CORRELATING EVENTS FOR MONITORING BUSINESS PROCESSES

Josef Schiefer
IBM Watson Research Center, 19 Skyline Drive, Hawthorne, NY, 10606, US

Carolyn McGregor
Centre for Advanced Systems Engineering, University of Western Sydney, Locked Bag 1797, Penrith South DC, NSW, 1797, Australia

Keywords: Event Stream Processing, Event Correlation, Business Process Monitoring

Abstract: With the increasing demand for real-time information on critical performance indicators of business processes, the capturing, transformation and correlation of real-world events with minimal latency are a prerequisite for improving the speed and effectiveness of an organization's business operations. Events often include key business information about their relationship to other events that can be utilized to collect relevant event data for the calculation of business performance indicators. In this paper we introduce an approach for correlating events of business processes that uses correlation sessions to represent correlation knowledge. Correlation sessions facilitate the processing of data across multiple events and thereby enable a calculating of business metrics in near real-time. The benefit over existing approaches is that it is tailored to instrument business processes and business applications that may operate in a heterogeneous software environment. We propose a Java-based, container-managed environment which provides a distributed, scalable, near-real time processing of events and which includes a correlation service that effectively manages correlation sessions. We also show a complete example that illustrates how correlation sessions can be utilized for computing the cycle time of business processes.

1 INTRODUCTION

Operational business systems, such as enterprise resource planning systems or workflow management systems (WFMSs) are able to report business process state changes, such as the execution of business operations or the completion of customer requests, which can be utilized for monitoring and analysis purposes. Today’s e-business platforms suffer from inappropriate management of these events. The symptoms of exceptional business situations are frequently detected by many managed applications simultaneously causing them to generate events. Event correlation in the business domain is a technique for collecting and isolating related data from events so as to condense many events each with little information into a few meaningful business metrics. By automatically correlating events and calculating business metrics from the correlated event data, business managers and staff members can monitor these metrics and quickly respond to exceptional business situations.

Many business processes are executed in a distributed environment where a large number of events are generated and processed in different locations by various operational systems. When every state change of a business process generates a new event and an organization creates thousands of process instances daily, a very large number of events must be handled. For instance, an order process with thousands of process instances can result in potentially millions of state change events. Therefore, a scalable solution for processing and correlating a very large number of process events is required.

Figure 1 shows the process for continuously integrating events from various source systems. The processing steps are not equivalent to the traditional

Schiefer J. and McGregor C. (2004).
CORRELATING EVENTS FOR MONITORING BUSINESS PROCESSES.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 320-327
DOI: 10.5220/0002618403200327
Copyright © SciTePress

ETL (Extraction, Transformation, Loading) because the underlying assumptions for the data processing and the latency requirements are different. Traditional ETL tools often operate during a batch window and do not affect or disrupt active user sessions. If data is to be integrated continuously, a permanent stream of data must be integrated into the data warehouse environment while users are using it.

Figure 1 shows three stages for continuous event data integration: 1) Receive/Extraction, 2) Event Processing, and 3) Evaluation. A separate loading stage is not included because event data is usually not buffered and prepared for bulk loading. We extend the traditional data integration process with an evaluation stage that allows the system to respond to conditions or irregularities in the integrated event data. An evaluation of continuously calculated business metrics can be very valuable to the business

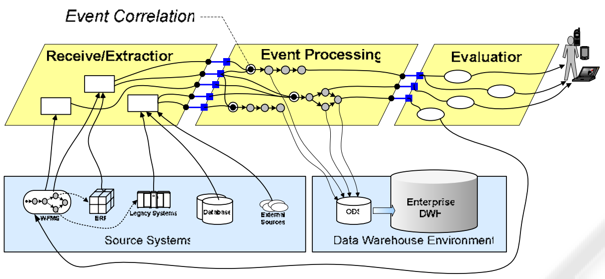

Figure 1: Continuous Event Data Integration

because it promotes intelligent responses to business operations and proactive notification mechanisms in near real-time.

Dependent on the external interface capabilities of the source systems, events are transmitted either by push (e.g. messaging) or pull mechanisms (e.g. J2EE connector, web-services). For the processing of events, the following two challenges have to be addressed in combination:

Online processing. Events are processed as soon as they stream into the system. Therefore, 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. The processing can include any type of data transformation, data cleansing, calculation or evaluation 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 accelerates the data processing by moving data between the different processing steps without any intermediate file or database storage.

Event correlation. Many business metrics require a set of related events for its calculation. The related events often stream into the system at different points in time. To deliver metrics with minimal delay to business users, a mechanism to continuously gather related event data is needed that allows metric calculation as soon as sufficient event data is available. A simple event correlation example is the calculation of business activity processing times where event pairs of when an activity started and finished have to be collected. As soon as a business activity completes, the processing time can be calculated. This example requires a mechanism for holding event data for a certain time interval.

Business process event correlation is challenging because:

– Business process events can occur in various source systems producing events in different formats that must be captured with minimal latency and minimal operational system impact.

– Late arriving events due to network failures or downtimes of operational systems within heterogeneous and distributed software environments can be common situations that have to be considered for the event correlation.

– Only relevant event data for applications and users must be unified, transformed, and cleansed before they are correlated. In many situations, only a small set of selected event attributes are needed for the correlation.

– Correlated events can occur at different points in time requiring the temporary storage of correlated event data that is transparent for applications.

– Correlated business process events can be analyzed and processed by multiple components to calculate business metrics or to discover irregularities in the business process.

McGregor and Schiefer (2003) introduce a Solution Management Web Service (SMWS) architecture that supports near real-time integration of event data to provide involved parties and decision makers with comprehensive information about the status and performance of business processes independent from the type of systems that are used to execute the business process.

This paper, further defines the EPC processing by defining an approach for correlating events that overcomes the challenges detailed above. This approach can be summarized as follows: We propose a *correlation session* to gather correlated event data that exists temporarily and have a lifecycle. Before correlating events, event data often has to be unified, transformed, cleaned or condensed. Correlation sessions reduce the overall amount of event data and condense large correlated data sets that are only partially used for the processing, reducing the resources required for holding correlated data in memory or on disk. We use a container-managed environment that automatically manages correlation sessions which enables scalability and significantly decreases the effort for developers to utilize correlation sessions.

The remainder of this paper is organized as follows. In section 2, we give an overview of approaches for event correlation developed in the research community and the industry so far and discuss our contribution. In section 3, we introduce and define correlation sessions. In section 4, we present an architecture for a container-based environment which allows to integrate and correlate events from various sources and which automatically manages the lifecycle of correlation sessions. Section 5 shows an example of an application that utilizes correlation sessions for continuously calculating the cycle time of business processes. Finally, section 6 concludes the paper.

2 THE ROLE OF CORRELATION FOR EVENT MANAGEMENT

The emergence of e-business has forced many organizations to improve operational efficiency, turning their attention toward real-time business activity monitoring (BAM) solutions. These initiatives require enterprise data to be captured and analyzed in real-time from a wide variety of business applications, operational data stores and data warehouses. While traditional data integration approaches, built on top of core ETL (extraction, transformation, loading) solutions are well suited for building historical data warehouses for strategic decision support initiatives, they do not go far enough toward handling the challenge of continuously integrating data with minimal latency and implementing a closed loop for business processes. Traditional solutions are optimized for batch-oriented data integration and make the assumption that large data sets can be extracted from various source systems in order to transform and integrate them into a data warehouse environment. The correlation of data for these scenarios is a minor problem since the data integration tools are always able to access the entire data sets.

However, when it comes to continuous data integration, where events are propagated and processed continuously from various source systems, event data has to be correlated in order to be able to generate business metrics. One single event includes very little information about the business process and is therefore too detailed for monitoring purposes. What is needed is more granular business information in the form of representative business metrics that are derived from a set of raw events. In order to be able to transform raw events into valuable business metrics a correlation mechanism is needed that enables the capture of required event data for calculating a single business metric. The events of business processes often have inherent dependencies that have to be considered during the event processing. For example, an order process might include processing steps whose processing times are of interest to the business. By correlating the events that indicate when process activities started and completed, a calculation of the processing times becomes very straightforward.

Detecting and handling exceptional events also plays a central role in network management (Feldkuhn and Erickson, 1989)0. Alarms indicate exceptional states or behaviors, for example, component failures, congestion, errors, or intrusion attempts. Often, a single problem will be manifested through a large number of alarms. These alarms must be correlated to pinpoint their causes so that problems can be addressed effectively. Many existing approaches for correlating events have been developed from network management. Event correlation tools help to condense many events, which individually hold little information, to a few meaningful composite events.

Rule-based analysis is a traditional approach to event correlation with rules in the “conclusion if condition” form which are used to match incoming events often via an inference engine. Based on the results of each test, and the combination of events in the system, the rule-processing engine analyzes data until it reaches a final state (Wu et al., 1989).

Another group of approaches incorporate an explicit representation of the structure and function of the system being diagnosed, providing information about dependencies of components in the network (Katzela and Schwartz, 1995) or about cause-effect relationships between network events. The fault discovery process explores the network model to verify correlation between events.

NetFACT (Houck et al., 1995) uses an object-oriented model to describe the connectivity, dependency and containment relationships among network elements. Events are correlated based on these relationships. Nygate (1995) models the cause-effect relationships among events with correlation tree skeletons that are used for the correlation.

InCharge (Yemini et al., 1996) represents the causal relationships among events with a causality graph using a codebook approach to quickly correlate events to their root causes. The code-book approach uses a network model to derive a code – a set of possible symptom observations for every problem that may occur in the network – performed in advance, eliminating the runtime computational complexity involved in model traversals and is therefore more scalable than rule-based systems.

The Streams project (Babu and Widom, 2001) attempts to provide complete DBMS functionality along with support for continuous queries over streaming data. Thereby, continuous data streams are queried by using SQL like statements to select, group and process data stream elements. While this approach can be used for a wide range of continuous queries, the processing of data has to be specified in a SELECT statement limiting developers to implement user-defined functions for the event processing because it is limited to the functionality provided in SQL.

In spite of intensive research in the past years for correlating events in the network management domain, there are to our knowledge no existing approaches addressing the issues for correlating business process events. In this paper, we propose a container-based architecture to integrate business process events in near real-time. We introduce the concept of correlation sessions for collecting related event data, and show how a container environment can be used to effectively manage their lifecycle.

3 CORRELATING EVENTS WITH SESSIONS

The “instant” of an event is relative to the time granularity that is needed or desired. Thus, certain activities that are of short duration relative to the time granularity are represented as a single event. An activity spanning some significant period of time is represented by the interval between two or more events. For example, an order review might have a “begin-review” and “end-review” event-pair. Similarly, a workitem in a workitem list could be represented by the three events “enter-worklist” “start-processing” “end-processing”.

For purposes of maintaining information about an action, events can also capture attributes about the context when the event occurred. Event attributes are items such as the agents, resources, and data associated with an event, the tangible result of an action (e.g., the result of an approval decision), or any other information that gives character to the specific occurrence of that type of event.

Elements of an event context can be used to define a relationship with other events. This relationship can be expressed by a correlation expression that is used to create a container for a set of correlated event data. We label the container for holding this set of correlated event data as *correlation session*. In other words, a correlation session is a container with a set of data items that exists for each relationship between events.

Therefore, we define a correlation session CS as a triple of the form (O, X, A) where O defines the owner, X the correlation expression and A the correlating event attributes for the session. For a given event stream a correlation session is created for each unique set of event attributes A = { A_1, … A_n } that is extracted by a correlation expression X during the event arrival. X is an expression that selects attributes from incoming events and uses these attributes as key for the correlation sessions. A correlation session is a container for a set of data items D = {D_1, …, D_n} which can originate from event attributes or from other sources. Every correlation session has an owner O which is a component that is allowed to utilize the session. This component can be called in multiple threads in parallel and each thread can read or updated the correlation session.

Figure 2 shows an event stream with events that include an attribute A_1. Let’s assume that we use the attribute A_1 for correlating the events. In this case, we will create a separate correlation session for each different value of the attribute A_1. For instance, in Figure 2 *Session 3* is correlated with all events that have an attribute A_1 with the value 4.

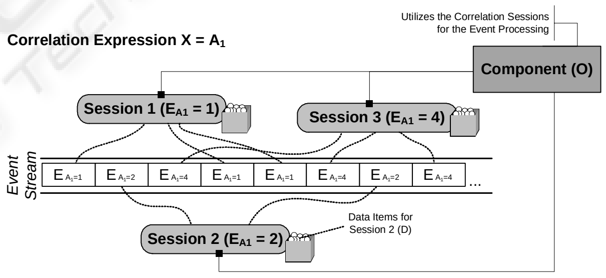

Figure 2: Correlation Sessions

The data items of correlation sessions can include entire events, extracted event attributes, or also external data items that are required for the event processing. For instance, for the calculation of metrics about a business process, sessions can be utilized to store event data of a business process instances that are uniquely identified by a process instance IDs. For each process instance ID, a separated correlation session can be created during the event processing to gather runtime data about the business process instance. The data items in the correlation sessions might include event attributes or also external data that is needed for calculating a business metric. In section 5, we will show an example with correlation sessions for computing the cycle time of a business process.

4 MANAGING CORRELATION SESSIONS WITH A CONTAINER

In this section, we discuss a container-based approach for managing the lifecycle of correlation sessions. The usage of a container decreases the need for programming and programmer training by creating standardized, reusable modular components and by enabling the container to handle many aspects of programming automatically. 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. Similar to traditional J2EE containers, we use a customized container for the event processing. In our approach, we extend this concept by adding container services, which are useful for developers that write programs that process events. A *container service* is responsible for the monitoring of the data extracts and ensures that resources, workload and time-constraints are optimized. Also, the management of correlation sessions is implemented as a container service that can be utilized for event data integration solutions. Developers are able to specify configuration information for correlation sessions (e.g. correlation expressions, lifetime of session data, parameters for the persistence of session data) in a deployment descriptor and the container will use and try to optimize these settings.

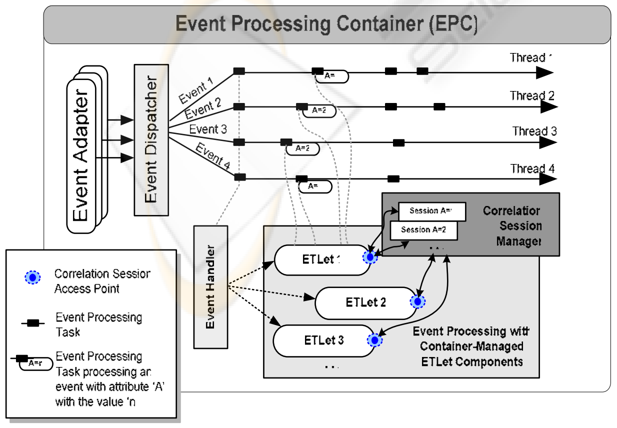
Event Processing Container (EPC)
Event Adapter
Event Dispatcher
Event 1
Event 2
Event 3
Event 4
Thread 1
Thread 2
Thread 3
Thread 4
A=
A=2
A=2
A=
Session A=
Session A=2
Correlation Session Manager
Event Handler
ETLet 1
ETLet 2
ETLet 3
Event Processing with Container-Managed ETLet Components
Correlation Session Access Point
Event Processing Task
A=n
Event Processing Task processing an event with attribute 'A' with the value 'n

Figure 3: Container-Managed Correlation Sessions.

Figure 3 shows the internal components of an Event Processing Container (EPC). The components shown with round boxes (event adapters and ETLets) are user-defined components that are managed by the container and have to be implemented by developers. Event adapters are used to receive event data from various source systems, and they unify the data in a standard XML format. ETLet components use the extracted XML event data as input and perform the processing tasks, such as the calculation of business metrics and storing them in a database.

The container uses the Correlation Session Manager (CSM) to manage the lifecycle of correlation sessions. The CSM uses access points to ETLet components in order to assign the appropriate session during the event processing in runtime. Similar to a J2EE web container, where user sessions are used to maintain state information in the course of performing a user task by returning the correct user session instance in web components (e.g. servlets or JSPs), the CSM returns the correct correlation sessions to ETLet components. The CSM uses correlation expressions in XPath defined in a deployment descriptor in order to determine events that should be correlated. The tasks of the CSM can be summarized as follows:

Construction and Initialization. The CSM creates and initializes a new correlation session for every set of related events. A correlation expression in XPath is used for associating events and to determine whether a new correlation session should be created, or whether an existing correlation session should be reused.

Correlation Sessions Access. After correlation sessions have been created, the CSM uses access points to give ETLet components access to the sessions. In our current implementation, we expose a method called getSession() in the base class of ETLet components that determines in runtime the correct correlation session during the event processing.

Session Data Persistence. The CSM supports the following modes to make the correlation sessions persistent: 1) memory mode – correlation sessions are only kept in memory, 2) file mode – correlation sessions are stored in the file system, 3) database mode – correlation sessions are stored in a database, and 4) replication mode – correlation sessions are replicated among multiple containers in a server farm. The persistence mode is configurable in a deployment descriptor and the container automatically takes care of storing and retrieving session data.

Isolation for Session Access. During the event processing, ETLet components can read and write to a correlation session. In order to prevent side effects of concurrently executing events that use the same correlation session, the CSM ensures appropriate isolation independent of which persistency mode was chosen. The current implementation supports two isolation levels: 1) serialized (default) and 2) read uncommitted. Some solutions require concurrent, long-running event data processing where the isolation can significantly decrease the throughput of events due to the serialization of the session data access. A read uncommitted isolation level, allows ETLet components to take responsibility for handling session data access concurrency issues.

Destruction. The CSM also takes care of the destruction of correlation sessions either: 1) during the event processing in ETLets by calling the method getSession().invalidate(), or 2) by specifying a timeout parameter in the deployment descriptor and the CSM will automatically remove the session after a timeout.

5 EVENT PROCESSING EXAMPLE

In this section, we show a simple example of an application for calculating the cycle time of business processes with correlation sessions. Figure 4 shows a container environment that receives business process events from two sources. The first source is a WFMS that notifies the container with messages about state changes of workflows. The second event source is an ERP system that uses a J2EE connector for transmitting business process events. The event adapter components are used to unify the event formats from the two event sources and to dispatch standardized XML events. The container delivers the dispatched events to the CycleTime ETLet which calculates the cycle times with the support of correlation sessions. Finally, the ETLet dispatches the cycle time metrics which allows evaluator components to evaluate the metric and generate an intelligent response (e.g. sending out notifications to business people or triggering business operations) in near real-time.

The ETLet in our example calculates the cycle times by processing the PROCESS_STARTED, PROCESS_COMPLETED and PROCESS_CANCELED business process events. Figure 5 shows two examples for these events. The ETLet implementation shown in Figure 6 has a method called processEvent() that does the processing of the previously mentioned events. The processEvent() method calculates the cycle time by extracting the timestamp information from the events and determining the time differences. The timestamp of the PROCESS_STARTED event is stored in a correlation session in order to be retrieved at a later point in time. In other words, the correlation sessions in this example capture all PROCESS_STARTED timestamps of currently running process instances.

When the PROCESS_COMPLETED or PROCESS_CANCELED event is received, the cycle time of a process instance is calculated. The calculated metric is also published which allows other managed components of the container to receive this metric (e.g. components that evaluate the metric). Since the data items in the correlation session aren't needed for other upcoming events, it is finally destroyed by calling the method getSession().invalidate().

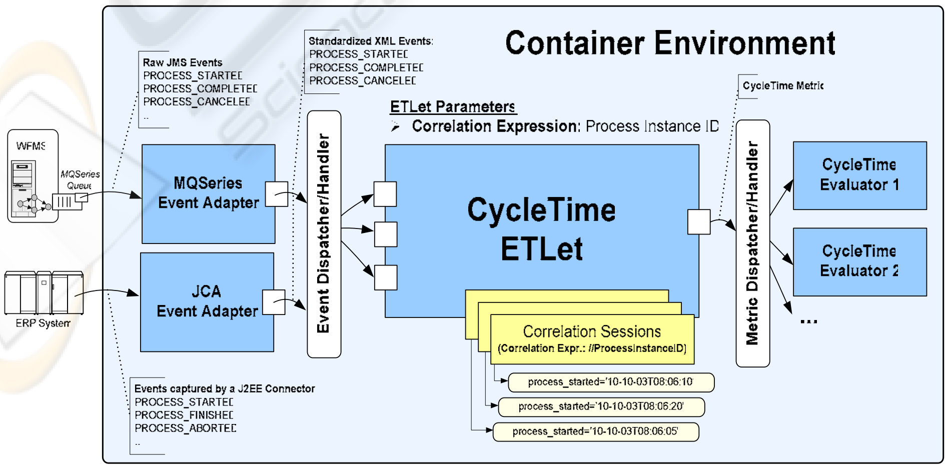

Figure 4: Example - Processing Events for Calculating Cycle Times

```
<event>
    <event-ID>PROCESS_STARTED</event-ID>
    <time-created>2003-01-25T02:33:00</time-created>
    <event-source>
        <source-ID>MQSeries WF</source-ID>
    </event-source>
    <priority>2</priority>
    <business-process-event>
        <process-ID>Order Process</process-ID>
        <process-instance-ID>221</process-instance-ID>
    </business-process-event>
</event>
```

```
<event>
    <event-ID>PROCESS_COMPLETED</event-ID>
    <time-created>2003-01-27T05:03:00</time-created>
    <event-source>
        <source-ID>MQSeries WF</source-ID>
    </event-source>
    <priority>2</priority>
    <business-process-event>
        <process-ID>Order Process</process-ID>
        <process-instance-ID>221</process-instance-ID>
    </business-process-event>
</event>
```

Figure 5: PROCESS_STARTED and PROCESS_COMPLETED Events

```
public class CycleTimeETLet extends ETLet {

  // Event processing for the events PROCESS_STARTED and PROCESS_COMPLETED
  public void processEvent(WorkflowEvent event, MetricPublisher metricPublisher) {
    if(event.getEventID().equals("PROCESS_STARTED")) {
       getSession().put("ProcessStarted",event.getTimestamp());
    }
    if(event.getEventID().equals("PROCESS_COMPLETED") ||
       event.getEventID().equals("PROCESS_CANCELED")) {
      Date started = (Date)getSession().get("ProcessStarted");
      Date done = event.getTimestamp();

      // Calculate the cycle time in seconds
      long cycleTime = (done.getTime() - started.getTime())/1000;

      // Store the cycle time in the database
      ... store cycle time in database

      // Publish the cycle time metric
      metricPublisher.publish("CycleTime", new Long(cycleTime), null);

      // Cleanup
      getSession().invalidate();
    }
  }

  // Handles immediately exceptions
  public boolean failed(WorkflowEvent event, Exception exception) {
      // Container will automatically store events and exceptions in an error table
      return false;
  }
}
```

Figure 6: ETLet Example.

Figure 7 shows the sections in the deployment descriptor for the CycleTime ETLet. The ETLet section carries information about the ETLet name, the implementation class, the triggering events, the published metrics, and the correlation sessions. The correlation session configuration includes the XPath expression "//process-instance-ID" which is used by the container to extract data that is used for the correlation of incoming events (see also Figure 5 which shows the events that are the XPath expression applied to). The extracted correlation data is used by the container to create and associate correlation sessions. With the correlation expression shown in Figure 7, a new correlation session is created for each new process instance. Furthermore, the configuration for correlation section includes parameters about the isolation level, the session timeout (in minutes), and the persistent storage for the session data. In this example, the persistent storage for session data is crucial since processes can have a long cycle time and the container might loose the session data due to machine crashes or restarts.

The container uses the information shown in Figure 7 for instantiating and initializing the CycleTime ETLet component. Please note that the deployment descriptor includes other parameters that are not shown in the figure (e.g. database configuration for the error tables, global parameters, control flow for the event processing, etc.).

```
...
<!-- ETLet section -->
<ETLet>
  <name>CycleTimeETLet</name>
  <impl-class>CycleTimeETLet</impl-class>
  <ETLet-triggers>
      <event-trigger event-id="PROCESS_STARTED"/>
      <event-trigger event-id="PROCESS_COMPLETED"/>
      <event-trigger event-id="PROCESS_CANCELED"/>
  </ETLet-triggers>
  <published-metrics>
     <metric-name>CycleTime</metric-name>
   </published-metrics>
   <correlation-session-config>
      <correlation-expression>//process-instance-ID</correlation-expression>
      <isolation-level>serialized</isolation-level>
      <session-timeout>43200</session-timeout>
      <persistent-store>
          <persistent-store-database>
              <jdbc-connection-string>jdbc:db2:csmdb</jdbc-connection-string>
              <database-driver>COM.ibm.db2.jdbc.app.DB2Driver</database-driver>
              <user>dbuser</user>
              <password>dbpassword</password>
          </persistent-store-database>
       </persistent-store>
    </correlation-session-config>
</ETLet>
...
```

Figure 7: Deployment Descriptor Example.

6 CONCLUSION

In large organizations, huge amounts of data can be generated and consumed by business processes. Business managers need up-to-date information to make timely and sound business decisions. This paper described a container-based approach for correlating event data with the aim of providing continuous, near real-time data integration for data warehouse environments. We introduced the concept of managed correlation sessions to capture related event data and to facilitate its data management. Correlation sessions provide direct access to correlated event data and leave the component developer with the simplified task of developing business functionality. Our container-based approach for managing correlation sessions allows a reconfiguration without changing the component code, making components useful in a wide range of contexts.

REFERENCES

Babu, S. and Widom, J., Continuous queries over data streams, ACM SIGMOD Record, 30(3):109–120, Sept. 2001.

Feldkuhn, L. and Erickson, J., Event Management as a Common Functional Area of Open Systems Management, in Proc. IFIP Symposium on Integrated Network Management, North Holland, 1989.

Houck, K., Calo, S., Finkel, A., Towards a practical alarm correlation system, IEEE/IFIP Symposium on Integrated Network Management, 1995.

Katzela, I. and Schwartz, M., Schemes for fault identification in communication networks, IEEE Transactions on Networking, 1995.

McGregor, C. and Schiefer, J., A Framework for Analyzing and Measuring Business Performance with Web Services, Proceedings of the 2003 IEEE Conference on E-Commerce, CEC'03, Newport Beach, CA, 2003.

Nygate, Y.A., Event correlation using rule and object base techniques, IEEE/IFIP Symposium on Integrated Network Management, 1995.

Wu, P., Bhatnagar, R., Epshtein, L., Bhandaru, M., Shi, Z., Alarm correlation engine (ACE), In Proceedings of the IEEE/IFIP 1998 Network Operations and Management Symposium (NOMS), New Orleans, 1998.

Yemini, S. A., Sliger, S., Eyal, M., Yemini, Y., Ohsie, D., High Speed and Robust Event Correlation, IEEE Communications Magazine 34, No. 5, 82–90, 1996.