrics in the process data store or the process warehouse. ETLets can also publish the workflow metrics to allow the ETL container to pass these metrics to Evaluator components that have subscribed to the metric type. The Evaluator components can be either implemented by the ETL developers or they forward the evaluation requests to rule engines for more sophisticated evaluations.
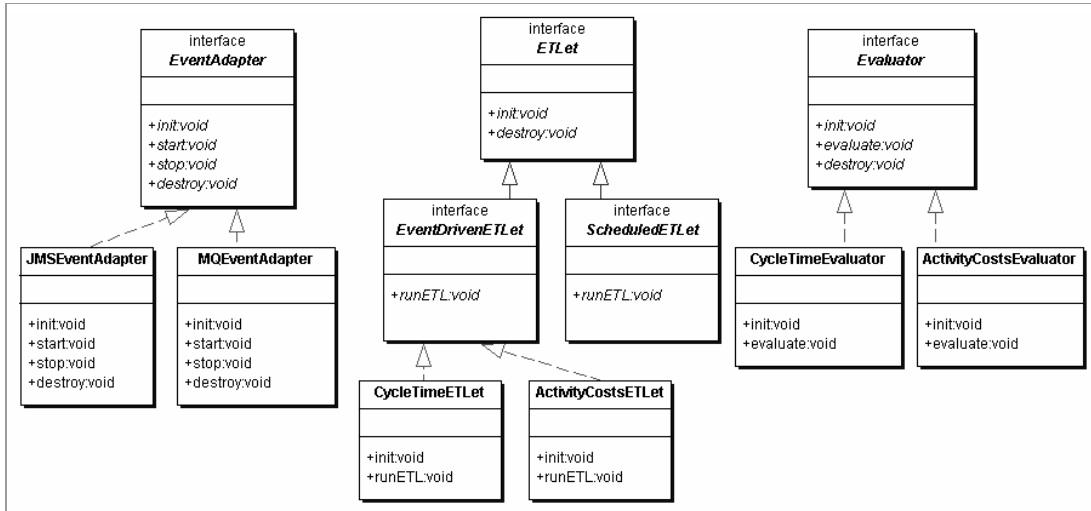
Figure 6. Interfaces/Classes of the ETL Components

Figure 7 shows the internal processing of workflow events. The components shown with round boxes are the ETL components that are managed by the ETL container and have to be provided by the ETL developers. The components shown with square boxes are internal ETL container components that are used to bind all ETL components together. Please note that ETL developers never see or have to deal with these internal components. We show these internal components only for illustration purposes.
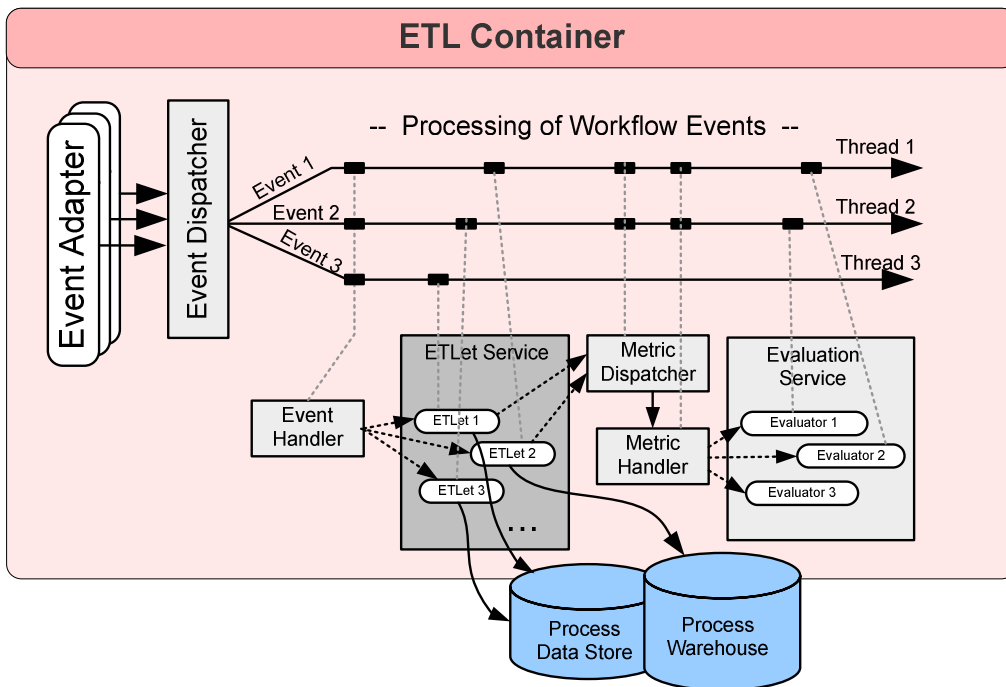

Figure 7. Multithreading within the ETL Container

6. ETL Processing Example

In this section, we show a simple example of an ETL application with an ETLet and an Evaluator for calculating and evaluating the cycle time of a process instance. The ETLet in our example calculates the cycle times by processing the PROCESS_COMPLETED workflow event. The ETLet implementation shown in Figure 8 has a method called *runETL()* that does the processing of workflow events.

```
public class CycleTimeETLet extends DBETLetBase {
  PersistentCache persistentCache = new PersistentCache();
  ...
  // ETL processing for the events PROCESS_STARTED and PROCESS_COMPLETED
  public void runETL(WorkflowEvent event, MetricPublisher metricPublisher) throws Exception
  {
    if(event.getEventID().equals("PROCESS_STARTED")) {
       persistentCache.put(event.getProcessInstanceID(),event);
    }
    if(event.getEventID().equals("PROCESS_COMPLETED")) {
      WorkflowEvent startEvent = persistentCache.get(event.getProcessInstanceID());
      // Calculate the cycle time in seconds
      int cycleTime = (event.getTimeCreated().getTime() –
                    startEvent.getTimeCreated().getTime())/1000;
      // Store the cycle time in the database
      Connection con = getDBConnection();
      PreparedStatement prepStmt = con.prepareStatement(
                             "INSERT INTO fact_table set  cycle_time=?, ...");
      prepStmt.setInt(1, cycleTime);
      prepStmt.executeUpdate();
      ...
      prepStmt.close();
      closeDBConnection();

      // Publish the cycle time metric
      HashMap metricMetaData = new HashMap();
      metricMetaData.put("PID",event.getProcessInstanceID());
      metricPublisher.publish("CycleTime", new Integer(cycleTime), metricMetaData);

      // Cleanup
      persistentCache.remove(startEvent.getProcessInstanceID());
    }
  }
  public boolean failed(WorkflowEvent event, Exception exception) {
      // ETL Container will automatically store events and exceptions in an error table
      return false;
  }
}
```

Figure 8. ETLet Example

The *runETL()* method calculates the cycle time by extracting the timestamp of the PROCESS_COMPLETED event and determining the time differences between the PROCESS_STARTED event of the same process instance. The PROCESS_STARTED event is retrieved from a persistent cache that contains all historical events that have been emitted from currently running workflow instances. After calculating the cycle time, the ETLet opens a pooled database connection, writes the cycle time and some other attributes of the received

event message into the fact table of a data warehouse and then closes the connection. Finally, the ETLet publishes the calculated cycle time that allows the ETL container to forward it to Evaluator components for a further evaluation. Note that there is no *try-catch* block in the source code. The ETL container manages exception handling, database connection pooling, and other administrative tasks automatically. If an exception is thrown during runtime, the ETL container will automatically call the *failed()* method of the same ETLet. If the *failed()* method is not able to handle the exception, it can return false which tells the ETL container to store the event and exception information in a separate error table for a manual correction of the problem at later time.

```
public class CycleTimeEvaluator extends MetricEvaluator {
  int upperLimit;
  static Logger logger = ETLLogger.getLogger(CycleTimeEvaluator.class);

   // Evaluates the cycle times
   public void evaluate(String metricName, Object metricValue, Map metricMetaData) {
      if(metricName.equals("CycleTime")) {
         if(((Integer)metricValue).intValue() > upperLimit) {
            logger.warn("Process instance " + metricMetaData.get("PID") +
                        " exceeded upper limit.");
        }
       }
     }
     public void init(MetricEvaluatorConfig config) {
        super.init(config);
        upperLimit = Integer.parseInt(config.getInitParameter("CycleTimeUpperLimit"));
     }
}
```

Figure 9. Evaluator Example

Figure 9 shows an example of an implementation for an Evaluator in the above scenario. The Evaluator compares the cycle time with an upper limit that is specified in the deployment descriptor.

Figure 10 shows the sections in the deployment descriptor for the aforementioned ETLet and the Evaluator. The ETLet section carries information about the ETLet name, the implementation class, the triggering events, and the published metrics. The Evaluator section contains information about the Evaluator name, the implementation class, the upper limit for the cycle times and the name of the evaluated metrics. The ETL container uses the information from these sections for instantiating and initializing the ETL components. The deployment descriptor includes many other parameters that are not shown in Figure 10 (e.g. database configuration for the error tables, global parameters, concurrency control parameters of workflow events, etc.).

```
...
<!-- ETLet DD Section -->
<ETLet>
  <name>CycleTimeETLet</name>
  <impl-class>CycleTimeETLet</impl-class>
  <ETLet-triggers>
      <event-trigger event-id="PROCESS_STARTED"/>
      <event-trigger event-id="PROCESS_COMPLETED"/>
  </ETLet-triggers>
  <published-metrics>
      <metric-name>CycleTime</metric-name>
   </published-metrics>
</ETLet>
...
<!-- Evaluator DD Section -->
<metric-evaluator>
  <name>CycleTimeEvaluator</name>
  <impl-class>CycleTimeEvaluator</impl-class>
  <init-param>
      <param-name>CycleTimeUpperLimit</param-name>
      <param-value>86400</param-value>
  </init-param>
  <evaluated-metrics>
      <metric-name>CycleTime</metric-name>
  </evaluated-metrics>
</metric-evaluator>
...
```

Figure 10. Deployment Descriptor Example

7. Conclusion and Future Work

A near real-time propagation of workflow audit data is very critical, because it allows workflow participants and process analysts an early detection of weaknesses and problems in the process execution. In this paper, we discussed the challenging issues of integrating workflow audit trail data into data warehouse systems. We introduced the concept of an ETL container that enables a near real-time data integration and provides services for the extraction, parsing and translation of workflow audit trail data. The main advantages of the ETL container are summarized as follows:

1. Clean separation of the extraction logic, transformation logic and evaluation logic,

2. Event Adapters are used to standardize the workflow events that makes the ETL processing logic less complex and more reusable,

3. Java components (called ETLets) are used for the ETL processing that can include complex calculations and aggregations of workflow metrics,

4. Mechanism for evaluating calculated workflow metrics, and

5. Usage of lightweight Java threads for the event processing to support the processing of a large number of workflow events concurrently in near real-time.

To the best of our knowledge, we have seen no other approach, which uses container managed Java components for a near real-time integration of workflow events.

There are several future works on this research. We are building a distributed environment for ETL containers that allows them to work together in a server farm. We are also developing an evaluation framework that allows rule engines to be plugged into the ETL container for more dynamic metric calculation and evaluation. Furthermore, we want to add new container services, which are useful for ETL developers, such as services for caching, concurrency control, and security.

References

Bouzeghoub, M., Fabret, E., Matulovic-Broque, M. (1999), 'Modeling the Data Warehouse Refreshment Process as a Workflow Application', DMDW 99, Heidelberg, Germany.

Brobst, S. A. and Ballinger, C. (2000), 'Active Data Warehousing', *Whitepaper EB–1327*, NCR.

Gartner Group (1998), 'Introducing the Zero-Latency Enterprise', *Research Note COM-04-3770*.

Inmon, W. H, Imhoff, C., Sousa, R., (2001), *Corporate Information Factory, Second Edition*, J.Wiley and Sons, New York.

Kimball, R., Reeves, L., Ross, M., Thornthwaite, W. (1998), *The Data Warehouse Lifecycle Toolkit*, John Wiley & Sons.

Koksal, P., Alpinar, S. N., Dogac, A. (1998), 'Workflow History Management', ACM Sigmod Record 27(1): 67-75.

Kueng, P. (1998), 'Supporting BPR through a Process Performance Measurement System', Business Information Technology Management, New Delhi.

List, B., Schiefer, J., Tjoa A M., Quirchmayr G. (2000), 'A Generic Data Model for the Process Warehouse - An Approach fzor Multidimensional Business Process Analysis', *Selected Aspects of Knowledge Discovery for Business Information Systems*, Kluwer Academic Publishers, Boston, USA.

Zur Muehlen, M. (2002), 'Workflow-based Process Controlling', PhD Thesis, Muenster, Germany.

Muth, P., Weissenfels, J., Gillmann, M., Weikum, G. (1999), 'Workflow History Management in Virutal Enterprises Using a Lightweight Workflow Management System', RIDE99, Sydney, Australia.

Sayal, M., Casati, F., Dayal, U., Shan M. (2002), 'Business Process Cockpit', VLDB 2002, Peking.

Workflow Management Coalition (1998), Audit Data Specification.