Managing Continuous Data Integration Flows

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

[1]IBM Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
{josef.schiefer,jjjeng}@us.ibm.com

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

Abstract. Understanding the performance and dynamic behavior of workflow is crucial in being able to modify, maintain, and improve it. A particularly difficult aspect of measuring the performance of workflows is the dissemination of event data and its transformation into business metrics. In this paper we introduce an architecture that supports a continuous integration of event data from various source systems in near real-time into a data warehouse environment. The proposed architecture takes full advantage of existing J2EE (Java 2 Platform, Enterprise Edition) technology and uses an ETL container for the event data processing. We discuss the challenges of managing flows for continuously integrated data and show how a container environment can provide services that facilitate the flow management.

1. Introduction

Since customers demand faster services, more transparent business transactions, and instant responses to business situations or exceptions, organizations without real-time information delivery capabilities and feedback of their business operations will sacrifice their competitive advantage. Workflow management systems log the activities that occur in the executed workflow, and even non-workflow-specific operational systems often log activities that users perform. Thus, within and without workflow systems, there is often a rich source of activity logs that can be transformed in valuable business metrics that provide an accurate feedback about the quality and performance of executed business operations.

Separated from operational systems, data warehouse and business intelligence applications are used for strategic planning and decision-making. As these applications have matured over time, it has become apparent that the information and analysis methods they provide are also vital for tactical day-to-day decision making processes, and many organizations can no longer operate their businesses effectively without them. Consequently, there is a trend towards integrating decision processing into business processes in an organization. Examples are manufacturing processes, click-streams in web-personalization, and call detail records in telecommunication [4].

In this paper, we introduce an approach for managing continuous data integration flows with the aim of propagating data with minimal latency into a data warehouse environment. Continuous data integration flows are a necessity for achieving a minimal latency between the moment the data has been become available and the time it is required by the users for monitoring and analytical purposes. Data propagation delays can significantly decrease the value of the integrated data for the users, since managers and operational staff have to respond very quickly and prudently to business situations and business exceptions in order to improve customer service.

When data is integrated continuously and concurrently, traditional batch-oriented ETL processing cannot satisfy the real-time requirements. The data processing flows are on a very granular level and data processing mechanisms must be able to handle a very large number of flow instances. Continuous data integration flows are aimed to process and transform incoming events individually. The types of received events are used to trigger the adequate logic for processing the events. Multiple events may depend on one another and must be correlated accordingly during the data propagation when they are integrated into the data warehouse system. Furthermore, because of the parallel processing of events, concurrency issues can arise.

Since continuous data integration enables a more accurate monitoring of business processes, it can also be extended with automated response mechanisms that are part of the data integration itself. This is usually referred to as active data integration. In the case where an automatic response is required, the data integration process also evaluates or analyzes integrated data and triggers business operations.

The remainder of this paper is organized as follows. In section 2, we discuss related work. In section 3, we give an overview of continuous data integration and discuss the differences of traditional batch-oriented approaches. Section 4 introduces the concept of an ETL container which is proposed as a solution for continuous data integration. In section 5, we discuss the flow management capabilities of the ETL container. Finally, we discuss our future work and give a conclusion in section 6.

2. Related Work

Approaches for measuring the performance of business processes depend on the ability of collecting data from an executing process. Selby et. al describe in [7] a system, Amadeus, for automated collection and analysis of process metrics. It acts as an event monitor, allowing the triggering of actions based on certain events.

Wolf and Rosenblum [9] use a hybrid of manual and automated collection methods to collect event data from a build process. Krishnamurthy and Rosenblum [5] built a system event monitor, Yeast, which can record events occurring on computer, and can react to those events. Barghouti and Krishnamurthy [2] describe a process enactment system that alternatively could be used to collect event data. The system is based on watching for events and matching the events and their contexts with the current state of the process. Lacking a process model, their infrastructure could be used to simply collect the event data.

Bouzeghoub et al. discuss in [3] 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 show a design and implementation for data refreshments from various sources as a workflow and demonstrate that the refreshment process is more complex than the view maintenance problem [10], and different from the loading process. The authors propose for the implementation of the ETL processes the usage of workflow management systems (WFMSs) and active rules which are executed under certain operational semantics.

Vassiliadis et. al. describe in [8] an approach that uses workflow technology for the management of data warehouse refreshment processes. The authors propose a meta model for data warehouse processes that is capable of modelling complex activities, their interrelationships, and the relationship of activities with data sources from a logical, physical and conceptual perspective. The physical perspective of the meta model covers the execution details of the data warehouse processes. The logical and conceptual perspectives allow the modelling of complex data warehouse processes as workflows.

The approaches discussed in [3] and [8] focus on data refreshment processes with batch-oriented data integration. The ETL container proposed in this paper also supports batch-oriented ETL processing flows that are triggered by events or by a scheduler. In addition, the ETL container allows the definition and efficient execution of processing flows for individual event types which is very crucial for continuous data integration and which is very difficult to achieve with a workflow management system.

3. Continuous Data Integration

This section describes a framework for continuous data integration as the context of this work. Figure 1 shows the process for continuously integrating data from various source systems. The processing steps are not equivalent to the traditional ETL because the underlying assumptions for the data processing and the latency requirements are different. Traditional ETL tools often take for granted that they are operating during a batch window and that they do not affect or disrupt active user sessions. If data is meant to be integrated continuously, a permanent stream of data must be integrated into the data warehouse environment while users are using it. The data integration process is performed in parallel and uses regular database transactions (i.e. generating *inserts* and *updates* on-the-fly), because in general database systems do not support block operations on tables while user queries simultaneously access these tables.

Figure 1 shows three stages for continuous data integration: 1) Extraction, 2) Data Processing, and 3) Evaluation. Please note that we do not include a separate loading stage in the process because data is not buffered and prepared for bulk loading. We extend the traditional data integration process with an evaluation stage that allows the system responding to conditions or irregularities in the integrated data. An evaluation of continuously calculated business metrics can be very valuable to the business because it promotes intelligent responses to business operations and proactive notification mechanisms in near real-time.

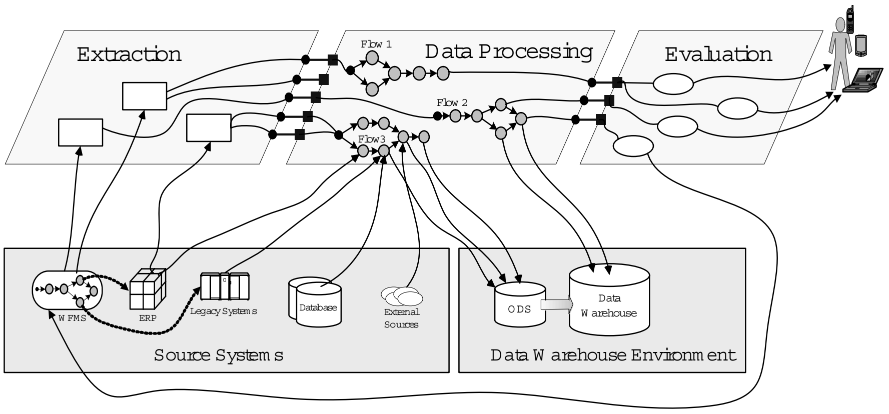


Fig. 1: Continuous Data Integration Process

Data Extraction. Real-time integration of data from various operational sources addresses the issue of timeliness by minimizing the average latency from when a fact is first captured in an electronic format somewhere within an organization until it is available for the knowledge worker who needs it. In general, not all but only a relatively small amount of data represents transactions or other relevant information that must be captured continuously and "live" from transactional systems. The ultimate goal is to integrate the transactional data in near real-time with the historical information in the data warehouse. We need to be able to accept real-time feeds of transactional key business data, which can be a burden for the systems involved if they work in a synchronous mode using a direct connection. Therefore, an asynchronous messaging infrastructure with a publish-and-subscribe architecture is very advantageous because of the following reasons:

- Operational systems are able to publish (push) their transaction data to message queues without being slowed down or even disturbed if a data integration process fails.
- Data warehouses are able to decide themselves when to pull new data from operational sources. Therefore, they subscribe to particular message queues, filled by operational systems.
- Messaging middleware provides reliable data delivery.

Data Processing. Continuous data streams require light-weight data processing of the events that were raised in the source system and propagated in near real-time to the data warehouse environment [1]. The data processing can include any type of data transformation, data cleansing, the calculation of business metrics, and storing the metrics in a database table. Since the data has to be integrated with minimal delay, an architecture is needed that facilitates streamlining and accelerating the data processing by moving data between the different processing steps without any intermediate file or database storage. Traditional (batch-oriented) ETL scripts are not suitable for event-driven environments 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. Another limitation of ETL scripts is that they are written for a specific task in a self-sustaining manner, and don't provide any kind of interfaces for data inputs and outputs. Because of this constraint in the traditional approach, a data processing environment must be very light-weight and scalable to handle a large number of processing flows. Continuous data streams often only include key business data from the transactional source systems. Therefore, the calculation of complex business metrics frequently requires additional information from other data sources. For instance, in the case of an order process, the workflow events do not include detailed information about the orders and customers. Nevertheless, order and customer information might be needed for the calculation of business metrics about the order transactions.

Evaluation. The monitoring of business operations often entails a direct or indirect feedback into a workflow management system or an operational system. This response can be done either manually or automatically and enhances the business intelligence of operational systems. This is usually referred to as closed loop analysis. In the case when an automatic response is required, the data integration process has to evaluate the integrated data on-the-fly and trigger business operations based on the results of the evaluation. One major challenge of evaluating business metrics is the binding of the evaluation logic to the data processing logic. During the data processing, business metrics are generated that have to be passed to the evaluation logic.

4. Data Integration with an ETL Container

An ETL container provides a robust, scalable, and high-performance data staging environment, which is able to handle a large number of data extracts or messages from various source systems in near real-time. In [6] we introduced the architecture of the ETL container which provides services for the execution and monitoring of event processing tasks. Container services are responsible for creating, initializing, executing and destroying the managed components for the data extraction, data processing and evaluation. In this paper, we want to focus on the flow management capabilities of the ETL container. Before showing the approach taken for managing continuous data processing flows, we will briefly describe the components of the ETL container.

An ETL container is a component of a Java-based application server and is used to manage the lifecycle of ETL components. It provides services for the execution and monitoring of ETL tasks. Three types of ETL components are managed by the container: 1) Event Adapters, 2) ETLets, and 3) Evaluators. Event Adapters are used to extract and receive data from source system, and they unify the extracted data into a standard XML format. The container controls the processing steps performed by a specialized Java components - so-called *ETLets*. ETLets use the data extracted by Event Adapters as input and perform the ETL processing tasks. ETLets also publish business metrics that can be subsequently evaluated by Evaluator components. Figure 2 shows how these ETL components work together.

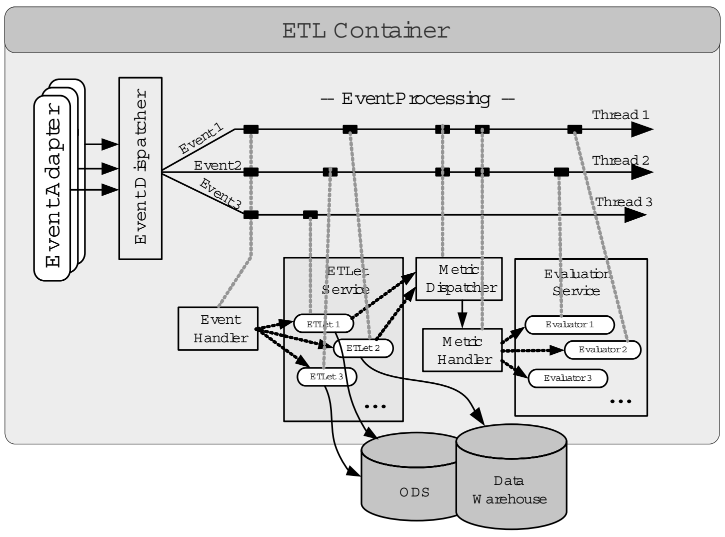


Fig. 2: Multithreading in the ETL Container [6]

The ETL container handles incoming events with a lightweight Java thread, rather than a heavyweight operating system process. Figure 2 shows the internal event processing of the container. The components shown with round boxes are components that are managed by the container. The components shown with square boxes are internal container components that are used to conjoin the ETL components. Please note that the ETL developers never see or have to deal with the internal components. We show these internal components for illustration purposes only. Every managed component has its own deployment descriptor for the configuration parameters. The ETL container controls and monitors the event processing by optimizing these configuration settings. As shown in Figure 2 the internal container components bind together the ETL processing steps without using intermediate storage. When new events arrive, the ETL container decides which ETL components are called and in what sequence these components are executed. In the next section, we describe in detail the mechanisms of the ETL container for managing the data processing flows.

5. Managing Continuous ETL Processing Flows

Traditional WFMS are suitable to control the execution of ETL batch-processes [3], [8]. However, when it comes to continuous data integration, a very large amount of ETL process instances will arise because the incoming events are processed individually and have to be handled by a separate flow. For instance, an order process with the magnitude of thousands of process instances can result in potentially millions of workflow events. Therefore, it is not feasible to use traditional workflow engines for managing millions of ETL processing flows. As an alternative, a solution is needed that is extremely lightweight and supports sufficient capabilities to control the ETL processing flows.

For the control of the processing flow, the ETL container uses ETLet triggers that define conditions for the execution of an ETLet. These conditions are checked at different points in time during the ETL processing. The current ETL container implementation supports the following triggering conditions: 1) event-IDs or XPath selectors, 2) ETLet processing outcome, 3) evaluation results, and 4) schedules. We will discuss these trigger mechanisms in detail in the following sections.

5.1. ETLet Triggers Based on Event-IDs or XPath Selectors

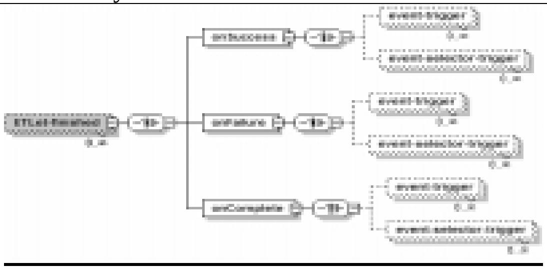
ETLets can be triggered if an incoming event has a particular *event-ID* or a *matching XPath expression*. The matching of incoming events with XPath expressions allows a content-based subscription to XML events (e.g. subscription to all events that include an order business object). Since event adapters transform all incoming raw events to XML events, the triggering XPath expressions can be used as a universal subscription mechanism which is more flexible than traditional queue topic subscription mechanisms. Figure 3 shows examples for the definition of ETLet triggers in the deployment descriptors that are based on event-IDs and XPath expressions.

```
<event-trigger event-id="ORDER_CHANGED"/>
<event-trigger event-id="ORDER_ACCEPTED"/>
<event-selector-trigger selector="//Order"/>
```

Fig. 3: Example - ETLet Triggers with Event-ID and XPath Selector

5.2. ETLet Triggers Based on the ETLet Processing Outcome

The container can trigger ETLets based on the execution outcome of another ETLets. This triggering mechanism can be used to construct processing flows (e.g. ETLet chaining). For instance, an ETLet successfully calculated the cycle time of a business process, another ETLet could be triggered which calculates the costs which are determined based on the before calculated cycle time.



Example:
```
  <ETLet-finished ETLet-name="previousETLet">
   <onComplete>
      <event-selector-trigger selector="//Order"/>
   </onComplete>
…. <ETLet-finished>
```

Fig. 4: Example - ETLet Triggers Based on the Processing Outcome

Figure 4 shows the structure of ETLet triggers in the deployment descriptor that are based on the processing outcome of other ETLets. The example in Figure 4 triggers an ETLet after the *completion* of the ETLet with the name “previousETLet” only if the current processed event includes an “Order” element.

5.3. ETLet Triggers Based on Evaluation Results

An ETLet can also be triggered based on the outcome of a metric evaluation. For instance, if a metric reaches a certain threshold it could be an indicator that additional data processing is required. Figure 5 shows an ETLet trigger that triggers if the evaluator “CycleTimeLimitEvaluator” returns ‘1’. A return value of ‘1’ indicates that the cycle time was exceeded.

```
 <evaluator-finished evaluator-name="CycleTimeLimitEvaluator">
    <onEvaluate result="1" />
</evaluator-finished>
```

Fig. 5: ETLet Trigger Based on an Evaluation Result

5.4. ETLet Triggers Based on a Schedule

An ETLet can have associated with it a triggering schedule. Between predefined start and the end timestamps, the scheduler triggers the associated ETLet. The frequency of triggering an ETLet is defined by a time interval. Figure 6 shows an example of an ETLet that is triggered every 10 minutes during the time period 2001-08-25 till 2002-08-25.

```
 <schedule-trigger>
   <period>
     <start>2001-08-25</start>
     <end>2002-08-25</end>
   </period>
   <interval>
      <minutes>10</minutes>
   </interval>
</schedule-trigger>
```

Fig. 6: ETLet Trigger Based on a Schedule

These four ETLet trigger types can be used in the ETL container to compose complex data processing flows. Figure 7 shows a summary of the ETL flow management capabilities available in the ETL container. Please note that the ETL container does not provide the same advanced flow management capabilities as those of workflow management systems. However, the flow management capabilities are very well integrated and support typical ETL processing scenarios. Moreover, the ETL flow management is very lightweight and thereby able to support a very large number of flows.

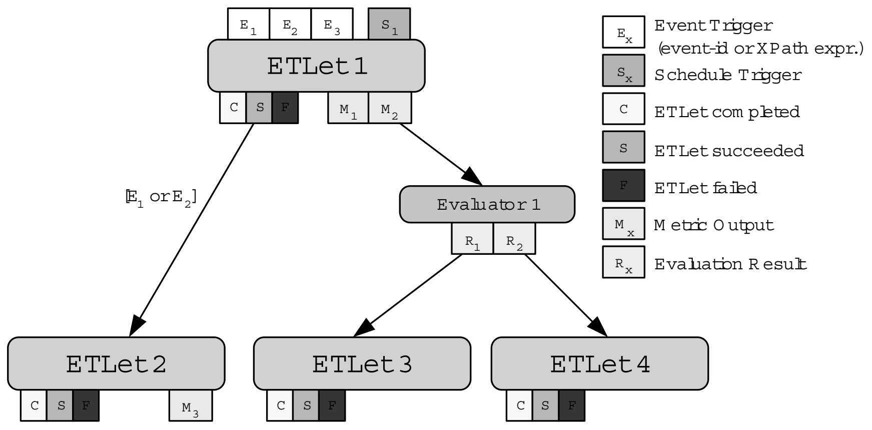


Fig. 7: Flow Management Capabilities

5.5. Serialization of Processing Flows

ETLets can subscribe to a number of *event types* and they are invoked in parallel for the processing. There are situations where the data processing of ETLets or entire processing flows have to be serialized. For instance, there are two incoming events that contain the same Order-ID. The first event places an order and the second event cancels the same order. Due to propagation delays, the first event arrives only one second before the second event. If these two events are processed in parallel by ETLets, there will be a high risk of data inconsistencies, because we need to preserve the serialization order also for the ETL processing. This situation is typically referred to as late-arriving data. The ETL container solves this problem by providing mechanisms for serializing the data processing flows. The serialization of processing flows is also configured within the deployment descriptor of an ETL solution. Therefore, it is possible to specify for example, that events of the event type "order" should be serialized, but only if they refer to the *same Order-ID* (the corresponding deployment descriptor section would be `<event-flow-serialization selector="//Order@ID"/>`). Therefore, the decision regarding whether parallelism should be used depends on the contents of incoming events.

Figure 8 shows an example for serializing the data processing needed to update the customer ratings. In the example we assume a database table in an operational data store that captures the current ratings for customers. Many external events might have an impact on these ratings (e.g. a customer cancelled many orders or did not pay). To ensure a consistent update of the customer rating, it is necessary to serialize the event processing for a customer rating. Please note that the ratings of customers are generally calculated in parallel (e.g. the rating for customer "Joe" and for customer "Alex"). However, all events that affect a particular customer (e.g. all events for the rating of customer "Joe") must be serialized.

The ETL container is able to solve the above discussed serialization problems. It uses XPath selector expressions for incoming events which are used to identify common attributes for the serialization. Selector expression can be applied globally to entire flows (as shown before with the order example) or locally to ETLets (as shown with the customer rating example). If the selector returns the same contents as a currently processed ETLet or flow, the ETL container waits with the processing of the new event. Figure 8 shows that the ETL container waits with the invocation of CustomerRating ETLet on thread 3 until the processing in “Thread 2” is finished.

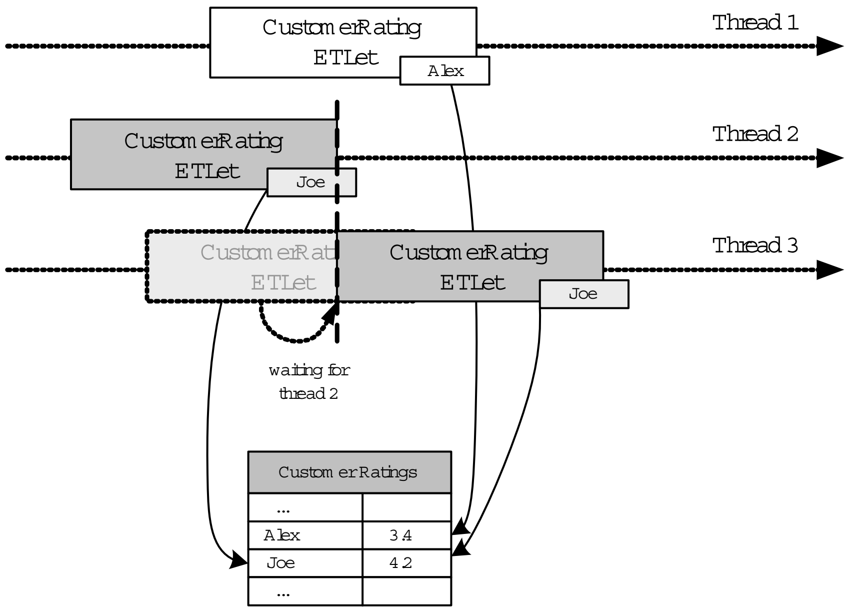


Fig. 8: Serialization of ETLets

6. Conclusion and Future Work

In large organizations, huge amount of data can be generated and consumed by business processes. Business managers need up-to-date information to make timely and sound business decisions. Conventional workflow management systems and decision support systems do not provide the low latencies needed for the decision making in e-business environments. This paper described a container-based approach with the aim of providing continuous, near real-time data integration for data warehouse environments. We use an ETL container for the data processing, which provides a robust, scalable and high-performance event processing environment. We introduced the flow management capabilities of the ETL container and discussed serialization issues as well.

The work presented in this paper is part of a larger, long-term research effort aiming to develop real-time data services for business process management platforms. In our future work, we want to extend our model to formally describe and execute data processing flows. Furthermore, we want to add new services to the ETL container, which are useful for the ETL processing. This includes a set of services for the data extraction and data transformation, including caching, concurrency, locking, and transformation engines.

References

1. B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom: *Models and Issues in Data Stream Systems*. In Proc. of 21st PODS, pp. 1–16, Madison, Wisconsin, May 2002.
2. N.S. Barghouti, B. Krishnamurthy: *Using event contexts and matching constraints to monitor software processes*. In Proceedings of the 17th International Conference on Software Engineering, IEEE Computer Society, April 1995.
3. M. Bouzeghoub, E. Fabret, M. Matulovic-Broque: *Modeling the Data Warehouse Refreshment Process as a Workflow Application.* In Proc. of DMDW'99, Heidelberg, Germany, 1999.
4. Q. Chen, M. Hsu, U. Dayal: *A Data-Warehouse / OLAP Framework for Scalable Telecommunication Tandem Traffic Analysis*. In Proc. of 16th ICDE, pp. 201–210, San Diego, CA, 2000.
5. B. Krishnamurthy, D.S. Rosenblum, Yeast: *A General Purpose Event-Action System*. IEEE Transactions on Software Engineering, October 1995.
6. J. Schiefer, J.J. Jeng, R.M. Bruckner: *Real-time Workflow Audit Data Integration into Data Warehouse Systems*. ECIS, Naples, 2003.
7. W. Selby, A. A. Porter, D.C. Schmidt, J. Berney: *Metric-Driven Analysis and Feedback Syststems for Enabling Empirically Guided Software Development*. In Proceedings of the 13th International Conference on Software Engineering, IEEE Computer Society, May 1991.
8. P. Vassiliadis, C. Quix, Y. Vassiliou, M. Jarke: *Data Warehouse Process Management*. Information Systems, Elsevier Science, Vol. 26(3): 205–236, 2001.
9. A.L. Wolf, D.S Rosenblum: *A Study in Software Process Data Capture and Analysis*. In Proceedings of the Second International Conference on the Software Process, IEEE Computer Society, Februrary 1993.
10. Y. Zhuge, H. Garcia-Molina, J.L. Wiener: *Consistency Algorithms for Multi-Source Warehouse View Maintenance*. Distributed and Parallel Databases, Kluwer, Vol. 6(1), pp. 7-40, 1998.