Complex Event Processing in the context of Business
Activity Monitoring

An evaluation of different approaches and tools taking the example of the Next Generation easyCredit

Diploma Thesis according to § 12 APO and § 31 of RaPO
of the University of Applied Sciences Regensburg

Presented by:

Hans-Martin Brandl
Nawiaskystrasse 15
81735 München

AND

David Guschakowski
Am Hohen Kreuz 37
93426 Roding

Advisors:
Dr. Rainer v. Ammon

Referees:
Prof. Dr. Fritz Jobst
Dr. Rainer v. Ammon

Handed in on:
31.05.2007
Erklärung

Mir ist bekannt, dass dieses Exemplar der Diplomarbeit als Prüfungsleistung in das Eigentum des Freistaates Bayern übergeht.

Ich erkläre hiermit, dass ich die Kapitel 1, 2, 3, 3.3, 4, 6, 7.3, 7.4, und 9 dieser Diplomarbeit selbständig verfasst, noch nicht anderweitig für andere Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benützt sowie wörtlich und sinngemäß Zitate als solche gekennzeichnet habe.

Regensburg, 31. Mai 2007

Hans-Martin Brandl

Mir ist bekannt, dass dieses Exemplar der Diplomarbeit als Prüfungsleistung in das Eigentum des Freistaates Bayern übergeht.

Ich erkläre hiermit, dass ich die Kapitel 5, 3.3, 7.1, 7.2, 8 und 9 dieser Diplomarbeit selbständig verfasst, noch nicht anderweitig für andere Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benützt sowie wörtlich und sinngemäß Zitate als solche gekennzeichnet habe.

Regensburg, 31. Mai 2007

David Guschakowski
Assertion

1. It is well-known by the authors that this copy of the thesis (diploma) changes as test achievement into the property of the Free State of Bavaria.

2. I explain that I independently wrote this thesis (diploma) and have not yet used it for other test purposes. I did not use any different than the indicated sources and aids as well as literally and corresponding quotations than the marked.

Regensburg, 31. Mai 2007

................................. ........................................
Hans-Martin Brandl          David Guschakowski

Partition
The particular chapters and paragraphs are written by the authors as follows:
Hans-Martin Brandl: 1, 2, 3, 3.3, 4, 6, 7.3, 7.4, 9
David Guschakowski: 3.3, 5, 7.1, 7.2, 8, 9
Outline

1 Introduction and purpose ........................................................................................................... 1
   1.1 Motivation and relevance .................................................................................................. 1
   1.2 Content ......................................................................................................................... 2

2 Basics ....................................................................................................................................... 3
   2.1 Enterprise Java beans ................................................................................................... 3
   2.2 JBoss application server ............................................................................................... 8
   2.3 Java Messaging Service ............................................................................................... 10
   2.4 Interceptors .................................................................................................................. 13

3 The state-of-the-art of BPM/BAM/CEP/SOA ...................................................................... 17
   3.1 Complex Event Processing ............................................................................................ 17
   3.2 Business Process Management ..................................................................................... 28
   3.3 Business Activity Monitoring ....................................................................................... 36
   3.4 Service Oriented Architecture ..................................................................................... 40

4 The easyCredit-Internet application ....................................................................................... 44
   4.1 The use case ................................................................................................................ 44
   4.2 The as-is solution of easyCredit and the pipeline model .............................................. 46
   4.3 The challenges for the Next Generation easyCredit system ........................................ 47
   4.4 Traditional OLAP-based views versus future CEP-based views ................................ 50

5 BAM-views and event patterns .............................................................................................. 53
   5.1 Requirements for the integration of a CEP platform into an enterprise environment .... 53
   5.2 BAM-view definition ................................................................................................... 58
   5.3 Event design ................................................................................................................ 72
   5.4 Rule design ................................................................................................................... 78
   5.5 Conclusion .................................................................................................................... 87

6 Implementation of the event generators by EJB interceptors and servlet filters .......... 90
   6.1 Implementation environment and TeamBank´s server architecture ......................... 91
   6.2 Deployment .................................................................................................................. 92
   6.3 Conclusion .................................................................................................................... 105
7 Implementation of the enterprise cockpit and the BAM-views ......................... 107
   7.1 Coral8 .................................................................................................................. 107
   7.2 StreamBase ...................................................................................................... 120
   7.3 TIBCO ............................................................................................................. 134
   7.4 Systar / Claresco ............................................................................................ 145
   7.5 Java Dashboard .............................................................................................. 158
8 Event Processing Languages.................................................................................. 164
   8.1 “Pseudo” SQL approaches................................................................................ 164
   8.2 Special rule languages or script languages..................................................... 166
   8.3 Java .................................................................................................................. 168
   8.4 XML ................................................................................................................ 169
   8.5 Graphical User Interface ................................................................................ 169
   8.6 Summary and evaluation of the results .......................................................... 170
9 Summary .............................................................................................................. 173
   9.1 Principle design of future CEP/BAM/BPM platforms ................................... 173
   9.2 Conclusions of the event generation concepts ............................................. 174
   9.3 Summary of the evaluated platforms ............................................................ 175
Acknowledgement and devotion ............................................................................. 176
List of literature ...................................................................................................... 177
List of figures .......................................................................................................... 196
List of tables .......................................................................................................... 201
List of abbreviations ............................................................................................... 203
Appendix A – Common Base Event Schema [IBM 2003a] .................................. 205
Appendix B – Reference model „loss pattern“ [Silberbauer 2007] ....................... 221
Outline

1 Introduction and purpose ........................................................................................................... 1
  1.1 Motivation and relevance ......................................................................................................... 1
  1.2 Content ................................................................................................................................ 2

2 Basics ...................................................................................................................................... 3
  2.1 Enterprise Java beans .............................................................................................................. 3
    2.1.1 Overview ......................................................................................................................... 3
    2.1.2 Roles ............................................................................................................................... 5
    2.1.3 Session beans .................................................................................................................. 7
    2.1.4 Message driven beans ...................................................................................................... 8
  2.2 JBoss application server .......................................................................................................... 8
    2.2.1 Introduction ....................................................................................................................... 8
    2.2.2 The JBossAS architecture ............................................................................................... 9
    2.2.3 EJBs on JBossAS .............................................................................................................. 10
  2.3 Java Messaging Service .......................................................................................................... 10
    2.3.1 Point-to-point (Queues) ................................................................................................ 11
    2.3.2 Publish and subscribe .................................................................................................... 12
    2.3.3 Conclusions .................................................................................................................. 13
  2.4 Interceptors ........................................................................................................................... 13
    2.4.1 Basics .............................................................................................................................. 14
    2.4.2 Interceptors in EJB2.0 ................................................................................................... 15
    2.4.3 Future interceptor EJB 3.0 adoption .............................................................................. 16
    2.4.4 Conclusion .................................................................................................................... 16

3 The state-of-the-art of BPM/BAM/CEP/SOA ................................................................. 17
  3.1 Complex Event Processing .................................................................................................... 17
    3.1.1 The architecture of CEP .................................................................................................. 18
      3.1.1.1 Low-level events ....................................................................................................... 18
      3.1.1.2 Complex events ........................................................................................................ 18
      3.1.1.3 Event patterns .......................................................................................................... 19
    3.1.2 Event Cloud ..................................................................................................................... 22
    3.1.3 IT-blindness .................................................................................................................... 24
    3.1.4 IT-insight ......................................................................................................................... 25
    3.1.5 Predictive Business ......................................................................................................... 26
    3.1.6 Conclusions and recommendations – the “event tornado” ......................................... 27
  3.2 Business Process Management ............................................................................................... 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Business Process Modeling</td>
<td>30</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Business Process Execution</td>
<td>33</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Business Process Monitoring</td>
<td>34</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Conclusions</td>
<td>36</td>
</tr>
<tr>
<td>3.3</td>
<td>Business Activity Monitoring</td>
<td>36</td>
</tr>
<tr>
<td>3.4</td>
<td>Service Oriented Architecture</td>
<td>40</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Event Driven Architecture</td>
<td>42</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Recommendations</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>The easyCredit-Internet application</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>The use case</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>The as-is solution of easyCredit and the pipeline model</td>
<td>46</td>
</tr>
<tr>
<td>4.2.1</td>
<td>The pipeline model</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>The challenges for the Next Generation easyCredit system</td>
<td>47</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Business Process Management challenge</td>
<td>47</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Multichanneling challenge</td>
<td>48</td>
</tr>
<tr>
<td>4.3.3</td>
<td>The challenge of Service Oriented Architecture / Event Driven Architecture</td>
<td>49</td>
</tr>
<tr>
<td>4.4</td>
<td>Traditional OLAP-based views versus future CEP-based views</td>
<td>50</td>
</tr>
<tr>
<td>4.4.1</td>
<td>OLAP-based views</td>
<td>50</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Differences to CEP-based views</td>
<td>51</td>
</tr>
<tr>
<td>5</td>
<td>BAM-views and event patterns</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Requirements for the integration of a CEP platform into an enterprise environment</td>
<td>53</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Web Service Distributed Management</td>
<td>54</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Common Base Event</td>
<td>56</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Conclusion</td>
<td>58</td>
</tr>
<tr>
<td>5.2</td>
<td>BAM-view definition</td>
<td>58</td>
</tr>
<tr>
<td>5.2.1</td>
<td>BAM-view: “Loss by canceled credit applications”</td>
<td>58</td>
</tr>
<tr>
<td>5.2.1.1</td>
<td>Enterprise cockpit solutions</td>
<td>59</td>
</tr>
<tr>
<td>5.2.1.2</td>
<td>Alerts</td>
<td>65</td>
</tr>
<tr>
<td>5.2.1.3</td>
<td>Appropriate compensations</td>
<td>66</td>
</tr>
<tr>
<td>5.2.2</td>
<td>BAM-view: “Performance monitoring of a service”</td>
<td>67</td>
</tr>
<tr>
<td>5.2.2.1</td>
<td>Enterprise cockpit solutions</td>
<td>68</td>
</tr>
<tr>
<td>5.2.2.2</td>
<td>Alerts</td>
<td>70</td>
</tr>
<tr>
<td>5.2.2.3</td>
<td>Appropriate compensations</td>
<td>71</td>
</tr>
<tr>
<td>5.3</td>
<td>Event design</td>
<td>72</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Determination of needed information in an event to build the views</td>
<td>72</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Mapping of information into the event format</td>
<td>73</td>
</tr>
</tbody>
</table>
9.3 Summary of the evaluated platforms ................................................................. 175

Acknowledgement and devotion ............................................................................. 176

List of literature ........................................................................................................ 177

List of figures ............................................................................................................ 196

List of tables ............................................................................................................. 201

List of abbreviations ............................................................................................... 203

Appendix A – Common Base Event Schema [IBM 2003a] ...................................... 205

Appendix B – Reference model „loss pattern“ [Silberbauer 2007] ............................ 221
1 Introduction and purpose

The following paragraphs shall introduce the purpose of this thesis. It is constituted how the thesis is set up and what intensions and requirements are achieved.

1.1 Motivation and relevance

To realize the business strategies, define business goals and constitute business structures of enterprises, business processes are applied as a save, concise and economic way. Coevally business processes also provide the operational departments of enterprises the same view on the business strategies as IT departments, which have the ability to optimize and support the performance of the business processes. In this thesis the online easyCredit-Internet business process of the TeamBank Company is monitored, applying the concepts of real-time Business Activity Monitoring (BAM) based on Complex Event Processing (CEP) conceptions. Therefore multiple tools and approaches are evaluated in this thesis with special focus on the real-time monitoring of canceled credit applications, monitoring of performance and status of the deployed components implementing the business logic as well as the frontend components based on metrics and key performance indicators which are visualized via dashboards. This thesis is expressively not intended as a comparative evaluation of the tested CEP platforms, whereby the functionality on base of the use case of the TeamBank Company is evaluated. Therefore the necessary functionalities as well as a sufficient stage of maturation of the language constructs and scopes of the underlying Event Processing Languages (EPL) of the current CEP platforms are exhibited. Furthermore multiple additional options are being argued for future implementations.
1.2 Content

In the following chapters the essential basics utilized for the implementation for the realization of this CEP-based BAM of the easyCredit-Internet application are introduced and discussed. Followed by the description of the basics technologies and concepts used to apply the concepts of CEP-based BAM in chap. 2, the principle concepts and purposes of the state-of-the-art paradigms of BAM, CEP and Business Process Management (BPM) are explained in detail in chap. 3 with regards to their utilization for the next generation easyCredit business process use case.

Chap. 4 describes the implemented monitoring system of the pipeline model, the use case and the challenges of CEP/BPM/SOA and multichanneling in detail while chap. 5 explains the design of the BAM-views, the definition of event rules and the structure of the processed events itself. Chap. 6 gives attention to the implementation of the event generation under reserve not to modify any already established components, business processes or architectures. Followed by that, chap. 7 focuses on the implementation of the enterprise cockpit and the applied BAM-views resp. the evaluation of the CEP-based platforms Coral8, StreamBase, TIBCO and Systar.

Chap. 8 describes the event processing languages (EPLs) used for modeling event patterns, describing situations and designing CEP-based applications. Finally chap. 9 gives an overview over the evaluated CEP-based platforms by means of their usability for this use case, followed by chap. 10 which mentions all persons and companies involved for enabling these realizations.

Main goal of this master thesis is the evaluation and realization of the use-case provided by the TeamBank Company. This is primarily a test of functionality but for example no performance-test – which should have been defined as benchmark then –, high availability-test, usability-test and so on.
2 Basics

The aim of this chapter is to show the principal techniques and standards for implementing, deploying and utilizing the appropriate possibilities provided by the JBoss application server cross-platform [RedHat 2007], Aspect Oriented Programming (AOP) [Burke 2003] concepts and the Java Enterprise Edition 5 (Java EE 5) [Sun 2006a]. It is dealt only with bases of the Java EE 5-technologies which were needed to orchestrate the easyCredit-application in an adequate way; to generate the necessary events for the implementation of concepts like the “event tornado”, BAM and other concepts explained in chap. 3. This applies in particular to the “drill-down feature” for the localization of the origin of a problematic situation like described in chap. 5. Further Java EE-5 concepts will not be discussed.

2.1 Enterprise Java beans

As Enterprise Java Beans [Sun 2006b] are used to implement the core business process application easyCredit-Internet, it is given a short overview of this technology as well as information about the roles of this implementation and a more detailed description of the session beans, entity beans and message driven beans technologies in view of their application in this use case and the differences to the already established EJB 3.0 specifications [Sun 2006c] which were not utilized in this project.

2.1.1 Overview

Enterprise Java Beans (EJBs) are part of the standardized, server-sided component-architecture Java 5 EE which makes it possible to develop distributed, transaction-controlled applications [Sun 2006b]. The EJB-standard is being consequently developed by the Java Community Process (JCP) [JCP 2007] with the goal to ease the development of Java applications and also EJB-components for distributed systems. The first publication of a prerelease of the Enterprise
JavaBeans Architecture was in October 1997. In this master thesis the possibilities of the EJB version 2.0 were utilized. But the new EJB 3.0 specification – which was introduced in June 2005 – will be mentioned too for further adoptions and prospective project dilation. EJB-components run on a special EJB container which provides the required middleware for communication, transaction management, security, persistence and so on. With the use of this infrastructure a developer has the possibility to concentrate on the implementation of the business logic itself. Through this reduction of complexity throughout each component a new level of software stability can be reached. Figure 1 shows a simplified but typical runtime environment of EJB-applications. A so called application server – like in this case the JBoss application server – normally provides a Web container and one or more EJB containers.

![Figure 1: Similar to Java EE 5 application server architecture](Sun 2006d)

As seen in Figure 1, one or more applications can exist independently within an EJB container at the same time [Sun 2006d]. These applications consist of again independent components, the Enterprise JavaBeans. These containers are categorized regarding their functionality: the so called session beans (see para. 2.1.3) implement the business logic of the application, entity beans handle the
persistent storage of application data and message driven beans (mentioned in para. 2.1.4) can be used to receive and handle asynchronous messages via the Java Messaging Services (JMS) (see para. 2.3). For client access protocols like Java Remote Method Invocation (RMI) [Sun 2006e] are allocated.

Besides enterprise beans and their interfaces, the appropriate deployment descriptors are needed for configuration purposes of the utilized application server or for altering the JNDI directory [Sun 2006d]. Those deployment descriptors are XML-documents, which hold all container-relevant information like security, transaction management of a method or for embedding interceptors (see para. 2.4). It is possible to modify or adjust these deployment descriptors even without having the source code of the particular component, like mentioned in para. 6.2 ff.

2.1.2 Roles

The EJB-standard not just means development of Enterprise JavaBeans, but also the role allocation throughout the development process of enterprise applications, like seen in Figure 2 [Boone 2003].

![Figure 2: EJB roles similar to [Boone 2003]](image)
Bean Provider:
The bean provider is the software developer who designs and provides EJB components. He provides the implementation of the beans, their home- and remote interfaces as well as the corresponding deployment descriptors. For this diploma thesis the TeamBank respectively the mandated Senacor consultancy [Senacor 2007] acts as the role of the bean provider. It wasn’t the mission of the authors to develop any business logic and to deliver any EJBs, servlets or JSP [Sun 2006d].

Application assembler:
The application assembler typically is the architect of the application which is created out of multiple components. The tasks of the application assembler are the assembly of the components to get a deployable application. TeamBank and mandated consultancies like Senacor provided the base application “easyCredit”. Key features like e.g. the integration of interceptors extended the already deployable application.

EJB deployer:
The EJB deployer is responsible for the deployment of the whole application. If it is necessary he has also to adjust further properties respecting the runtime environment (security, databases). The deployment is handled by the authors.

Server provider / container provider:
The server provider / container provider usually is a manufacturer who develops and sells an application server. He provides a full runtime environment for the EJB application and tools for the bean deployment. For this use case the JBoss Inc. [JBoss 2005a], [RedHat 2007] is the application server provider. The reasons why TeamBank decided to deploy the easyCredit-Internet application on this open source Java-EE based application server are based on their former evaluations concerning functionality, scalability or high availability - criteria of open source products which were evaluated as enough satisfyingly according to the requirements of the easyCredit-application.
System administrator:
The system administrator watches the application server and the running applications. The authors did the job of the system administration in a special test environment by themselves.

2.1.3 Session beans

The session beans typically implement the business logic of an EJB-application as utilized by TeamBank with the easyCredit. They are separated in two different types: stateless and stateful.

Stateful session beans can store information beyond several method calls. Usually this state drops away at the end of the lifespan of the session bean. Stateless session beans on the other hand do not hold any information for more than one call. They have to be endued with a default constructor and are instantiated on demand by the EJB-container to execute client requests. It is possible that they hold information, but those are not client specific and can get lost at anytime when the instance of a stateless session bean is destroyed. There is no guarantee for how long the information will be available. This way stateless session beans provide the container a simple way for memory optimization and calls without saving the data. Because they are not conversational it is possible for multiple users and methods to call the stateless session bean at the same time. This can be realized since the dynamic range of stateless session bean instances is based on the number of the calls on the stateless session beans. If a client wants to get access to one of these beans, an instance is automatically being assigned which executes the call. In the concern of a stateless session bean this instance will be set free to provide it to the next client request which uses this bean. If the client, on the other side, is calling a stateful session bean multiple times, the container has to assure that this client gets a bean assigned with the exact status. By this it does not necessarily has to be the same instance of this bean. But it has to be an instance of this bean with the same status. Session beans can be integrated in transactions. The transaction management can either be controlled by a container, the so called container managed transaction [Sun 2002a] or it can be controlled by the developer itself by coding the bean managed transactions [Sun 2002b]. The
behavior of beans can be set up in the deployment descriptor of the bean which defines how applications have to be deployed. The deployed Java EE 5 application needs an XML [W3C 2006] file which holds the information for the deployment descriptor. The classes implemented in this use case are proprietary EJB 2.0 conform. This is described in detail in chap. 5.4ff.

2.1.4 Message driven beans

With EJB 2.0 the message driven beans were introduced. They are special components which deal with Java Messaging Services (JMS) messages. The option for receiving the correlated event data of a CEP engine by a MDB was not utilized. Although all evaluated platforms provide a JMS output adaptor for sending data to a JMS topic, the vendor specific output adaptors for transferring the data to a visualization tool like dashboards instead is preferred. This way a clinical point of view is possible for every single evaluated CEP platform. For visualization a standard dashboard created with JFreeChart [JFreeChart 2007] is used as mentioned in para. 7.5.

2.2 JBoss application server

Following paragraph describes the basic concepts of the JBoss application server which was inter alia used for the deployment of the easyCredit-Internet application provided by TeamBank.

2.2.1 Introduction

The JBoss application server (JBossAS) is an open source J2EE application server, which currently runs on version 4.x. It is completely compatible to the Java J2EE Version 1.4 and Java EE 5. The J2EE1.4 certification process, stipulated by Sun Microsystems, is responsible for this compatibility. JBossAS passed this process on July, 19th 2004 [JBoss 2004]. The JBoss application server was developed in the year 1999 by Marc Fleury and his team. In these times the server was called EJBoss. 2001 the JBossGroup was
founded with the goal to provide professional support around the JBossAS. 2004 the JBossGroup changed to JBoss Inc [Fleury 2004]. One of the main attributes the JBossAS features is the possibility of the hot deployment. This facilitates the start, update or stop of applications while the server is running. It is not necessary to re-start the server which saves a lot of development time. Besides that an integrated Web-container and SQL-database is delivered together with the JBossAS (for further information see the JBoss homepage [JBoss 2005a], [RedHat 2007]).

2.2.2 The JBossAS architecture

The JBoss application server consists of a small core and components which are individually registered to the core. The core itself is an MBean server. MBeans help to index the single components as seen in Figure 3. The EJB container is a single component, too. On startup the server boots all MBeans and executes on the next steps their corresponding lifetime-methods.

![Figure 3: JBoss application server architecture (derived from JBoss 2005a)](image)
2.2.3 EJBs on JBossAS

On request of EJB methods by a client the call runs as well as on client-side as on server-side through several objects, each implementing different kinds of functionalities. This can be seen on Figure 4. A call goes through the proxy just to run through multiple interceptors within a client container. At the end of this chain there is a so called invoke interceptor. This invoke interceptor in turn calls the correspondingly detached invoker. The JBossAS supports several invokers which define themselves by the used transport protocol or by their cluster attributes. The detached invoker then routes the call to the corresponding MBean within the EJB container. After that, a series of server-sided interceptors are passed as well, before the call is passed to the requested component.

Figure 4: A typically Invoker Chain (derived from [Rupp 2005])

2.3 Java Messaging Service

The Java Messaging Service (JMS) [Monson-Haefel 2001], [Sun 2004] is a Java API for messaging between applications. Using a message service allows the integration of multiple applications within an enterprise. Therefore JMS is based on creation and delivery of messages. Because messages are structured data that one application sends to another, it fits exactly into the requirements of having a standardized way to distribute the generated event strings to all connected applications. The creator of the message is known as the producer and the receiver of the message is known as the consumer. The JBoss messaging provider is used to realize the transportation of the generated CBE-conform XML strings as a producer (see para. 5.3.2, 6.2.4) to the evaluated correlation engines which act the
role of a consumer. All evaluated approaches like e.g. the approach of TIBCO which features the TIBCO Enterprise Message Service (EMS) (see para. 7.3.1.3.1) implements JMS and integrate support for connecting other message services. The JBoss application server acts as an intermediary for the message and manages its delivery to the correct destination. Figure 5 illustrates the application used to produce a CBE-conform XML (see para. 5.3) string message. The JMSPublisher class (see para. 6.2.4) publishes the message to a JMS topic deployed on the JBoss messaging provider. Subscribed input adapters receive the message the moment it is published simultaneously.

![Diagram](image)

**Figure 5: Message delivery (similar to [Haase 2002])**

JMS supports two messaging models:
- Point-to-point (queues)
- Publish and subscribe (topics)

### 2.3.1 Point-to-point (Queues)

Point-to-point messaging has one producer and one consumer per message. This style of messaging uses a queue to store messages until they are received. The message producer sends the message to the queue; the message consumer retrieves messages from the queue and sends acknowledgement that the message was received. More than one producer can send messages to the same queue and more than one consumer can retrieve messages from the same queue. The queue can be configured to be exclusive, if desired. If the queue is exclusive then all queue messages can only be retrieved by the first consumer specified for the queue. Exclusive queues are useful when only one application is wanted to receive messages for a specific queue. If the queue is not exclusive, any number
of receivers can retrieve messages from the same queue. Non-exclusive queues are useful for balancing the load of incoming messages across multiple receivers. Regardless of whether the queue is exclusive or not, only one consumer can ever consume each message that is placed on the queue. Figure 6 illustrates point-to-point messaging using a non-exclusive queue. Each message consumer receives a message from the queue and acknowledges the receipt of the message. The message then is taken off the queue so that no other consumer can receive it [Haase 2002], [Tibco 2006d].

![Figure 6: Example of point-to-point messaging](image)

2.3.2 Publish and subscribe

In a publish and subscribe messaging system producers address messages to a topic. In this model, the producer is known as a publisher and the consumer is known as a subscriber. They are generally anonymous and are able to dynamically subscribe or publish to the content hierarchy. Many publishers can publish to the same topic and a message from a single publisher can be received by many subscribers. Subscribers subscribe to the topic, and all messages published to the topic are received by all subscribers simultaneously. This type of message protocol is also known as broadcast messaging because messages are sent over the network and received by all interested subscribers, similar to how radio or television signals are broadcast and received. Figure 7 illustrates publish and subscribe messaging. Each message consumer subscribes to a topic. When a message is published to that topic, all subscribed consumers receive the message [Haase 2002], [Tibco 2006d].
2.3.3 Conclusions

For this thesis the publish/subscribe concept was implemented due to the fact that in the point-to-point messaging model every message has just one consumer. The needed acknowledgement upon receiving a message as consumer and the problem of missing timing dependencies between the sender and receiver of a message did not fit the requirements. With the realization of this model the goal of a common initial position could be attained since each single message, created in the JMS publisher class may be received by multiple consumers simultaneously. Furthermore the model of publish and subscribe has a timing dependency. This implies that every client (application) that subscribes to the established “EventCloud” topic can consume just messages which are published after the client has created subscription. Besides that, the subscriber has to stay active in order for it to consume messages [Haase 2002], [Tibco 2006d]. For attaching dashboards for monitoring purposes the JMS publish and subscribe model proved to be adequate too, since more than one application can be the subscriber of a JMS topic containing the correlated events at the same time.

2.4 Interceptors

An intrusive way of the event generation would mean to modify the source code of the already deployed components. This would increase the complexity of each component because the event generation would then be part of the implemented functionality. Changes within the business logic or the functionalities may lead to changes of the event generation. A non-intrusive form of event generation can be
realized with AOP-based encapsulation techniques such as the proprietary EJB 2.0 interceptor of the JBossAS (see chap. 6). Events are already created in form of, for example SNMP-traps [Cisco 2006], however those events could not aid in filling the relevant situation specific event patterns (see chap. 5). To generate and send these additional and relevant events into the event cloud (as described in para. 3.1.2), which already contains the events generated by TeamBank, the interceptor concept was implemented as part of the event tornado concept (see para. 3.1.6). The following paragraphs will explain the approach of this concept.

2.4.1 Basics

Interceptors are methods or classes which are intercepting request on EJBs at runtime. For the invoker it is not realizable if a component, class or method is wrapped by an interceptor or not because the interceptor runs within the same context. This makes various extensions possible, because an interceptor has access to the local JNDI environment, data sources, EJB-references and other resources. By intercepting every target call to a session bean, the interceptor traces the activity calls within the business logic. According to chap. 5.4, the interceptor concept is based upon that the application server initiates an interceptor instance every time a session bean is being instantiated, either directly by client or by another bean. This instance of the interceptor, in return, generates two events; before and after the execution of the target bean. Therewith interceptors provide a concept to improve and extend the functionality of an EJB container in a dimension which was not possible to reach in prior EJB-standards [JBoss 2005a]. The interceptors are executed in a predefined order which is called “interceptor chain”. The order of the execution of each interceptor is defined within the deployment descriptor. Every step within this chain is not just passed in one way (the call itself), but also on the other way back (result of the call). With this technique it is possible with the help of interceptors to manipulate calls respectively results on the way to resp. on the way back of a call.
Figure 8: Example of an interceptor chain (similar to [Rupp 2005])

Figure 8 shows an example of a typical interceptor chain. Prior a call reaches its target bean, it passes several interceptors. After reaching the target, every interceptor which was passed on the way towards the target, is now passed in a backwards order.

2.4.2 Interceptors in EJB2.0

Figure 9 shows a simple example of a server sided interceptor. This interceptor extends the abstract class `org.jboss.ejb.plugins.AbstractInterceptor` and has to implement the abstract method `invoke()`. By implementing the `invoke()` method the interceptor receives an object of the type `org.jboss.invocation.Invocation` which encapsulates the attributes of the EJB invocation. With this object it is possible, within the interceptor, to query or edit every kind of information contained in the invocation to retrieve relevant data like “requested credit amount”, “calculated interest” or “calculated total payback”. This is the real entry point of the interceptor. Each interceptor has to guarantee the call of the next interceptor in the chain. This is done by passing the actual invocation object via the `invoke()` method of the next interceptor which, in turn, is instantiated by the `getNext()` method.

For implementation purposes EJB 2.0 based interceptors are implemented by extending the `standardjboss.xml` configuration file, within the conf folder of the application server.
Future plans of the project contain the realization of implementing new standards like EJB 3.0 based interceptor technology. Basically standardized EJB 3.0 interceptors work exactly like JBoss-proprietary EJB 2.0 interceptors. However, in the meantime, they demonstrate part of the JBoss AOP frameworks for AOP. Therewith an interceptor is not just restricted for utilization within EJB containers, but also can be deployed in every program which avails the AOP framework. An interceptor within the AOP framework is a so called aspect which interrupts an invocation of a method. Thereby the interceptor has access to as well as the parameters of the called function as to the results of the method [JBoss 2005b].

2.4.4 Conclusion

The adoption of older EJB 2.0 conform interceptors was no drawback. These interceptors provide the same methods and functionalities as the EJB 3.0 conform interceptors. By extending the deployment descriptor of the JBoss configuration file, they could be integrated without adding or modifying the source code of the already deployed bean classes. EJB 3.0 interceptors provide the usage of annotations, but for this use case the annotation technique was not necessary due to the fact that editing the code of already deployed components was no option.
3 The state-of-the-art of BPM/BAM/CEP/SOA

This chapter shall display the connections between the Business Process Management (BPM), Business Activity Monitoring (BAM), Complex Event Processing (CEP) and Service Oriented Architecture (SOA) concepts. It is described how those concepts complement one another and which relevance these state-of-the-art concepts depict for this master thesis.

3.1 Complex Event Processing

“As event-driven architectures continue to proliferate in the business landscape, a company’s agility and ability to drive big wins are becoming increasingly burdened by indecipherable data and unrecognized signals. That’s where CEP (complex event processing) comes in. CEP solutions continuously troll your real-time data streams in search of defined event patterns, then fire off alerts to your enterprise systems to automate a follow-on process or corrective action.”

[Brock 2007]

CEP is the processing of diverse event data streams. By definition of relevant event patterns, significant events within those streams can be identified. The event streams can contain single low-events, without business information, like SNMP traps or composite events which contain two or more events [Luckham 2002b]. Examples of low-level events can depict a page view, single key strokes, or a request on services or beans. A composite event is what a human or application derives from those event occurrences: a customer called a site, typed in his login data and pushed the login button.

Through CEP complex, abstracted events can be discovered, modified, created or destroyed by analyzing and correlating other complex or low-level events as described in para. 3.1.1.1 and 3.1.1.2 [Luckham 2007]. The evaluated CEP engines consume event-oriented data, utilize defined rule sets to the received events in real-time and process them in order to trace the path of a potential...
customer throughout the business process of easyCredit. CEP is used, upon corresponding of patterns, to execute business decision in real-time. By receiving single events which are part of an event-pattern, CEP can help to realize the concept of “predictive business” [Ranadive 2006] (see para. 3.1.5) by evaluating the chances of those patterns to match.

3.1.1 The architecture of CEP

To implement the concepts of CEP an event processing platform is required. Each event processing platform has its specific event programming language (EPL) (see chap. 8ff), event visualization tool, event storage and adaptors for mapping the data into the specific platform data model as described in chap. 7.

Within CEP platforms applications are designed and running, in order to process large amount of low-level events (see para. 3.1.1.1) or complex events (see para. 3.1.1.2) in real-time. These applications are utilized for multiple purposes like: fraud detection [Covelight 2006], click-stream analysis [SearchCRM 2007], network intrusion/exception detection [Stanford 2002], algorithmic trading [Bates 2007], or, as realized for this thesis, BAM (see para. 3.3).

3.1.1.1 Low-level events

Low-level events are events, which occur on the network layer. They do not have a semantic significance on their own for strategic or business decisions. Low-level events are immutable and are no abstraction of other events [Luckham 2002a], but can, being merged to event patterns, compose complex coherences. They can range from SNMP-traps [Cisco 2006], logbook entries [SGI 2004], logins [Talkback 2006], exceptions of applications, to database commits [Adobe 2003] and so on.

3.1.1.2 Complex events

Both, low-level and complex events are recognizable atomic occurrences which happen in an instant [Etzion 2004]. Like presented on Figure 10 complex- or
composite events are abstractions of at least two events [Luckham 2005a]. These can, in turn, be part of complex events themselves.

![Complex Event Pattern Diagram](image)

**Figure 10: Complex events as an abstraction of at least two events [Luckham 2005a]**

On the example of easyCredit, complex events were defined as patterns. Upon receiving new events, those patterns are fulfilled step by step. On completely matching a pattern, new events are triggered, alerts are sent or applications are initialized. Thereby complex event patterns can consist between two and an arbitrary amount of other events. As for the easyCredit use case the event patterns consist of two to five low-level events respectively complex events.

### 3.1.1.3 Event patterns

Event patterns describe templates of certain simple or complex event sets contained in the event cloud (see para. 3.1.2). Applications and CEP-based BAM tools (see para. 3.3) which are able to handle the data manipulation of events in real-time, trigger simple events self-sufficient, are able to process streams of events [Luckham 2007] and have the ability to detect patterns of the events within an event cloud to react in real-time on matching [Luckham 2002a]. Therefore
event patterns do not just describe the events themselves but also their timings, causal dependencies, data parameters and contexts. The so called event pattern rules are reactive and predictive rules which specify actions to be taken whenever an event pattern is matched or is probably going to match. These rules consist of two parts. The first is a trigger, which is an event pattern. The second part is a body of actions. This body of action is an event itself and will be fired as soon as the trigger is matched [Böge 2004]. Luckham defines an event as “an object that represents, encodes or records an event, generally for the purpose of computer processing” [Luckham 2007].

Complex event patterns are usually built out of many events emerged of the enterprise’s various layers. Those events can be created at different times – even within different time zones - or at different locations within the enterprise architecture and may involve events which are related to other events which are causally independent of the matching of the pattern itself. An example would be an activity pattern of a set of collaborating business processes, fraud activity patterns, etc [Luckham 2002a].

By matching those patterns, new events – the so called event abstractions or complex events – can be generated. The abstractions are enriched with relevant business information of the processed events which trigger the pattern while omitting useless details on the other hand. Those new events in return are sent into the event cloud themselves and can now be processed by CEP tools like other events. By utilizing this so called event pattern abstraction method [Luckham 2005b], high level views can be granted to the right persons, applications and components e.g. credit application reports can automatically be created and distributed, based on matched patterns of completed user application business processes of TeamBank.

That way complex event pattern respectively event pattern abstractions define the complex events created by matching the pattern. As soon as the empty event pattern template triggers, the variable parameters of the templates are replaced with values, derived of the relevant information pattern’s trigger events [Luckham 2007]. Like seen in Figure 11 every new event is an instance of the defined event pattern.
The low-level events respectively simple events have different degrees of causal and chronological dependencies. The timeline to the left displays the chronological flow, while the red arrows demonstrate the causal dependencies in-between the events. Events that do not affect the matching pattern but influence the credit decision process event are therefore relevant, too. The events are abstracted and aggregated and by matching the event pattern of a successfully approved credit application, the trigger fires and a new instance of a complex event based on this pattern is generated. The complex event itself just contains the relevant information of the lower level events for being processed itself. This new abstraction can now either fit in an event pattern too, or can represent a high level information event, which can be seen on a dashboard, monitored or sent to the right person or application in order to react to it [Luckham 2005a].

Following rules fit into this CEP concept:

- Complex pattern of events with causal and timing relationships have to be detected. This is the basic principle for real-time action or predictive actions by human or autonomous processes. There are various ways to
detect event patterns and their context throughout every layer and among themselves, for example heuristic methods, statistics, analytical functions or grid-computing [Aptsoft 2006b], [Aptsoft 2006c].

- To understand what is happening in the enterprises the moment it happens, event pattern abstractions have to be individually designed for the right persons in the right places. Only by that it can be guaranteed that real-time actions can be accomplished by the right instance of an enterprise. The possibility to drill-down (see para. 3.1.6) to each single event which led to the origin of the high-level event helps to reproduce the actual situation [Luckham 2005a].

3.1.2 Event Cloud

The term event cloud was coined by Luckham [Luckham 2002a]. It classifies the fact that events flow to and from each enterprise. Each enterprise reacts to the events it receives and communicates by sending new events. Therefore enterprises operate in a global event cloud. CEP is used to manage the events in the cloud [Luckham 2004d].

Local IT-systems operate in an event cloud like the IT-system of TeamBank. Other systems can be e.g.:

- Chip fabrication lines.
- Automobile assembly lines.
- Automated warehouses.
- RFID tracking systems.
- Algorithmic trading.

All of these IT-systems are event-driven. Their activities are prompted by events. They receive events and react on them. They communicate by events. They generate new events and send them to each other. By this, billions of events per day are received and reacted to [Luckham 2005a], [Luckham 2005b]. Figure 12 explains the event cloud by means of a typically possible event cloud of an online banking website.
As shown in Figure 12, complex event patterns describing business relevant information like possible frauds, performance analyses, load analyses or the realization of business processes respectively single components can be defined based on events which occur within the IT-systems of an enterprise. That way the correlation of the provided events of the event cloud in real-time via CEP platforms enables the company to react in real-time on certain situations and reduces the level of IT-blindness by mapping every single event to a specific event emitter. Furthermore the event cloud constitutes some kind of interface between the IT and the business departments of every enterprise. Both parts can build their monitoring views on events within the event cloud. The only difference is the varied abstraction of these events in order to visualize the information they need. Figure 13 illustrates this connection.
As seen in Figure 13, the cockpit on the left side is the most abstract point of view. Here all important information is centralized, can be obtained through for example a control flow indicator, or can display the aggregations of the layers below like the process- or function level. On the right side, the IT-infrastructure generates the events. The business departments and the IT-departments are connected to the same event cloud. Therefore both use the same information pool. However the monitoring views are different, depending on the abstraction of the used low-level events [Jobst 2006].

3.1.3 IT-blindness

“The fewer data needed, the better the information. And an overload of information, that is, anything much beyond what is truly needed, leads to information blackout. It does not enrich, but impoverishes”

Peter F. Drucker

Founding father of the study of management

The term “IT-blindness” was also coined by Luckham [Luckham 2002a], [Luckham 2004b]. It specifies the problem that companies produce such a tremendous amount of data and events that this cannot be reviewed and appraised anymore. Because of that, enterprises are not capable to identify important information, problems, tendencies or effects on certain scopes, which can be derived from data or low-level events. Therefore it is not possible for the enterprises to predict how specific event patterns of the IT-levels of the company
have impact on higher business objectives, strategies or business processes. Events of diverse origins could indeed be sent into the event cloud, but this does not solve the problem of IT-blindness inside the enterprise. An aggregation of void events in form of an event cloud is therefore even counterproductive because increasing the sum of data which cannot be handled anyway does lead to an even more unmanageable situation [Ammon 2006a], [Ammon 2006d]. Companies like TeamBank can send and receive thousands of events without semantic per second or usable information for CEP-based BAM. For example SNMP traps events [Cisco 2006] are already clustered in an event cloud, logged and realized, but could not be processed in order to improve the IT-insight. Thus TeamBank accumulated events and data in an event cloud they produced or received on one hand, but from this, processes and relevant business level information cannot be identified properly. This shows that events can be generated in order to enrich the event cloud, but if they don’t contain any kind of necessary information, they cannot help in order to solve the problem of the IT-blindness.

3.1.4 IT-insight

One of the goals of companies like TeamBank could be to reach a certain level of IT-insight. This can be obtained by understanding how complex event patterns (see para. 3.1.1.3) eventuate due to events emitted by multiple components of different IT-layers, like for example the network layer. By realizing how those event patterns impact high level business objectives, strategies and business processes IT-blindness will turn step by step into IT-insight [Luckham 2002a]. CEP based real-time BAM (see para. 3.3) helps to gain more IT-insight for companies [Luckham 2004b]. Through the monitoring of applications and components within the IT-system by correlating the events they emit, every connection between them can be recognized. By this, it is possible to realize the impact a component has on another components, even if they are not meant to be connected in any way. Gaining a high level of IT-insight, a company can accomplish competitive advantages by realizing the vision of “Instant Insight” which “is a breakthrough analysis of electronic commerce which should be kept secret and used to beat rivals” [Luckham 2004c].
The goal for archiving a high level of IT-insight by processing the events generated through the IT-layers, is the possibility of the realization of the concept of “predictive business” (as seen in para. 3.1.5) [Luckham 2004c], [Luckham 2002a], [Vinayak 2006]. This will turn the businesses to anticipatory enterprises with predictive supply chains. Luckham characterizes this as a way from supply chains towards “eager chains” where orders can be fulfilled just at the right time based on the event patterns who categorize the necessities [Luckham 2004c]. Therefore the delivery of real-time operational IT-insight enables timely well-informed decisions and visualizes the abstracted data for different information consumers. By correlating events generated on the activity of every single component, TeamBank reaches a certain level of IT-insight and therefore is in the position, to act on possible credit applications in real-time while users are still active. Due to event pattern definition of typical user actions, TeamBank can act predictively, based on the customer’s input, while he is online. Therefore, based on these inputs, special conditions, possibilities and solutions can be provided in order to help the potential customer fulfill his application.

3.1.5 Predictive Business

The concept of “predictive business” is not just to react on customer demands and certain situations in real-time, but to anticipate them [Ranadive 2006]. This forecast in business activities can be realized by recognizing and defining complex event patterns – based on the events of the event cloud (see para. 3.1.2) – and defining BAM-views in terms of real-time capable, preemptive ways, together with the companies itself; like in this case, the TeamBank. Besides that, compensating actions and preventions have to be specified on occurrence of certain events respectively event patterns. To translate this idea into the concept of predictive business, low-level events without semantics have to be aggregated that way that complex event patterns respectively high-level events with business impact can be recognized. After recognizing and defining these patterns, possible actions in forefront of the incidence of a business critical situation can be defined (see [Ranadive 2006], [Luckham 2004c], para. 5.2 ff).
3.1.6 Conclusions and recommendations – the “event tornado”

Like quoted in para. 3.1.2, TeamBank logs and transmits lots of low-level events without semantics, like for example SNMP-traps, from every component throughout the IT-system into the event cloud. These events do not support the achievement of the desired level of IT-insight. Therefore the concept of the “event tornado”, coined by [Ammon 2006a], is applied on the already existent event cloud. Together with the low-level events without semantics, additional events have to be collected by the “event tornado” from out the business processes and components to enrich the event cloud. Like displayed in Figure 14, high-level events with impact on the business logics, strategies and business goals are generated and collected by the “event tornado” [Ammon 2006a].

![Diagram showing the event tornado concept](image)

*Figure 14: The event tornado provides additional events from business process components and SOA services for a better IT-insight [Ammon 2006a]*

By realizing the concept of real-time based BAM-views (like explained in para. 3.3), TeamBank is able to detect which impact the IT-infrastructure has on the business itself and can react preemptive on situations which are going to happen. Furthermore with the realization of a real-time based BAM-tool, based upon the
additional events of the “event tornado”, TeamBank is in a position to recognize each specific low-level event and its origin by analyzing the accordant pattern. This is the fundament to facilitate the implementation of the drill-down concept. By using the drill-down technique, it is possible to focus the lowest layers and single components or event emitters of the IT-levels of the company. This way the accrualment of problematic events can be identified and revised in order to circumvent critical situations or settings with negative business effects. Therefore the realization of the “event tornado” concept gives TeamBank the possibilities for:

- Predictive actions to circumvent critical situations respectively failures in the early development stage.
- Utilization of drill-down techniques to focus the most bottom IT-layer to fathom specific situations.
- Setting up a CEP-based real-time BAM providing the possibility of instantaneous actions on espousal of defined event patterns.

3.2 Business Process Management

“Each company’s unique way of doing business is captured in its business processes. For this reason, business processes are rapidly becoming the most valuable corporate asset. BPM provides companies the ability to model, manage, and optimize these processes for significant gain.”

Vivek Ranadive  
Chairman and CEO  
TIBCO Software Inc.

BPM is the methodology of system implementations and change managements which makes continuous comprehensions and the management of business processes possible. This concept is based on following assumptions [BEA 2005], [TIBCO 2006a]:

- Different business processes cross-cut each other.
- Business processes are flowing between multiple interested parties and organizations.
- Business processes interact with systems and people equivalently. People can be partners, customers, suppliers or employees.
- Business processes are permanently developing and changing.

The successful deployment of BPM suites will benefit both lines of IT-departments and business. For TeamBank as an organization as a whole, BPM could ensure business process visibility and transparency. This leads to reduced error rates, higher productivity and tighter compliances with legal requirements. BPM may enable TeamBank to adapt to changes in the marketplace (e.g. the introduction of new ideas or products), improvement of costumer services and the reduction of operational costs directly within their IT-system. A random survey of TIBCO BPM customers, conducted by the independent agency Intercaim Mondiale, reported a 100% increased productivity, 95% improved quality of services, 82% reduced operating costs and 82% saw faster process cycle times [Intercaim Mondiale 2005]. This displays how BPM can help companies like TeamBank to connect disparate systems while increasing the value of their current investments and gaining more IT-insight. Besides that, the successful deployment of BPM allows the IT to proof their current infrastructure for future improvements and to adopt changes or reinvention of the business processes more effectively. An implementation of the SOA concept of the infrastructure of the TeamBank would allow rapid deployment and development of new applications, components and processes. Thereby the IT can be more responsive to the fluctuating demands of the TeamBank management organization. This demonstrates the reputed influence of BPM on technologies or business purposes. By modeling their business processes, organizations are able to save money, time, gain insights into the company’s structures and deliver values through the supposed return on investments (ROI) [Handelsblatt 2006]. By defining the BPM concept, the importance of the ability of TeamBank in order to stay competitive while remaining agile through adapting their business strategies to changing global marketplaces could be accomplished [TIBCO 2006a], [Handelsblatt 2006]. However an evaluation of the advantages of BPM is not part of this thesis.
3.2.1 Business Process Modeling

For most use cases business processes and structures of organizations are known – and especially in the German language regions – as “Ereignisgesteuerte Prozessketten” (Event-driven Process Chains (EPCs)) by the tool set ARIS [Scheer 2000]. Other popular business modeling languages could be IDEF0 [Hill 1994], [Vinayak 2006] which is still an official standard in the USA or IDEF3 as well as languages based on catalyst [CSC 1995], [Kalnins 2003] notation charts or simple flowcharts. In case of ARIS modeling, EPCs are functioning methods of business process modeling. This kind of modeling is based on Petri nets [PetriNets 2004]. As an example of the simplicity of modeling business processes with the ARIS language style, Figure 15 illustrates a small fragment of an order processing example.

![Figure 15: Example of business process modeling using ARIS at the example of an Order Request [Barzdins 2004]](image)

The notation seen in Figure 15, can be seen generally as flow-chart based. Activities (functions, etc.) are linked by control flows respectively object flows. Action performer activities can be specified most frequently by the slim line notation. Thus activities can be refined by other diagrams of the same type. Standard business processes start with some kind of start symbol; activities are executed sequentially in a prescribed order, dictated by the corresponding control flows. Possible concurrent execution is possible too, as long as the diagram explicitly describes this attitude [Barzdins 2004]. Another way to model business processes can be accomplished by defining them via UML activity diagrams.
Activity diagrams have become widely adopted with the advent of UML version 1.3 and this type of diagram was heavily revised in the UML version 2.0, released in the year 2005. Since then, the UML approach has been admitted to be a proper part of business modeling and is therefore standardized now [OMG 2004]. The larger an enterprise becomes the more processes have to be modeled. So the number of processes can reach more several thousand per company. A new and flexible way to handle such an amount of processes, concepts like SOA (see para. 3.4), BAM (see para. 3.3) or especially BPM (see para. 3.2) may help to transfer the overview of as well as the company management as the IT-departments [TIBCO 2005], [Luckham 2004a], [Luckham 2004b]. Realizing those concepts may lead to flexible, adoptable business processes and transparent IT-insights (see para. 3.1.4). Figure 16 displays that. In doing so, business processes have to remain first priority at whose business steps services have to be applied.

TeamBank designed their business process „easyCredit-Internet“ using the EPC notation. This process consists of a series of multiple business related activities respectively components. According to the SOA concept, every activity can either
represent internal or external resources in form of, for example, Web services. Figure 17 illustrates this modeling of TeamBank’s easyCredit business process.

As seen on Figure 17, the customer first hits the „Minicalculator“ page. This component gives the customer the option to choose his preferred amount of credit,
the payback duration, etc. After that, credit information is provided, finance data is collected and the chosen replacement payments are calculated. Within the next steps more details about the customer are collected followed by an agreement for the data protection clause in order to save the customer’s typed in data. On agreeing to the data protection clause the customer’s data is saved and a credit decision process is executed. The final decision is done either automatically or, in critical situations, manually by a TeamBank employee like mentioned in the pipeline model in para. 4.2.

3.2.2 Business Process Execution

For the execution of an EPC model, it has to be described and defined in a language which is able to execute the process on its own. The BPEL/WS-BPEL execution language was a joint proposal of Microsoft and IBM for advancing the standardization process for the controlling and coordination of business based Web services and components within business processes which got approved as an OASIS standard 2006 [OASIS 2006c]. Using the XML [W3C 2006] based language BPEL, business processes can be modulated in form of Web service chains and modeled for the coordination with business partners. Thereby BPEL defines the interface protocols which assess the sequence of dialog steps and describes the communication with business partners [IT-Wissen 2007]. Business processes can be described in two ways [Service-Architecture 2005]:

- As an executable business process model behavior of a partner during a business interaction.
- As business protocols that use process definitions and descriptions that specify the mutually visible message exchange handling of each of the parties included in the protocol, without explaining their internal individual behavior.

Those business protocol’s process descriptions are called abstract processes. The behavior of both, abstract and executable processes can be modeled using BPEL [Service-Architecture 2005].

This includes the:

- Correlation of process instances and messages.
• Behavior of recovery in case of exceptional conditions or failures.
• Web services based relationships between process roles.
• Sequencing of Web service interactions and other process activities.

Often the requirements of the SOA concepts of progressive process automation and the definition and execution of business processes with workflow engines [Wikipedia 2007c] on the basis of standardized BPEL-languages with Web services cannot be satisfied anymore by the originally designed processes. They have to be remodeled and re-designed using the SOA-principles via executable business process modeling languages like BPEL or non-executable languages like e.g. ARIS [Oracle 2004]. On the strengths of the above mentioned abilities and opportunities an executable modeling language offers, TeamBank might reach a new level of IT-insight by re-designing their current business processes according to these basic principles. However a reason not to utilize BPEL driven models lies in its deficit of not supporting human interactions. The specifications of WSBPEL 2.0, Oasis standard release April 2007, still does not comprise human interactions as a standard feature [OASIS 2007]. Since business processes, implemented by TeamBank Company, contain human interactions, a way to solve this would be to access proprietary solutions of diverse vendors. However this could turn out to be problematic hence dependencies on particular vendors might arise.

3.2.3 Business Process Monitoring

The concept of Business Process Monitoring (BP Monitoring) is the visualization, monitoring and administration of events within business processes and its underlying IT-infrastructure and applications. The status of every component and IT-resource can be depicted and integrated in a breakdown. Therefore a higher-level visualization of the business process status can be granted which, in return, reduces the diagnostics of exception causes in complex applications systems [@ctiveFRIEND 2005]. As seen on Figure 18, the EPC of the “easyCredit-Internet” application can be monitored as well as a complete business process or as a set of several single business activities and components. This kind of tracking helps to facilitate the monitoring of every state. Performance and statistical information like measurements of how quickly a credit application is processed or
how many applications were processed within a specified timeframe can be provided too, via BPM. Three categories of statistical information monitoring can be defined as:

- Defect rate monitoring of each component.
- Average cycle time of every single credit application.
- Measurement of the business process productivity.

As seen in this example, sensors can be placed at the relevant points within the EPC to provide and record this kind of information.

Figure 18: Business Process Monitoring using the example of "easyCredit-Internet" EPC [Greiner 2006b]

Figure 18 illustrates the ability of the determination of every state of a credit application, like application ordered, application is being rated and if an application is approved or not. In this EPC just contracts which have been saved in the database can be tracked throughout every component. Real-time based BAM, like mentioned in para. 3.3, is the next step from the current ad-hoc monitoring of BP Monitoring towards a real-time analyzing of the data. The exact
level of monitoring depends on what information TeamBank Company needs to evaluate and analyze. Based on this information the status of each application can be determined. Additional to the individual usage of the information, it can be used to help suppliers and customers improve their own connected processes. However, the reasons for a declination or approval of an application cannot be recognized that way. Therefore an additional solution has to be found (see para. 3.3).

3.2.4 Conclusions

If TeamBank’s decision would be to design their main business application “easyCredit-Internet” using an executable modeling language, a new understanding of the business processes within the company might be accomplished. Together with the possibility to adopt and react on chances and changes, a higher level of IT-insight and flexibility could be reached. The efficiency, effectiveness and visibility of these re-modeled processes might enable TeamBank to reach and exceed their business goals and to differentiate them within a strong competitive market. By that, TeamBank might be able to take direct ownership of their main processes without suffering from delays by virtue of new implementations of the IT and to react on every change of the process. Through accomplishing this goal of connecting SOA approaches with BPM concepts, the so called “business-IT divide” is being dramatically reduced [Smith 2006], [Luckham 2005b]. Both operation departments at TeamBank Company – as well as IT department as management level – could plan and design the same projects, but solve different individual problems. This is made possible by focusing of both sides on the BPM-conform business process. The designed process model has to be passed towards the corresponding process engine for the compilation and implementation.

3.3 Business Activity Monitoring

As mentioned in para. 3.2.3, BAM and BP Monitoring differ in several points. It has to be considered which monitoring fits best, based on the given IT-
infrastructure, the modeled processes and the business goals. If the IT-infrastructure already features the generation of all relevant events, BAM would provide an appropriate way to monitor situations in real-time. According to Gartner’s term of BAM, events, which are generated from multiple application systems, enterprise service busses or other inter-enterprise sources, are being processed in real-time to identify critical business key performance indicators in order to get a better insight into the business activities and thereby improve the effectiveness of business operations [McCoy 2002]. BAM identifies and analyzes cause-and-effect relationships among events in real-time, allowing personnel and/or systems to proactively take effective actions in response to specific scenarios. Figure 19 shows the application integration of a real-time enterprise using the CEP concept as event correlation engine and BAM. Events can be generated throughout the enterprise. They are correlated in real-time and monitored on dashboards. This shall enable the enterprise to optimize every single component, design better strategies or react on alerts in real-time.

![Figure 19: Example of a real-time enterprise, using BAM](TIBCO 2004b)

With BAM it is possible to monitor the activities of every utilized component and either present all generated events as they are being processed, just key performance indicators (KPI) or only events which lead to an alert within the system in real-time to the temporarily person in charge. Based on interests of the
U.S. Department of Defense (DoD) in information superiority, Hayes-Roth researched into information supply to “quickly get information to those who will benefit”. Therefore he studied alternatives to the smart pull concept, which is a theory that a system that enables every user to pull relevant information results in information superiority. The most direct alternative is the Smart Push theory, which assumes that many operators can and should delegate the work of monitoring for newsworthy information to a system with enough smarts to determine when a user needs specific information. The Naval Postgraduate School and the World Wide Consortium for the Grid (W2COG) has been working on implementations of the smart push concept. They call the concept VIRT, which stands for Valuable Information at the Right Time [Hayes-Roth 2006]. Gunderson from W2COG defines VIRT as an approach to the increasingly vast market place of data that encourages careful choices regarding how valuable time is spent “buying” and “consuming” information. An institution gets competitive advantage from finding opportunity and acting quickly. It loses competitive advantage if it wastes time processing insignificant data. Time spent acting on critical information makes money and wins battles. A decision made too late is not a useful decision. He describes VIRT productivity model as Productivity ~ Value/Bit [Gunderson 2005], [Gunderson 2006]. Hayes-Roth says, that VIRT answers questions like “Who needs what information?”, “How does that information find them?” and “How to assure that receivers are not glutted by a deluge of low-value data and consumed by attendant low-value tasks?”. The answers to these questions employ dynamic context and operator requirements to assure that high-value information flows quickly where it's needed and is processed promptly by recipients [Hayes-Roth 2007]. He also says that VIRT services essentially filter information so that high-valued bits are prioritized and low-valued bits are deprecated or withheld. In this way, each collaborator’s incoming queue of messages is dynamically prioritized, enabling the person or agent to work on the most important information first. This “best first” approach provides the productivity-gains organizations need to thrive in a networked, information-rich environment [Hayes-Roth 2004].

To realize VIRT, following components are necessary [Hayes-Roth 2004]:
• Tools for operators to describe events that they need to be alerted about, which are called “conditions of interest” (COI).
• Vocabularies and semantic schemas that relate available information sources to such COIs.
• Geospatial and time-indexed databases for maintaining current and forecast values of the relevant variables.
• Continuous monitoring for changing values of conditions of interest, including projected future states.
• Alerting methods for communicating significant news quickly and concisely.

![Simplified Architecture for VIRT](Hayes-Roth2007)

**Figure 20:** A simplified architecture for VIRT [Hayes-Roth 2007]

This concept of presenting VIRT is one of the key features BAM offers in opposite to traditional real-time monitoring concepts [Hayes-Roth 2007]. BAM monitoring of single or multiple business activities helps to reach, in relation to the important business strategies, following goals in real-time:

• Early detection of abnormal characteristics of whole business processes respectively single business activities or components.
• The reference to the business strategies, concepts and goals helps to understand the impacts on possible affected Service Level Agreements (SLA) [SLA 2005] or what business process component cannot be executed anymore properly.
• Cause analyses of internal or external failures and impacts onto the business process, the infrastructure, applications or flows.
Risk analyses which help to predict compliances of volumes, deadlines, data throughputs or failure rates.

The automatic deduction of action plans.

Other goals of BAM can be seen in the supply of organized information for a quick understanding of problematic situations and decision making. This can be implemented by defining special, custom-made dashboards (even with integrated historical views for comparing issues), real-time alerting in order to control operative risks and to have the possibility to react on customer demands in real-time [Systar 2007a].

Figure 21 displays the need for the definition of different BAM dashboards in order to provide the required information to each particular information consumer, key person or business level.

![Figure 21: Different information requirements need different BAM dashboards for visualization](Clarseco 2007c)

### 3.4 Service Oriented Architecture

SOA is a concept for system architectures. Those architectures implement functions in term of loosely coupled, independent services or software components that define reusable business functions. SOA was first described by Gartner in 1996, but the recent interest in the architecture has been spurred by the emergence of the powerful industry trend of Web services. Although Web services do not necessarily translate to SOA, and not all SOA is based on Web services [IBM 2004b], the relationship between the two technology directions is
important and they are mutually influential: Web services bring SOA to mainstream users, and the best-practice architecture of SOA makes Web services initiatives successful as for the instant online applications like easyCredit. Essentially, SOA is a software architecture with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations and interface calls. SOA is a relationship of service providers and service consumers. Services are software modules that are accessed by name via an interface, typically in a request-reply mode. Service consumers like JSP are software that embeds a service interface proxy (the client representation of the interface) [Natis 2003]. In contrast to SOA many companies still feature heterogeneous, historical grown system sceneries which union workstations, individual software and standard software. Figure 22 displays traditional integration practices like peer-to-peer connections [Wolff 2004].

Figure 22: Integration with traditional methods (like e.g. peer-to-peer connections,...) [Ammon 2006d]

By cutting and porting such architectures towards enterprise application platforms to be Enterprise Application Integration (EAI) conform the SOA concept can be
adopted. Figure 23 shows how the chaos displayed in Figure 22 can be ported into a multi-tier architecture application-server based integration which provides certain advantages like standards, synchronicity, asynchronism and so on.

![Diagram of a multi-tier application-server based integration](image)

**Figure 23: Example of a multi-tier application-server based integration [Ammon 2006d]**

One of the main advantages of the SOA approach is that by building standard-based interfaces between components, developers can incrementally construct applications and swap out, reuse and modify components without having to concern themselves with their inner workings. Designers of Web services typically describe the interfaces using the Web Services Definition Language (WSDL) [W3C 2001] and send XML-based messages between components using SOAP over http [W3C 2004].

3.4.1 **Event Driven Architecture**

Event Driven Architecture (EDA) is complementary to SOA [Sliwa 2003]. It describes an approach for designing and extending applications or components as part of a SOA. Triggers generate events which are sent via messages in between those independent components. The difference in contrast to components of a SOA is, that the components within an EDA are completely separated respectively decoupled and do not know the existence of other components. Information and interactions among components just happen due to event receiving / sending. SOA and EDA consist of two distinct interactions. Those interactions between events
and services are referred to as Event-Driven SOA (EDSOA) [Seeley 2006]. In an EDSOA the occurrence of one and more events can trigger the invocation of one or more services. Those services may perform simple functions, initialize the execution of entire business processes or generate events. The event may signify a problem or impending problem, an opportunity, a threshold, or a deviation. Upon generation, the event is immediately disseminated to all interested parties (human or automated). The interested parties evaluate the event and optionally take actions. The event-driven action may include the invocation of a service, the triggering of a business process, and/or further information publication/syndication. In this interaction, the service is purely one of many event sources in a broader event-driven architecture [Michelson 2006].

3.4.2 Recommendations

The tailoring of components and applications like the easyCredit instant online application as services would lead to a SOA-conform concept. This, in turn, is complementary to an EDA. The generation of events throughout components within this architecture could be used for prospective CEP-based applications. Following this path, the development and maintenance of independent and shared sets of unpredictable and asynchronous activities can be realized. Besides that, if the same data has to be processed at multiple destinations within the IT-infrastructure, the EDA approach is more efficient, in oppose to a pure SOA approach, due to the fact that just one event has to be generated and will be received at every single component simultaneously. Within a pure SOA, a client would have to make gradual calls. Another benefit of integrating the easyCredit application within an EDA might constitute advantages like low-cost reconfiguration, reusability of business components within other applications and the reassembling of business components in new business processes.
4 The easyCredit-Internet application

The browser based easyCredit-Internet application of TeamBank is a German online credit system featuring instant application confirmation. It was the first online instant credit application worldwide and got introduced in the year 2000 by TeamBank, the former Norisbank. To apply for an online credit, customers have to click through various JSP [Sun 2006b] using a standard browser like shown on Figure 24.

![Figure 24: Example of JSPs of the easyCredit application taken from www.easycredit.de](image)

Hence its successfulness, the aim of TeamBank now is to going to go worldwide and to raise this achievement to an international level. But with this expansion, many more new potential customers will increase the impact on this very kernel business process. As mentioned in para. 3.2ff, the actual implementation of this business process is an inflexible non-BPEL workflow which is designed using a non-executable business process modeling language. Following paragraphs will describe the use-case and the actual as-is solution of the easyCredit process with all its process steps, particularly with regard to arising the present easyCredit to the “Next Generation Instant easyCredit System”.

4.1 The use case

The use case “monitoring of the loss of sales volumes by cancelled credit applications” is specified according to the UML 2.0 notation as described in
OMG’s UML 2.0 superstructure [OMG 2004]. According to this superstructure, the use case is necessary for the modeling of behaviors and structures of platforms and applications. It shows a certain view on the expected behavior of a system and is part of the system’s specification. Typically a use case diagram represents applications and participants as actors with their dependencies and relations. Figure 25 displays the use case diagram “monitoring of the loss of sales volumes by cancelled credit applications”.

![Use case diagram](image)

**Figure 25**: Use case diagram of the monitoring generated loss by cancelled credit applications of the next generation easyCredit-Internet

As displayed in Figure 25, the actor “easyCredit” on the left side creates via event generation the “event cloud” as described in para. 3.1.2. Actually it is a matter of two parallel running systems: the easyCredit application which is used for the credit application instantiation and the parallel running monitoring system which supervises the credit application instances. Depending on user requests and movements within the easyCredit application, events will be generated and sent into the “event cloud” according to the concept of the “event tornado” as depicted in para. 3.1.6. The actors who interact with the monitoring system are displayed on the right side of the diagram. They are called “operator”. Different actor instances of the type “operator” can have different entitlements and roles. The administrator can define and customize BAM views, define alerts and assign
transmission priorities of different roles or positions. On the other, hand actors with a C-level position [Wikipedia 2007f] can realize certain BAM-views inclusive the accordant drill-down features down to the cause of the critical situation resp. the event-origin. This C-level manager can monitor every type of process within the enterprise while an administrator can just observe the processes according to his entitlements. Part of this master thesis however is only a test of functionalities of the evaluated CEP-platforms. The role management etc. could also be an integration of a connected visualization tool, but is not part of the thesis. The part which is covered in this master thesis is the process “Internetgast” only. Thus a totality of all processes is restricted to this very process.

4.2 The as-is solution of easyCredit and the pipeline model

In the actual as-is solution the easyCredit application is an inflexible hard-coded, predefined business concept without the actual possibilities, flexibilities and advantages BPEL-based concepts come with. Part of this as-is solution already supports certain BAM features: a business process monitoring from a business-management point of view based on key data. This has been realized in a pipeline model [Ammon 2006b].

4.2.1 The pipeline model

As seen on Figure 26, this model assumes that every single credit application passes several steps within the application process during its lifetime, whereas single steps can be seen as a separate workflow respectively production line. Every section represents the progress of an ongoing credit application. Therefore the current position of an active customer within the easyCredit frontend can be implicated. Within this pipeline it is possible to trace every credit application and assess how many applications are within the whole model at a specific time. At the moment the amount of all applications are being calculated every hour. Thus it is possible to compare the calculated amount of credit applications per section with specified thresholds. Alarms will be activated if thresholds are exceeded for single sections indicating a technical fault [Ammon 2006b].
To extend the possibilities of not just calculating the amount of credit applications every hour or verifying if the application or process are available, the chances a CEP-based BAM offers were utilized.

4.3 The challenges for the Next Generation easyCredit system

The near-future goal of TeamBank to profit of the benefits could be to redesign that process in sense of SOA (see para. 3.4) and eventually to obtain the flexibility which can be reached by implementing the business concept by means of a BPEL-based workflow engine (see para. 3.2). Once the process is BPEL based, the events emitted by the BPEL-engine can be discerned or used together with the events which are being processed in the event cloud (see para. 3.1.2) by a CEP system. Some of the evaluated CEP platforms already feature input adaptors for events generated out of the BPEL process. Since this re-design would exceed the timeframe set, it was not implemented in this thesis.

4.3.1 Business Process Management challenge

Like mentioned in para. 3.2 BPM is a main part for setting up a SOA-conform architecture for a future event-driven enterprise with a high level of IT-insight and predictive abilities (see para. 3.1.4 and 3.1.5) [Ammon 2006d]. The challenge
here lies in the redesign and remodeling of business processes, using business process execution languages of the already modeled business processes without the accordant execution abilities. For an automatic event generation – either in every single process step or on completion of the process – the workflow model of, for example the easyCredit-Internet process can be executed by a BPEL-based workflow engine as a component deployed on the application server.

4.3.2 Multichanneling challenge

Another challenge to realize the features of BAM, BP Monitoring and CEP for the Next Generation easyCredit-Internet process lies in “multichanneling”. This describes the actual prevailing situation many enterprises have to deal with from a technological view. That is, the problematic of having multiple different processes and channels which are leading to the same result: the initialization of a business process [Ammon 2006d].

Figure 27 illustrates this at the example of a customer process of a retail bank.

![Figure 27: A typical customer process of a retail bank [Ammon 2006d]](image)

As seen in Figure 27, different channels like online terminals, call centers or other channels, are leading to the same business processes and goals. Here the effective completion of a credit disposition. According to the concept of CEP-based BAM, companies like TeamBank have to cope with the challenge of monitoring and implementing the involved channels via event generation in order to administrate the channels in delivering the same results and leading them to the same business process and real-time. By acquiring this goal of solving the multichanneling problem, CEP-based BAM concepts can be realized of real-time monitoring.
4.3.3 The challenge of Service Oriented Architecture / Event Driven Architecture

Yet another challenge to realize the opportunities and possibilities CEP-based BAM respectively BPM have to offer lies in the concept of SOA itself. Like mentioned in chap. 3, the SOA respectively EDA concept is the fundament of a fully event-driven enterprise with all its benefits provided by CEP-based technologies. Since many companies, like in this case study TeamBank, still design and run non 100%-SOA-conform business processes, the challenge of the successful realization of SOA/EDA is based on the modeling respectively remodeling of business processes under the perspective of a SOA. The goal of a business process utilizing the SOA principles is that every process step can request a service or call a sub process related to their particular required operations. These services respectively sub processes can be arranged either inside the enterprise architecture or can constitute external service/sub processes, which provide their functionality to any other process worldwide [Ammon 2006d]. This could be cascaded arbitrarily deep.

Figure 28 illustrates an example of possible customer credit process which realizes the SOA concept as depicted above.

![Figure 28: The SOA challenge at the design example vertical and horizontal coupling of services of a customer credit process [Ammon 2006d]](image-url)
4.4 Traditional OLAP-based views versus future CEP-based views

Online Analytical Processing (OLAP) [Codd 1998] is an analytical information retrieval system like data mining [Grob 1999] and is a component of the decision oriented information processing. That means the provision of data as first principle in order to gain more business insight or support business decisions.

4.4.1 OLAP-based views

The described as-is situation of the easyCredit application assumes that each single credit application with its total life cycle passes through several different processing steps; one processing step after another like within pipeline a workflow or a production line. However, the precondition for an adequate technical monitoring is that the status of each application has to be recorded with the corresponding time stamp in a database. This is the only way a database request can be established in order to retrieve the information of how many applications are located, at the time of the status change, within every section of the pipeline. The only advantage of this monitoring is the relatively easy procedure of the realization itself. This approach however just grants a very static view of state of the existing constellation at a time point t of the data base request. The disadvantage of this view can become serious pretty fast, as no state of change happens anymore. This can lead, for example, by running into any kind of technical or human failures. Even if no actual problems arise within the pipeline model it can happen, that no update of the existing application constellation will be triggered for an hour or even longer just because of the lack of state chances. Therefore the monitoring of economical states of every single application can just be realized in real-time if status chances would be polled on the data base permanently. By means of that the status information could be customized based on the configuration of the polling interval, but even if the status refresh would be set on a time interval of a second, a real-time monitoring like granted in the CEP-based view cannot be achieved. Besides that no database is able these days to realize such an amount of different requests per second. Hence the database is some kind of “bottleneck” for this case of complex status chances which have to
be monitored in real-time. The absence of highly scalable database features prevents the application of trigger-based databases and therefore cannot be set up to sort out the problem of real-time monitoring of every application.

4.4.2 Differences to CEP-based views

The following CEP-based metrics examples look seemingly OLAP-oriented. Typical OLAP-requests in this context are:

- How many exchanges/cancelations did we have, aggregated within a specified time period $\Delta t$?
- How many exchanges/cancelations could be observed per customer within a specified time period $\Delta t$?
- How many credit applications occurred per section within a specified time period $\Delta t$?
- How many credit applications could be processed per section within a specified time period $\Delta t$?
- Which section featured the most application cancelations within a specified time period $\Delta t$?

These OLAP-based monitoring views offer an introduction in a way of compensating, reactive arrangements. The problem that comes by is, that these actions can only be executed after a certain situations occurred respectively got realized and therefore are much too late. Precautionary, predictive arrangements cannot be realized with this model. For example applicants with a relatively high desired credit amount will not be contacted until the analysis of the logbook entries are done. They will be contacted a few days later by letter which says that the credit application has been canceled and are polled for, if further interest of an instant credit still exists. This is nowadays, where the internet provides very short distances to other business competition companies, an improper way.

CEP-based BAM-views, on the other hand, are able by espousal of certain patterns to visualize which set of problems will arise from the applicant’s user inputs. Hereon it can be reacted in real-time yet before this set of problems occurs. Therefore, an exception situation can be preventive avoided on one hand, while on
the other hand it can be reacted on those situations in real-time. For example it can be reacted instantaneously on a cancelation of a credit application above a certain volume and the customer is being contacted via email, SMS, or by phone at the same time as the pattern for a cancelation is fulfilled. By using CEP-based BAM-views, the current responsible person in charge also can be contacted forthwith via SMS, email, beeper and therefore, in turn, can initiate countermeasures to avoid critical situations.
5 BAM-views and event patterns

Coming events cast their shadows before.

Thomas Campbell

In this chapter the requirements for the integration of a CEP platform into an enterprise environment and the specification of the BAM-views “loss by canceled credit applications” and “performance monitoring of a service” will be discussed. The event format and the generation of BAM-view metrics with a CEP platform will be determined.

5.1 Requirements for the integration of a CEP platform into an enterprise environment

Within the data integration layer of a CEP platform the generated events can be transformed via mappings to the intern data model of a correlation engine. Luckham calls tools, which perform those transformations, adaptors. The adaptors basically use standardized transportation methods such as http, JMS or SMTP or acting as agents by polling on log files, databases or hardware. Thus the data integration layer could contain multiple different adaptors to transform different protocols and data formats [Luckham 2002a, p. 90]. Based on the considerations in chap. 2, events arrive at the CEP platform via JMS. To reduce complexity of data integration there is obviously a need for a consistent and flexible event format. To provide a common way to interact between different systems, IBM and Cisco attempted much endeavors to develop a standardized data format and brought this together under the Web Service Distributed Management (WSDM) standard in 2005 [Cover Pages 2003]. Figure 29 shows how a correlation engine could fit into an enterprise environment.
1. Event Infrastructure A set of services that allow filtering, transmission, and routing of event information between producers and consumers, as well as persistence and access to event data store.

2. Event Unsolicited noteworthy information about a managed resource set for management purposes.

3. Event Producer Producer of events about managed resources that it represents or controls. Events are sent asynchronously. Also known as Event Sources.

4. Customer Environment Customer’s IT environment supporting customer’s business

5. Management Systems (also known as Event Consumers) Consumers of event information aiming management of customer’s businesses and the IT environment supporting them.

6. Event Correlation A set of analytics, and the components that support the analytics: rules language, engines, and tools. These allow the detection of event patterns, automation, filtering, and so forth.

7. Event Format A well-defined, accepted, and structured way of representing event information so that events can be processed.

8. Managed Resources IT and business resources being managed in the customer’s IT environment.

Table 1: Legend for Figure 29 [IBM 2003b]

5.1.1 Web Service Distributed Management

At the 9th of March 2005 WSDM was accepted as an OASIS standard under “Web Services Distributed Management v1.0”. As “Web Service Distributed
BAM-views and event patterns

Management V1.1“, the standard was extended on 1st August 2006 [OASIS 2006a]. Basically WSDM is an extension of WSDL, consists of “Management Using Web Services“ and is extended with „Management Of Web Services“ [OASIS 2006b]. The WSDM standard specifies a common messaging protocol for managed resources and their consumers. Therefore it provides extensions for SOAP to manage components with other Web services. So for example, performance metrics of a service can be used by a business process when deciding which instance of a Web service to use [OASIS 2006b]. To make a component WSDM-enabled, the component has to be represented as a Web service by implementing a WSDL interface with additional manageability capacities [OASIS 2006b]. This can be done for software components such as EJBs, C++ classes, C# classes etc. and for hardware components, such as network components like routers, load balancers, security gateways, electronic devices like televisions, radios and periphery devices like printers or scanners [OASIS 2006b]. WSDM enables resources by describing them as Web services to directly participate in a SOA and business processes. Thus a CEP platform can be fully integrated in a SOA by making their adaptors WSDM-enabled. To communicate with other Web services, the WSDM architecture provides the following three modes:

- A manageability consumer can retrieve management information about the manageable resource. For instance, the consumer can retrieve the current operating status of the manageable resource or the current state of the process running on the manageable resource.
- A manageability consumer may affect the state of a manageable resource by changing its management information.
- A manageable resource may inform, or notify, a manageability consumer of a significant event. This mode of interaction requires the manageability consumer to subscribe on a desired topic to receive events.

For CEP purposes the third modus might be appropriate to let components of an enterprise system inform the correlation engine about activities and status. After applying rules on the information and detecting a potential problem, the resulting action can be as described in modus 2 to manipulate the corresponding resource and to redirect the related business process [OASIS 2006b].
5.1.2 Common Base Event

One part of the WSDM standard is the WSDM Event Format (WEF) which defines a common format for events that contains extension points that allow additional semantic information to be included. IBM studied different event data demonstrations to determine how to represent events in one format and the Common Base Event (CBE) is IBM's initial implementation of WEF [IBM 2003b]. IBM submitted the CBE format to the OASIS WSDM technical committee in late 2003. OASIS ratified the CBE as a component of WSDM as a standard in March of 2005. The CBE model is part of the autonomic computing toolkit which is available on IBM’s developerWorks Web site. It defines a common representation of events, expressed as an XML schema [SOA Magazine 2005], [IBM 2003a]. In an instant credit application, a typical participant in the today’s e-business world, many interconnected systems must work together to perform many of the simple housekeeping activities which are necessary to keep a computing system healthy. Thus, small things like events can have a wide-reaching impact, because they build the base for the communication of these complex systems. Enterprise management and e-business communication, such as performance monitoring, security and reliability, as well as fundamental portions of e-business communications, such as order tracking are based on the fidelity and viability of those events. Missing fidelity can lead to misguided, potentially harmful results, or even results that are fatal to the system. Even simple things such as the formatting of the date and time specified within an event can make an event useless because the sender’s format cannot be understood by the receiver [IBM 2003a]. The CBE picks up these problems and offers a solution which is completely based on open standards. This allows accurate, deterministic and proper management of manageable resources. Therefore it offers fundamental identification properties to detect occurrences within a system [IBM 2003a]:

- The identification of the component which reports a situation.
- The identification of the component which has caused the situation (this may be the same as the component that is reporting the situation).
- The situation itself.
The component which causes a situation is often also the reporter of the situation. For this purpose the CBE definition insists that only the affected component information should be included. The third part about the situation is mandatory and is a base classification, what the event is about. The element allows the description of events with given situation types to ensure a consistent situation description. The CBE element itself also delivers core information, which includes attributes such as the creation time of the event or identifiers for an event. There are also several optional parts of the structure which are flexible enough to support extra functionality and product-specific purposes. These elements can be added several times; with its description each single element represents such product-specific information [IBM 2003a]. Thus events normally represent state changes in a resource, an autonomic manager, or another component involved in the IT infrastructure. In a business process, an event can represent a business milestone or an anomalous business situation. Applying the CBE, state changes like the following can be communicated [IBM 2003b]:

- A component has stopped or started.
- A connection has been established or broken.
- A failure has occurred.
- An amount for a transaction has exceeded an automatic approval threshold.

To ensure consistency of events the CBE is expressed as an XML Schema Definition (XSD). XSD provides the necessary scope for ensuring consistent message content. The CBE schema doesn’t just declare which information a message should convey. Many places in the definition are prescriptive about the actual words that can be chosen to describe a situation or component. Despite ensuring consistency of format and content, the CBE model also encourages completeness. For typical problems of developers, like putting a lot of information into a limited space, the CBE schema delivers mandatory elements guaranteeing that this information will be supplied [IBM 2003a].
5.1.3 Conclusion

Generally WSDM offers all features to integrate a CEP platform into a SOA. A complete realization of the WSDM standard would result in implementing Web services for both sides. On one side the resources of the easyCredit application, which are wanted to be monitored. Additional capabilities have to be implemented, such as identification and manageability capabilities. To realize this, much intrusive work in the existing environment is needed [OASIS 2006b]. Because all tested CEP platforms do not deliver WSDM-enabled adaptors, a custom implementation of an adaptor to make it WSDM-enabled would be necessary. This means that they must not only be able to transform SOAP messages, but they also must have manageable consumer functionalities [OASIS 2006b]. An intermediate solution can be to implement parts of WSDM such as the CBE with the given automation technologies of the given J2EE application server. It is an appropriate way to implement medium-term solutions, like this thesis. For a long term view it’s recommended to integrate the WSDM standard to get a standardized way of managing components.

5.2 BAM-view definition

This paragraph describes the determination of the BAM-view metrics, the corresponding alerts and compensations.

5.2.1 BAM-view: “Loss by canceled credit applications”

The installment credit application easyCredit of TeamBank is a typical mass business application. An enterprise should also be able to have a control of its business processes to successfully run a mass business application. BAM is the foundation to realize this purpose. Therefore it provides transparency of enterprise’s processes based on key data (see para. 3.3). One key data of an instant credit application like easyCredit is the sales volume, which it produces. From this it follows, that a not successfully completed credit application does not contribute
to the key data sales volume. To optimize the sales volume output of the easyCredit application, there is need for real-time detection of canceled credit applications.

5.2.1.1 **Enterprise cockpit solutions**

Defining a view called “loss by canceled credit applications” the first question posed is “What is a real loss by a canceled application?”. Looking at loss from the business view, a loss could be defined either as a sales shortfall or as a loss of profits. Business volume of a credit is generated by the yield on interest of a credit, the profit results of the subtraction of all costs from the business volume. After analysing the online dialog of easyCredit, just the credit amount could be detected as a consistent measurement value for all applicants, because values such as payback or yield on interest are calculated of the inputs of an applicant and just appear at the end. Thus the credit amount can only be taken as a consistent indicator for a potential sales shortfall. So in this purpose the loss can be defined as “the credit amount of an application, which was not completed successfully”. But there also exist special cases just related to the customer TeamBank. One case is a credit for a person who is self-employed, because TeamBank just offers credits for people which are in an employment. Another case is a “red” credit decision. TeamBank classifies credit applications in three different risk categories, which are “green”, “yellow” and “red”. This classification is based on several statistical or other methods, such as expenses/earnings relation, address validation, Schufa (credit protection agency) rating request etc. If the person is after analyse and validation of his inputs founded to be reliable to repay the credit, his credit application gets status “green”. A person applying a credit with not having sufficient earnings to repay the credit within the life of the loan or other suspicious criteria, his credit application gets status “red”. The state “yellow” is reserved for applications of people with tight credit decisions or with unclear application entries, which can be a wide-spread name leading to an unclear Schufa rating. After this classification, a credit application is accepted, when it gets status “green”. If it gets status “red”, it will be denied and if it’s “yellow”, the decision
of acceptance or deny will be done by an employee of the TeamBank [Ammon 2006b].

The third case is about people, which visited the easyCredit website without even making application inputs. This can have different reasons:

- The person just wants information about easyCredit.
- The person just wants information about TeamBank.
- The person hit the website accidently.

So it’s obvious, that this person was not interested to get a credit and went on the website for other purposes.

To decide whether these three cases present losses for TeamBank, firstly the interests of TeamBank have to be resolved. Therefore the target group has to be defined. The TeamBank provides credits for reliable end consumers which are in an employment or receive other continuous earnings. People being self-employed or getting insufficient earnings do not belong to this customer group. Another group are people, not seriously applying for credits. Including those three groups would result in a multitude of the real loss, the TeamBank wants to detect. This follows of the fact, that those people are not within the target group of the TeamBank and wouldn’t get a credit at all. Hence, an exclusion of those groups is necessary to get an appropriate and accurate result. Resuming this, a loss by a canceled credit application can be defined as “the credit amount of a desired application of a person who started an application and would get a credit, when he would finish his application”. An additional consideration in the design phase of BAM-views is about the information, which has to be visualized. Basically the information shall offer a detailed view on a topic at a glance. An additional feature should be an evaluation of the reasons for emerged situations. Therefore the information must be defined, aggregated and classified in operational data and reasons. Because the aggregated information is related to an observation period, the period must also be defined.

In purpose of monitoring day-to-day business, a day or 24 hours is an appropriate observation period to get actual situation related information, although the observation period should also be configurable. The terminology continuous observation poses new questions such as what is a day or when does a day start.
To answer these questions, first an observation period has to be differed in a slicing or a fixed period. A fixed period has a predefined start and end time like starting daily at 0 am and ending next day at 0 am. Therefore a start time has to be defined, which delivers the most relevant information. If the period starts at 0 am, the available information is related to the current day. Starting at 8 am implies, that the available information is related to the current business day.

One problem of a fixed period is that continuous aggregation functions like summarizations are not applicable for fixed periods, especially when they are long, because each result is based on a different observation period and cannot be related to another result. So they bring small results at start time of a period and big results at the end time of a period like for example at 11 am, the summarized loss by canceled credit applications is 50.000 € and 2 hours later at 1 pm the loss is 100.000 €. This leads to imprecise and inadequate information. Another problem is that not all occurrences can be detected by humans. The following scenario shall explain the problem:

A responsible person for credit applications starts at 8 am and finishes at 5 pm. A credit application with the credit amount of 75.000 € is canceled at 10 pm because of technical problems and would normally be treated with a regarding email from a responsible person for credit applications. The observation period ends at 12 am and a new observation period begins. On the next day, the responsible person isn’t able to notify the canceled application, because the canceled application is not within the observation period of the current day.

This problem can be prevented by either defining the start position and end position on the end of a working day, which enables a person to notify all happenings within his normal working time or with automation techniques such as keeping historical data. A slicing observation period solves both of the above described problems by continuously applying a period backwards from the current timestamp. So for example for an observation period of a day, every monitored result is based on the last 24 hours. After passing the observation period once, continuous aggregation functions are always based on the same period. Supposing that a person looks on the enterprise cockpit at least once within 24 hours, the second problem of not detecting occurrences is also solved. Resuming those
considerations the most practicable way is applying slicing windows. Based on a defined observation period, data have to be collected to present a comprehensive view on the easyCredit application. Therefore a definition of metrics like the following is necessary:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Observation period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canceled applications</td>
<td>Number of all canceled applications within the observation period</td>
<td>1 day</td>
</tr>
<tr>
<td>Applications</td>
<td>Number of all applications within the observation period</td>
<td>1 day</td>
</tr>
<tr>
<td>Relation of canceled applications to all applications</td>
<td>(Number of canceled applications *100)/ number of applications</td>
<td>1 day</td>
</tr>
<tr>
<td>Loss by canceled</td>
<td>Accumulation of credit amounts, generated by canceled applications within the observation period</td>
<td>1 day</td>
</tr>
<tr>
<td>Relation of loss to credit amounts</td>
<td>Relation of accumulative loss by canceled applications to accumulative credit amounts by successfully applied credits based on the observation period</td>
<td>1 day</td>
</tr>
<tr>
<td>Average loss</td>
<td>Average credit amount, canceled application generate within the observation period</td>
<td>1 day</td>
</tr>
<tr>
<td>Maximum loss</td>
<td>Highest credit amount of a canceled application within the observation period</td>
<td>1 day</td>
</tr>
</tbody>
</table>

Table 2: Metrics for the easyCredit process

The above described metrics give a detailed view on the current status of the easyCredit application, but to determine reasons for the entered status, there have to be defined other metrics. The following key values give more detailed information to detect, why a situation occurred:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Observation period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average duration</td>
<td>Average duration of successfully applied credits</td>
<td>1 day</td>
</tr>
</tbody>
</table>

Table 3: Average duration metric for the easyCredit process
### Table 4: Application step metric for the easyCredit process

<table>
<thead>
<tr>
<th>Metric</th>
<th>Exit point: application step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Views the application step, where most applications were canceled</td>
</tr>
<tr>
<td>Metric</td>
<td>Number exit point: application step</td>
</tr>
<tr>
<td>Description</td>
<td>Number of the application steps, where most applications were canceled</td>
</tr>
<tr>
<td>Metric</td>
<td>Relation: exit point application step to canceled applications</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

\[
\frac{\text{Number exit point: application step} \times 100}{\text{number of all canceled applications within the observation period}}
\]

<table>
<thead>
<tr>
<th>Observation period</th>
<th>1 day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance</td>
<td>Indicator for technical problems or inadequate user guidance and its impact on canceled applications</td>
</tr>
</tbody>
</table>

### Table 5: Session bean metric for the easyCredit process

<table>
<thead>
<tr>
<th>Metric</th>
<th>Exit point: session bean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Views the most affected session bean, where most applications were canceled</td>
</tr>
<tr>
<td>Metric</td>
<td>Number exit point: session bean</td>
</tr>
<tr>
<td>Description</td>
<td>Number of the session beans, where most applications were canceled</td>
</tr>
<tr>
<td>Metric</td>
<td>Relation: exit point session bean to canceled applications</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

\[
\frac{\text{Number exit point: session bean} \times 100}{\text{number of all canceled applications within the observation period}}
\]

<table>
<thead>
<tr>
<th>Observation period</th>
<th>1 day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance</td>
<td>Indicator for technical problems and its impact on canceled applications</td>
</tr>
</tbody>
</table>

### Table 6: Application server metric for the easyCredit process

<table>
<thead>
<tr>
<th>Metric</th>
<th>Exit point: application server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Views the application server, where most applications were canceled</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Number exit point: application step</td>
<td>Number of the application server, where most applications were canceled</td>
</tr>
<tr>
<td>Relation: exit point application server to canceled applications</td>
<td>(Number exit point: application server *100)/ number of all canceled applications within the observation period</td>
</tr>
</tbody>
</table>

**Observation period**
1 day

**Significance**
Indicator for technical problems or high server load and its impact on canceled applications

Table 6: Application server metric for the easyCredit process

Besides the detection of reasons, BAM also gives the user the opportunity to prize this aggregated view. Therefore the so called drill-down which is normally a functionality of OLAP, allows the presentation of the causing events and a new aggregation based on desired filters to break it in desired pieces. Those filters can be based on time intervals, for example all credit applications of the last hour, or on values, for example all credit applications made on application server x. To provide a better business insight for users the drill-down functionalities must cover all possible perspectives on an abstract circumstance. In this purpose the functionality contains:

- Canceled applications caused by an application server.
- Canceled applications caused by a JSP.
- Canceled applications caused by a session bean.
- Canceled applications within the last x minutes.

Linking those filters allow a user to restrict abstracted data to a narrow perspective to determine causes for entered situation. This allows decision making based on key data drilled down to the cause. The visualisation of results is a first step to inform operational departments about business related topics. But besides this, a BAM platform should also be able to notify responsible persons of unexpected occurrences, even if they are not in front of the monitor, or perform compensation measurements.
5.2.1.2 Alerts

Despite visualization of information, a CEP based BAM can also provide VIRT in terms of alerts (see para. 3.3). Therefore CEP platforms provide output adapters or APIs to deliver information to a related decision maker. The implementation of COIs takes place in the CEP platform itself and the resulting events will be routed in real-time via email or SMS to the person, who subscribed to the desired topic. In business purposes, COIs might be based on metrics passing thresholds, so for example a sales is just interested, if the not accepted offers reach a certain value. Understanding the VIRT concept is the condition to see the impact of thresholds and their correct determination to an adequate alerting system. Applying too low thresholds leads to too many undesired alerts. The receivers are glutted by a deluge of low-value data and consumed by attendant low-value tasks [Hayes-Roth 2004]. Too high thresholds lead to delayed information which makes the idea of real-time lapsed.

The classification of a situation delivers a suitable way to react on unexpected situations. Therefore situations can be divided in different escalation classes, like for example “green” indicating that a metric is in the predefined interval, “yellow” indicating that a metric nears a predefined interval limit and “red” indicating that a metric passed a predefined interval limit [Ammon 2006b].

The correct adjustment of a threshold will normally be made via already existing empirical values from the operational department. This threshold should be already analyzed in the test phase intensively by examination of the resulting alerts. By defining thresholds for the metrics defined for the particular BAM-views, such alerts can be realized. The following COIs give such an opportunity:

- Yellow state threshold for loss by canceled applications was passed.
- Red state threshold for loss by canceled applications was passed.
- Yellow state threshold for the number of canceled applications was passed.
- Red state threshold for the number of canceled applications was passed.
• Yellow state threshold for relation of canceled applications to entered applications was passed.
• Red state threshold for relation of canceled applications to entered applications was passed.
• Application with a credit amount above a defined limit was canceled.

To allow a preferably fast decision based on an alert, the included information should also supply possible reasons respectively recommendations. Enriching the alert with the reasons, defined for the enterprise cockpit, can provide a fast explanation for the entered state.

• Most applications were canceled at application step x: Technical reasons or inadequate user guidance
• Most applications were canceled on application server y: Technical reasons or high server load
• Most applications were canceled at session bean z: Technical reasons.

So an alert including the described information could look like Figure 30:

<table>
<thead>
<tr>
<th>Situation</th>
<th>Loss of canceled applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>Yellow state threshold for relation of canceled applications to entered applications was passed</td>
</tr>
<tr>
<td>Threshold</td>
<td>200.000 €</td>
</tr>
<tr>
<td>Current Value</td>
<td>220.000 €</td>
</tr>
<tr>
<td>Reason</td>
<td>Exit point session bean: CalculatorBean</td>
</tr>
<tr>
<td>Recommendation</td>
<td>Technical Reasons – please call: 555-1234 system admin or 555-6273 operational department</td>
</tr>
</tbody>
</table>

Figure 30: Exemplary business alert

5.2.1.3 Appropriate compensations

An alert or notification of a problem itself brings no benefit without a compensating action. BAM enables enterprises to perform such measurements either manually or automatically to resolve lacks in processes. By automatically starting external compensation processes or scripted actions, a company is able to
save money by eliminating labor-intensive interventions [TIBCO 2005]. Utilizing
BAM, especially compensating actions via a CEP platform needs both, an
understanding of events and an understanding of workflows: “Which pattern
triggers a change in a business process and how to change a business process with
the given application programming interfaces (API) of a CEP platform?” The
platforms offer standardized adapters such as email, JMS or http, but there is still
much customization work to do to let a platform successfully change processes. A
standard like WSDM leads to a more efficient involvement of CEP into a business
process (see para. 5.1.2).

With the available email adaptor a compensation action of canceled applications is
to send an email to the affected applicant. Future works like an integration of CEP
via WSDM would make measurements like an adjustment of the interest rate
based on the number of applicants possible.

5.2.2 BAM-view: “Performance monitoring of a service”

The effort and hype of SOA is also generated of the new opportunity through
measurability of individual components within processes. From this, various tools
like Wily’s SOA-Manager [Wily 2007] or Mercury’s Mercury Service Test
[Mercury 2006] developed. The problem of these products is, that only the service
itself can actually be regarded and not the effects on processes. Thus the
responsible person needs additional process-referred knowledge to interpret the
results, what makes decisions more difficult. The present products usually offer a
too technical view on the problems and make it rather more difficult than simpler
for operational divisions to find problem solutions. By combination and
unification of the technical view and the business view, information technical
causes for lacks of processes can be determined. On the other hand impact on
business processes caused by failures within the IT can also be detected. The
easyCredit infrastructure is basically developed with EJB components. The
monitoring of these components grants an exact view on the infrastructure of the
easyCredit application. Additionally mechanisms must be inserted which make a
reference to related processes possible.
5.2.2.1 Enterprise cockpit solutions

Component performance measurement can be done on different wise, it can either be determined with absolute numbers or with relative percentages. The as-is state of TeamBank, described in the pipeline model, is a monitoring of components’ states with absolute numbers (see para. 4.2). The idea of the pipeline model is to measure the queues waiting for the processing of a service [Ammon 2006b].

A different approach is to monitor those requests which didn’t finish within a specified timeframe, like for example a service didn’t finish within the average processing time of three seconds. This can be rendered more precisely by escalating limitations, so average processing time is the measurement base 1 and maximum processing time is the next measurement base 2. A relation of those “slow” calls to all incoming calls allows a more precise determination of a service’s performance, then for example, service x did not complete 20% of all service calls within the average duration and 1% did not reply within the maximum duration.

Reconsidering the comments in para. 5.2.1, those determinations should be based on an observation period applied as slicing windows. The design of a component monitor should allow parameterization for each individual component to create a reusable monitoring service for all components of an enterprise. An exact adjustment of the three following parameters is fundamental for an accurate monitoring of a component:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observation period</td>
<td>The collected data is based on the observation period</td>
</tr>
<tr>
<td>Limitation yellow</td>
<td>Average processing duration of a component call + standard deviation as limitation for service call executed on time</td>
</tr>
<tr>
<td>Limitation red</td>
<td>Maximum processing duration of a component call as limitation for completed service calls</td>
</tr>
</tbody>
</table>

Table 7: Parameters for a component monitor

Based on those parameters the following metrics can be determined:
### Table 8: Request metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Number of all service calls within the observation period</td>
</tr>
<tr>
<td>Observation period</td>
<td>Parameter: Observation period</td>
</tr>
</tbody>
</table>

### Table 9: Completed request metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Completed requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Number of completed service requests within the observation period</td>
</tr>
<tr>
<td>Observation period</td>
<td>Parameter: Observation period</td>
</tr>
</tbody>
</table>

### Table 10: Yellow request metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Yellow requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Number of all service calls which didn’t finish within the “limitation yellow”, based the observation period</td>
</tr>
<tr>
<td>Limitation</td>
<td>Parameter: Limitation yellow</td>
</tr>
<tr>
<td>Metric</td>
<td>Percentage of yellow requests</td>
</tr>
<tr>
<td>Description</td>
<td>Percentage of the number of yellow requests to all incoming requests based the observation period</td>
</tr>
<tr>
<td>Observation period</td>
<td>Parameter: Observation period</td>
</tr>
</tbody>
</table>

### Table 11: Red request metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Red requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Number of all service calls which didn’t finish within the “limitation red”, based the observation period</td>
</tr>
<tr>
<td>Limitation</td>
<td>Parameter: Limitation red</td>
</tr>
<tr>
<td>Metric</td>
<td>Percentage of red requests</td>
</tr>
<tr>
<td>Description</td>
<td>Percentage of the number of red requests to all incoming requests based the observation period</td>
</tr>
</tbody>
</table>
Observation period
Parameter: Observation period

Table 11: Red request metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Minimum duration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The quickest completion of a service request within the observation period</td>
</tr>
<tr>
<td>Metric</td>
<td>Average duration</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average duration of a service request completion within the observation period</td>
</tr>
<tr>
<td>Metric</td>
<td>Maximum duration</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The slowest completion of a service request within the observation period</td>
</tr>
</tbody>
</table>

Observation period
Parameter: Observation period

Table 12: Processing duration metric

5.2.2.2 Alerts

Adapted from the considerations about VIRT in para. 3.3 and 5.2.1.2, a definition of valuable information about a component is necessary. Therefore thresholds will be applied on the above described metrics. An alert will be generated by passing a threshold. To create reusable component monitors, those thresholds should be parametrizable.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold: yellow – absolute</td>
<td>Limitation for the number of yellow service requests</td>
</tr>
<tr>
<td>Threshold: yellow – relative</td>
<td>Limitation for the percentage of yellow service requests to incoming service requests</td>
</tr>
<tr>
<td>Threshold: red – absolute</td>
<td>Limitation for the number of red service requests</td>
</tr>
<tr>
<td>Threshold: red – relative</td>
<td>Limitation for the percentage of red service requests to incoming service requests</td>
</tr>
<tr>
<td>Threshold: average duration</td>
<td>Threshold for the average processing duration of a component. The value should be close to the parameter “Limitation Yellow”</td>
</tr>
<tr>
<td>Threshold: maximum duration</td>
<td>Threshold for the maximum processing duration of a component. The value should be close to the parameter “Limitation Red”</td>
</tr>
</tbody>
</table>

Table 13: Threshold parameters
The generated alert includes the metrics described above to offer a comprehensive view on the component’s status. The difference to existing products brings the link to related business processes. It is the information “affected processes”, which displays the related processes and the impact on the enterprise’s credits.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Component overload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>CalculatorBean</td>
</tr>
<tr>
<td>Detection</td>
<td>Threshold “relation of yellow to incoming” was passed</td>
</tr>
<tr>
<td>Threshold</td>
<td>20 %</td>
</tr>
<tr>
<td>Current Value</td>
<td>20.2 %</td>
</tr>
<tr>
<td>Affected Processes</td>
<td>Process: online dialog easyCredit</td>
</tr>
<tr>
<td></td>
<td>Current number: 812 instances Current value: 430.000 €</td>
</tr>
<tr>
<td></td>
<td>Process: test environment easyCredit</td>
</tr>
<tr>
<td></td>
<td>Current number: 230 instances Current value: 140.000 €</td>
</tr>
<tr>
<td>Recommendation</td>
<td>Technical reasons – please call: 555-1234 system admin</td>
</tr>
<tr>
<td></td>
<td>or 555-1456 operational department</td>
</tr>
</tbody>
</table>

*Figure 31: Exemplary technical alert*

5.2.2.3 *Appropriate compensations*

Because lack of component performance may have different causes, there can be different measurements done to restore a properly running system. Those measurements can be applied either to hardware or to software. Too much traffic on servers, a typical hardware related problem, which may result in a lack of scalability can be solved with compensation measurements such as the insertion of additional servers. A software related measurement could be replacing a component with either a previous versioned component or a backup of the current one. Therewith software problems like not correctly functioning components can be avoided. As mentioned in para. 5.2.1.3, automated replacing or adding of hardware or software is quite difficult and can be improved by standards like WSDM (see para. 5.1.1).
5.3 Event design

The following paragraph describes the determination of the needed event attributes and the selection of the CBE elements which transport the event attributes.

5.3.1 Determination of needed information in an event to build the views

As mentioned in para. 2.3 and 2.4, the event generation takes place via interceptors and servlet filters sending JMS messages to a topic. Because those request level interceptors are just invoked on EJB invocations, in this case session beans, and the servlet filters are just invoked by http requests, in this case from JSPs, the tracing of a business process has to be done via those generated events. The analysis of the interceptor and servlet filter given possibilities of collecting data results in following event attributes (see para. 2.4):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process identifier</td>
<td>The http session thread id is the reference of all requests and responses to a credit application, respectively a business process instance</td>
</tr>
<tr>
<td>Event identifier</td>
<td>Unique identifier of an event as reference for other events</td>
</tr>
<tr>
<td>Event creation time</td>
<td>The creation time of an event, used to determine time intervals and temporal orders</td>
</tr>
<tr>
<td>Component identifier</td>
<td>Uniquely identifies a component, such as a session bean, within an enterprise</td>
</tr>
<tr>
<td>Type of component</td>
<td>The classification of the component. In this case it’s used to differ between “StatelessSessionBean” and “JavaServerPage”</td>
</tr>
<tr>
<td>Method identifier</td>
<td>Because of the fact, that most of the monitored components are session beans, an identification of their methods is also necessary</td>
</tr>
<tr>
<td>Additional identifier</td>
<td>Because a method of a component can occur multiple times within a process instance, an identification of request/response pair cannot be detected just by a component identifier and a method identifier. Therefore an additional identifier is necessary. This has to be unique within one process instance, so for example an integer, incrementing at component calls</td>
</tr>
<tr>
<td>Application server</td>
<td>The identifier of the physical machine of the event occurrence</td>
</tr>
<tr>
<td>Expression type of</td>
<td>The classification of the component expression, in this</td>
</tr>
</tbody>
</table>
application server | case “ip4”
---|---
Status | An event classification, what event reports, like method response or JSP request or JSP response
Business Data | Operational data, which the process generates, such as credit amount

*Table 14: Event attributes*

5.3.2 M **apping of information into the event format**

After the determination of needed elements, a transformation of the attributes is made. To improve the event’s quality, a second look on the event under the reference to the CBE properties is made [IBM 2003a].

<table>
<thead>
<tr>
<th>Property name</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Necessary for version management of the event, e.g. after enrichment and change of mutable elements</td>
</tr>
<tr>
<td>localInstanceId</td>
<td>Useful for performance improvement in distributed CEP systems, respectively their historical data, but not in this case</td>
</tr>
<tr>
<td>globalInstanceId</td>
<td>Attribute: Event identifier</td>
</tr>
<tr>
<td>creationTime</td>
<td>Attribute: Event occurrence time</td>
</tr>
<tr>
<td>severity</td>
<td>Not useful because the CEP platform is responsible for the detection, whether an event is important or not</td>
</tr>
<tr>
<td>priority</td>
<td>Not useful because events are processed when they appear, not depending on their priority</td>
</tr>
<tr>
<td>reporterComponentId</td>
<td>According to para. 5.1.2, the reporterComponentId is not needed, because the sourceComponent sends events by inserted interceptors respectively servlet filters.</td>
</tr>
<tr>
<td>sourceComponentId</td>
<td>According to the description of reporterComponentId this attribute is necessary to get the information, which software respectively hardware component is responsible for the event</td>
</tr>
</tbody>
</table>
| situation             | Attribute: Status
The element is needed to describe the status change of the event with the following keywords:
StartSituation
StopSituation |
| contextDataElements   | This element is not required because there are no useful information which can be transported |
BAM-views and event patterns

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Attribute: Application server</td>
</tr>
<tr>
<td>locationType</td>
<td>Attribute: Expression type of the application server. Done with the reserved keyword “IPV4”</td>
</tr>
<tr>
<td>application</td>
<td>A useful property to identify the application itself, it should be unique within the enterprise. In this case “easyCredit”</td>
</tr>
<tr>
<td>executionEnvironment</td>
<td>Also useful to identify the software platform the application is running on. Because all application servers are JBoss application servers, there is no usage.</td>
</tr>
</tbody>
</table>
| component           | Attribute: Component identifier  
                      Session bean: Name of the session bean  
                      JSP: Name of the JSP |
| subComponent        | In the author’s view the subcomponent property is useless at all because every component can have multiple subcomponents. Declaring an upper component would be more constructive. |
| componentIdType     | Because this element is required, the declaration is “ServiceName”          |
| instanceId          | Beside processId and threadId the instanceId could                          |

Table 15: CBE elements

The next tables give a more detailed view on those properties which are complex types. Table 16 describes the source component with the complex type ComponentIdentification.
be taken to identify a process instance, processing a credit application.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>processId</td>
<td>Beside instanceId and threadId the processId could be taken to identify a process instance, processing a credit application.</td>
</tr>
<tr>
<td>threadId</td>
<td>Attribute: Process identifier The threadId was taken because sessionthreadId of an http session identifies the process.</td>
</tr>
<tr>
<td>componentType</td>
<td>Attribute: Type of the component</td>
</tr>
</tbody>
</table>

Table 16: ComponentIdentification elements

The following table shows the complex type situation. The situation describes the state change, an event is reporting.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>situationType</td>
<td>This complex type allows a more exact description of a situation.</td>
</tr>
<tr>
<td>categoryName</td>
<td>This property defines which situation occurred, in this case, the method request is declared with: StartSituation The method response is declared with: StopSituation</td>
</tr>
</tbody>
</table>

Table 17: Situation elements

The two situation types define if an event describes either a method request or a method response. The first table shows the properties of StartSituation, the second table shows the properties of StopSituation.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>successDisposition</td>
<td>Declared with the key word: SUCCESSFUL</td>
</tr>
<tr>
<td>situationQualifier</td>
<td>Declared with the key word: START COMPLETED</td>
</tr>
<tr>
<td>reasoningScope</td>
<td>Declared with the key word: INTERNAL</td>
</tr>
</tbody>
</table>

Table 18: Situation type: StartSituation

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>successDisposition</td>
<td>Declared with the key word: SUCCESSFUL</td>
</tr>
<tr>
<td>situationQualifier</td>
<td>Declared with the key word: SUCCESSFUL</td>
</tr>
</tbody>
</table>
Table 19: Situation type: StopSituation

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>successDisposition</td>
<td>This property specifies the success disposition of an operation of a situation that caused the situation to be reported. The successDisposition is a string with the following set of values: SUCCESSFUL UNSUCCESSFUL This is a REQUIRED property and once it is set it MUST NOT change. The string length for this property MUST NOT exceed 64 characters.</td>
</tr>
<tr>
<td>situationQualifier</td>
<td>This property specifies the situation qualifiers that are representation of the parameters necessary to describe the situation. The situationQualifier is a string with the following set of values: STOP INITIATED ABORT INITIATED PAUSE INITIATED STOP COMPLETED This is a REQUIRED property and after it is set it MUST NOT change. The string length for this property MUST NOT exceed 64 characters.</td>
</tr>
</tbody>
</table>

Table 20: Complex type RequestSituation [IBM 2003a]

The RequestSituation is not applicable because the situation qualifiers don’t allow describing a situation like “method request started”. An additional situation qualifier like START INITIATED or START COMPLETED would result in a better described situation. A method request could then be marked as RequestSituation and the situationQualifier as START COMPLETED. The method response could be marked with a RequestSituation and the situationQualifier as STOP COMPLETED. The following three easyCredit specific extensions defined as ExtendedDataElements provide the necessary
information to build the views. The following table shows the method, which was called. In the author’s view, the property method should normally be included in the complex type ComponentIdentification.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>method</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
</tr>
<tr>
<td>values</td>
<td>The name of the called method</td>
</tr>
<tr>
<td>hexValue</td>
<td>Null</td>
</tr>
<tr>
<td>children</td>
<td>Null</td>
</tr>
</tbody>
</table>

Table 21: ExtendedDataElement expressing the method

The following table shows the already mentioned additional identifier.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>extId</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
</tr>
<tr>
<td>values</td>
<td>A value which is identical in request and response events.</td>
</tr>
<tr>
<td>hexValue</td>
<td>Null</td>
</tr>
<tr>
<td>children</td>
<td>Null</td>
</tr>
</tbody>
</table>

Table 22: ExtendedDataElement expressing the additional identifier

The operational data, in this case the credit amount, credit decision and status of employment, is shown in the next table.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>creditAmount</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
</tr>
<tr>
<td>values</td>
<td>The credit amount, taken from the easyCredit application</td>
</tr>
<tr>
<td>hexValue</td>
<td>Null</td>
</tr>
<tr>
<td>children</td>
<td>Null</td>
</tr>
</tbody>
</table>

Table 23: ExtendedDataElement expressing the credit amount

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>creditDecision</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
</tr>
<tr>
<td>values</td>
<td>The credit decision, classified by the easyCredit application, in this case “green”, “yellow”, “red”</td>
</tr>
<tr>
<td>hexValue</td>
<td>Null</td>
</tr>
<tr>
<td>children</td>
<td>Null</td>
</tr>
</tbody>
</table>
Table 24: ExtendedDataElement expressing the credit decision

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Deployment</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
</tr>
<tr>
<td>values</td>
<td>Boolean which describes a person is whether deployed or not</td>
</tr>
<tr>
<td>hexValue</td>
<td>Null</td>
</tr>
<tr>
<td>children</td>
<td>Null</td>
</tr>
</tbody>
</table>

Table 25: ExtendedDataElement expressing the status of employment

5.3.3 Summary

Based on the considerations above, the tracing of the easyCredit application and the determination of patterns and occurrences is possible with the event shown in the following table.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>xsd:string</td>
</tr>
<tr>
<td>globalInstanceId</td>
<td>xsd:ID</td>
</tr>
<tr>
<td>creationTime</td>
<td>xsd:dateTime</td>
</tr>
<tr>
<td>sourceComponentId</td>
<td>cbe:ComponentIdentification</td>
</tr>
<tr>
<td>situation</td>
<td>cbe:Situatin</td>
</tr>
<tr>
<td>extendedDataElements</td>
<td>cbe:ExtendedDataElement[]</td>
</tr>
</tbody>
</table>

Table 26: CBE with selected elements

5.4 Rule design

This paragraph examines the description of the rules and corresponding aggregations, which build the basis for the particular BAM metrics. For a better understanding the pattern description is in an almost natural language.

5.4.1 BAM-view: “Loss by canceled credit applications”

In order to deliver aggregated information in a BAM-view called “Loss by canceled credit applications”, obviously a canceled application has to be detected first. Credit cancelations can be determined in different ways. Therefore all possible scenarios should be determined to guarantee a near-time determination of
a canceled application. So, in order to define valid scenarios, knowledge about the particular process is needed.

- Time exceeding of the beginning/end of a credit application.
- Time exceeding of a dialog step.
- Time exceeding of a system answer.

In the following, the shown scenarios will be described, whereby „time exceeding of a system answer“ is described in para. 5.4.2, since this rule is also relevant for the component monitor.

### 5.4.1.1 Time exceeding of the beginning/end of a credit application

Processes such as a credit application are normally characterized with a start and end point, whereby they take the start event, perform actions and result in an end event. The start of a credit application is defined by describing the first called session beans in event S. On occurrence of an event, having those attributes, there has to be an examination, whether this session bean starts a new credit application or is called again within an existing one.

**Event S:**

```
component = “esys.ejb/workflow.InternetGastLocal“ AND
componentType = “StatelessSessionBean“ AND
method = “getProfil“ AND
situation = “StartSituation“
```

The definition of a credit application end point is made by the description of the last called session bean in event E.

**Event E:**

```
component = “esys.ejb/workflow.InternetGastLocal“ AND
componentType = “StatelessSessionBean“ AND
method = “antragEntsperren“ AND
situation = “StopSituation“
```

If the defined end point E is reached after the occurrence of the defined start point S within a given time window T, the application was successful. By the point of reference `threadId`, which marks a credit application clearly, the scenario “S followed by E” can be determined. If E meets S within the period T, whereby they fulfil the condition `S.threadId=E.threadId`, the rule is fulfilled.
ON: S followed by E WITHIN T  
Condition: S.threadId = E.threadId  
Action: Credit application end is reached

With a negation “!” of event E a not reached credit application can be described. By the point of reference threadId, the scenario “S followed by !E” can be described, which determines canceled applications. If E does not meet S within time window T with the condition S.threadId=E.threadId, the rule is fulfilled.

ON: S followed by !E WITHIN T  
Condition: S.threadId = E.threadId  
Action: Credit application end is not reached

5.4.1.2 Time exceeding of a dialog step

A successful application on the easyCredit website is reached by filling in several forms or dialog steps. Those dialog steps are represented by JSPs. Through a JSP call by event O and defining of an exit of a dialog step with a further JSP call defined by event N, the scenario “O followed by N” can be described, whereby activities can be determined within a JSP request or response.

Event O:
componentType = “JavaServerPage” AND  
situation = “StartSituation”

Event N:
componentType = “JavaServerPage”  
situation = “StopSituation”

An additional consideration is that an applicant can also jump back on a previous dialog step, the resulting request P has also an attribute situation = “StartSituation”. An actualization A of the page can take place. In these two cases there is no generated response of a JSP.

Event P:
componentType = “JavaServerPage”  
situation = “StartSituation”
Event A:

\[
\text{componentType} = \text{"JavaServerPage" AND situation} = \text{"StartSituation"}
\]

The determination of user activities takes place either by calls of new pages, actualization of a page or jump backs on previous pages. With the described events, a scenario ,O followed by N“ or “O followed P” or “O followed by A” can be described.

\[
\text{ON: (O followed by N) OR (O followed by P) OR (O followed by A) WHITHIN T}
\]

\[
\text{Condition: (O.threadId} = \text{N.threadId OR O.thread.ID} = \text{P.threadId OR O.thread.ID} = \text{A.threadId)AND}
\]

\[
(O.component} = \text{N.component OR O.component} = \text{P.component OR O.component} = \text{A.component})
\]

\[
\text{Action: User activity}
\]

By negation of the events following O, a stop on a page can be determined.

\[
\text{ON: (O followed by !N) OR (O followed by !P) OR (O followed by !A) WHITHIN T}
\]

\[
\text{Condition: (O.threadId} = \text{N.threadId OR O.thread.ID} = \text{P.threadId OR O.thread.ID} = \text{A.threadId)AND}
\]

\[
(O.component} = \text{N.component OR O.component} = \text{P.component OR O.component} = \text{A.component})
\]

\[
\text{Action: No user activity}
\]

A strong simplification of this rule can take place on the assumption that within a defined time interval T a user activity U, expressed by a JSP request or response, is followed by a next user activity NU, expressed by a JSP request or response. These two events describe calls of new pages, actualization of a page or jump backs on previous pages. The only restriction is that the user activity U doesn’t have the attributes, which mark the successful finish of an application, since this activity is the last one of an application. Here the events U and NU can be described with \text{componentType} = \text{"JavaServerPage"}.

The according rule looks like:

\[
\text{ON: U followed by NU WHITHIN T}
\]

\[
\text{Condition: (U.threadId} = \text{NU.threadId) AND}
\]

\[
(U.component} != \text{"vertragsUnterlagen_grau" AND U.componentType} != \text{"JavaServerPage" AND U.situation} != \text{"StopSituation")}
\]

\[
\text{Action: User activity}
\]
By negation of event NU, not existing user actions can be detected.

ON: U followed by !NU WITHIN T
Condition: (U.threadId = NU.threadId) AND
(U.component != “vertragsUnterlagen_grau” AND
U.componentType != “JavaServerPage” AND
U.situation != “StopSituation”)
Action: No user activity

5.4.1.3 Additional Rules

On the assumption that the necessary technologies exist in order to determine a leaving from a website, appropriate events directly could be generated at such a user activity. Tools, providing website statistics, already have functionalities, which determine the change of Web addresses. By the extension of these functionalities by the generation of CBEs, events could be generated directly in the case of an application cancelation. Thus, also such scenarios for the recognition of aborts would be conceivable.

- Website left
- Cancel button or browser’s close button pressed

In the following, the possible rules for these scenarios are explained. The rule for „Website left“ is fulfilled by event L, whereby event L is described as:

```
situation="ReportSituation"
reportCategory="Trace"
ExtendedDataElementName="Website left"
ExtendedDataElementValue="www.otherwebsite.de"
```

By closing the browser, an event C like the following could be generated:

```
situation="ReportSituation"
reportCategory="Trace"
ExtendedDataElementName="CloseButton pressed"
```

The corresponding rule could be:

ON: L OR C
Action: User activity: Website left

5.4.1.4 Restrictions

Since several rules define a canceled application, it must be guaranteed that only the first arrival of one of these rules determines one canceled application. Ignoring this step would result in a multitude of the detected application cancelations, so
for example firstly the rule “to exceeding the time system time“, then the rule “exceeding the time page call“ and then the rule “exceeding the time beginning/end request“ is fulfilled. The result would be that each of these rules would define a canceled application and thus three canceled applications would be determined. By referencing on the threadId, this can be prevented. The result R generated from the rules is examined, whether a result NR is not already present by another rule. If this case occurred, result R will be rejected. This can be determined by the following rule:

ON: !NR.threadId, R.threadId WHITHIN T
Condition: NR.threadId=R.threadId
Action: Canceled application

In the next step, the resulting magnitude will be restricted by canceled applications which are relevant for the TeamBank. Therefore, those canceled applications which have the characteristics, defined in para. 5.2.1.1, will be rejected. The resulting magnitude is examined for the following characteristics.

ON: Canceled application
Condition: ExtendedDataElementName = “CreditDecision“ AND ExtendedDataElementValue = “red“
Action: Red credit decision

ON: Canceled application
ExtendedDataElementName = “Employment“
ExtendedDataElementValue = “false“
Action: Self-employed

The third case of missing user data can be determined by different rules. A similar rule is offered in para. 5.4.1.2. Here user activities U are measured in the form of JSP requests. If only one (the one of entering the credit application) occurs within time window T, no further inputs were made by the user. If the number of activities U does not exceed 1, no serious request was made.

ON: U WHITHIN T
Condition: Count (U.component="JavaServerPage“)=1
Action: No serious application

Based on these characteristics, canceled applications can be proved to be relevant for the TeamBank.

ON: Canceled application OR Red credit decision OR Self-employed OR No serious application
Condition:
Canceled application.threadId != Red credit
decision.threadId
AND Canceled application.threadId != Self-employed.threadId
AND Canceled application.threadId != No serious
application.threadId
Action: Canceled credit application which is relevant for
the TeamBank

5.4.1.5 Enrichment and aggregation

Canceled applications which are relevant for the TeamBank can be enriched with
the referred business data. Furthermore the cancelation point can be determined
and enriched.

- Last entered name of the applicant.
- Last entered email address of the applicant.
- Last entered credit amount.
- Last involved JSP.
- Last involved session bean.
- Last involved application server.

Thus the following exemplary event, shown in table 27, for a canceled credit
application results:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastName</td>
<td>Guschakowski</td>
</tr>
<tr>
<td>firstName</td>
<td>David</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:david.guschakowski@fh-regensburg.de">david.guschakowski@fh-regensburg.de</a></td>
</tr>
<tr>
<td>threadId</td>
<td>YZD932INC34DUI3335</td>
</tr>
<tr>
<td>creditAmount</td>
<td>10000 €</td>
</tr>
<tr>
<td>lastJSP</td>
<td>CreditDecision</td>
</tr>
<tr>
<td>lastSessionBean</td>
<td>DecisionMaker</td>
</tr>
<tr>
<td>lastApplicationServer</td>
<td>192.168.1.12</td>
</tr>
</tbody>
</table>

Table 27: Exemplary event for a canceled credit application

To get the metrics described in para. 5.2.1.1, the resulting relevant applications
can be aggregated based on one day. To get the drill-down functionalities,
described in para. 5.2.1.1, the relevant applications can be aggregated by applying further conditions.

5.4.2 BAM-view “Performance monitoring of a service”

To provide information for the BAM-view called “Performance monitoring of a service”, which are described in para. 5.2.1, a determination of patterns, which detect completed methods, “yellow methods” and “red methods”, has to be made.

5.4.2.1 Completed component method

The detection takes place by comparing the request and response of a component’s method. The method call C can be described with a parametrizable attribute component and the situation attribute.

\[
\text{component} = \$\text{component} \\
\text{situation} = \text{“StartSituation”}
\]

The method response R can be described with the according situation and the component attribute.

\[
\text{component} = \$\text{component} \\
\text{situation} = \text{“StopSituation”}
\]

Thus a completed method call can be determined with the following rule, whereby the time window should be near to the maximum duration of a component’s method call.

ON: C followed by R WITHIN T
Condition: C.threadId = R.threadId AND C.component = C.component AND C.method = R.method AND C.extId = R.extId
Action: Completed method call

The fact, that a component can be called multiple times within a process, implies a need for the additional identifier for a request / response pair, which was described in para. 5.3.1. Thus the rule is extended with the condition C.extId = R.extId:

ON: C followed by R WITHIN T
Condition: C.threadId = R.threadId AND C.component = R.component AND C.method = R.method AND C.extId = R.extId
Action: Completed method call
5.4.2.2 *Yellow component method*

Yellow component methods can be determined by negation of method response R. As mentioned in para. 5.2.2.1, the time window or limitation should be near the average duration of the component’s method call.

ON: C followed by !R WITHIN T
Condition: C.threadId = R.threadId AND C.component = R.component AND C.method = R.method AND C.extId = R.extId
Action: Yellow method call

5.4.2.3 *Red component method*

Red component methods can also be determined by negation of method response R. As mentioned in para. 5.2.2.1, the time window or limitation for a red method should be nearly the same as the time window for completed component methods. Thus, a method call can be determined either as correctly completed or as red respectively not completed.

ON: C followed by !R WITHIN T
Condition: C.threadId = R.threadId AND C.component = R.component AND C.method = R.method AND C.extId = R.extId
Action: Red method call

5.4.2.4 *Aggregation*

The determined methods can be aggregated and related to all method calls of a component in order to generate the metrics described in para. 5.2.2.1. Therefore method calls can be aggregated by filtering those events of a component which include the situation `StartSituation`. The observation period, on which the aggregations are based on, should be also parametrizable and should be chosen for each individual component, in order to get a representative perspective on the current situation of a component. Based on those aggregations, the component monitor may visualize the following metrics.

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>DecisionMaker</td>
</tr>
<tr>
<td>observationPeriod</td>
<td>00:01:30:0000</td>
</tr>
<tr>
<td>limitationYellow</td>
<td>00:00:02:4500</td>
</tr>
<tr>
<td>limitationRed</td>
<td>00:00:06:8000</td>
</tr>
<tr>
<td>methodCalls</td>
<td>1000</td>
</tr>
</tbody>
</table>
completedMethods | 997  
|------------------|---  
| yellowMethods    | 20  
| redMethods       | 0   
| RelationYellowMethods | 2 %  
| RelationRedMethods | 0 %  

**Table 28: Exemplary component report**

## 5.5 Conclusion

The described considerations give an overview, which measurements, cases and standards are involved in the realization of a BAM project. Because of time purposes and the focus on BAM implementations on different CEP platforms, chosen patterns and parts of those standards were implemented. The implementation takes place on the CEP platforms Coral8, Streambase, Tibco Business Events and Sytstar Business Bridge, whereby the practicability and the complexity of the implementation of the defined patterns will be evaluated. Figure 32 shall give an overview on the desired BAM architecture.

![Figure 32: Architecture for the BAM implementation](image)

Additionally, the considerations of chap. 4 and 5 represent the basis for the creation of a reference model „loss pattern“. In order to reach a formalization of the reference model, an orientation on the theory of design patterns [Gamma 1995] [Buschmann 1996] is applicable. For the definition of design patterns the
authors offer different structures [Silberbauer 2007]. Table 29 shows that the structures exhibit similarities as well as differences in content wise points and arrangement of the points.

<table>
<thead>
<tr>
<th>Structure elements of Gamma</th>
<th>Structure elements of Buschmann</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Intent</td>
<td>Also Known As</td>
</tr>
<tr>
<td>Also Known As</td>
<td>Example</td>
</tr>
<tr>
<td>Motivation</td>
<td>Context</td>
</tr>
<tr>
<td>Applicability</td>
<td>Problem</td>
</tr>
<tr>
<td>Structure</td>
<td>Solution</td>
</tr>
<tr>
<td>Participants</td>
<td>Structure</td>
</tr>
<tr>
<td>Collaborations</td>
<td>Dynamics</td>
</tr>
<tr>
<td>Consequences</td>
<td>Implementation</td>
</tr>
<tr>
<td>Implementation</td>
<td>Example Resolved</td>
</tr>
<tr>
<td>Sample Code</td>
<td>Variants</td>
</tr>
<tr>
<td>Known Uses</td>
<td>Known Uses</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>Consequences</td>
</tr>
<tr>
<td></td>
<td>See Also</td>
</tr>
</tbody>
</table>

Table 29: Design pattern structure elements of Gamma and Buschmann [Silberbauer 2007]

For the description of the reference model the description structure of Buschmann was selected. The structure of Gamma also offers a suitable structure for the description of reference models [Silberbauer 2007]. For a better understanding of the reference model the particular elements of the description structure of Buschmann are described in table 30.

<table>
<thead>
<tr>
<th>Structure element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name and a short summary of the pattern.</td>
</tr>
<tr>
<td>Also Known As</td>
<td>Other names for the pattern, if any are known.</td>
</tr>
<tr>
<td>Example</td>
<td>A real-world example demonstrating the existence of the problem and the need for the pattern. Throughout the description we refer to the example to illustrate solution and implementation aspects, where this is necessary or useful.</td>
</tr>
<tr>
<td>Context</td>
<td>The situations in which the pattern may apply.</td>
</tr>
<tr>
<td>Problem</td>
<td>The problem the pattern addresses, including a discussion of its associated forces.</td>
</tr>
<tr>
<td>Solution</td>
<td>The fundamental solution principle underlying the pattern.</td>
</tr>
<tr>
<td>Structure</td>
<td>A detailed specification of the structural aspects of the</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Typical scenarios describing the run-time behavior of the pattern.</td>
</tr>
<tr>
<td>Implementation</td>
<td>Guidelines for implementing the pattern.</td>
</tr>
<tr>
<td>Example Resolved</td>
<td>Discussion of any important aspects for resolving the example that are not yet covered in the solution, structure, dynamics and implementation sections.</td>
</tr>
<tr>
<td>Variants</td>
<td>A brief description of variants or specializations of a pattern.</td>
</tr>
<tr>
<td>Known Uses</td>
<td>Examples of the use of the pattern, taken from existing systems.</td>
</tr>
<tr>
<td>Consequences</td>
<td>The benefits the pattern provides and any potential liabilities.</td>
</tr>
<tr>
<td>See Also</td>
<td>References to patterns that solve similar problems and to patterns that help us refine the pattern we are describing.</td>
</tr>
</tbody>
</table>

Table 30: Elements of the Design Pattern description of Buschmann [Silberbauer 2007]

The definition of the reference model takes place via generalization of the specifications in chapter 5 to a more general and abstract level. Therefore project-specific metrics are generalized and general solutions and implementation techniques are pointed out and ordered into the description structure of Buschmann (see Appendix B [Silberbauer 2007]).
6 Implementation of the event generators by EJB interceptors and servlet filters

A prerequisite was the implementation of an event generating mechanism without altering the already existing code of the easyCredit Online application. The concept of the interceptor (see para 2.4) shows a way for reaching this goal of a non-intrusive event generation. By designing and embedding the interceptor class in the prevailing existing infrastructure of TeamBank, events can be generated at every single execution of each deployed component, like for example session beans (see para 2.1.3), based on the inputs and actions of a customer throughout the whole credit application process. Due to the fact that these interceptors are behind the interfaces of the components (see para 2.4.1) and are just intercepting the requests on a technical component level which is part of the business logic like beans, another event generating concept had to be set up for tracking the user’s movement throughout each requested JSP. To cover these frontend actions a servlet filter was applied. The servlet filter acts basically like the interceptor and therefore also represents a non-intrusive way to generate user based events which consist crucial information, like the actual active page each user is viewing at the moment. Furthermore the servlet filter adds the threadId element for each user to the CBE (see para. 5.3) which identifies a credit application uniquely for the particular customer. Figure 33 illustrates the three layer architecture used by the TeamBank Company today. The first of the three layers represents the user interaction layer. Through this portal or entry point, users can connect to the easyCredit application. This Web-based user interface handles all user interactions. The above described servlet filter traces these user interactions being made at this layer and automatically creates a new threadId (see para. 5.3) for every credit application session. The generated CBE events consist of the information about the actual page and are being sent to the JMS topic messaging component of the JBoss server by the servlet filter. The second layer represents a non-BPEL workflow like described in para. 3.2.1. This inflexible workflow is implemented as a session bean. The third layer then contains the connected
services which also implement part of the business logic and are orchestrated by the static non-BPEL workflow.

![Three-layer architecture of TeamBank’s easyCredit application](image)

**Figure 33:** The three-layer architecture of TeamBank’s easyCredit application [Ammon2006a]

### 6.1 Implementation environment and TeamBank’s server architecture

TeamBank’s easyCredit instant online credit-application is completely deployed on a JBoss application server version 4.0.5 (see para 2.2). All created and deployed components in sense of the use case realization were designed and implemented via JBoss IDE for Eclipse 2.0.0.Beta [JBoss 2007] which devoted resources to the Java Runtime Environment 1.5.0_11. Figure 34 illustrates the TeamBank server architecture. Each client request is using the http protocol.
A load balancer handles every request based on the round robin concept [Wikipedia 2007d]. The load balancer then passes the JSP request to the Web-server with the slightest workload. Each Web-server, in return, forwards the request to the fixed coupled application server which runs the actual easyCredit application component.

6.2 Deployment

The application of the interceptor class respectively the servlet filter class was accomplished by extending the already predefined deployment descriptor files of the JBoss application server online configuration. For the integration of the applied session bean-interceptor Ejb2Interceptor into the easyCredit application the ..\server\online\conf \standardjboss.xml file which contains the default EJB configurations – e.g. invokers, cache sizes or interceptors – is modified like shown in Figure 35.

```xml
<container-configuration>
  <container-name>Standard Stateless SessionBean</container-name>
  <call-logging>false</call-logging>
  <invoker-proxy-binding-name>stateless-rmi-invoker/invoker-proxy-binding-name</invoker-proxy-binding-name>
  <container-interceptors>
    <interceptor>de.easycredit.ops.ejb2.Ejb2Interceptor</interceptor>
  </container-interceptors>
</container-configuration>
```

Figure 35: Integration of the session bean interceptor into the JBoss configuration
For integrating the servlet filter class CepFilter, the web.xml file is extended in order to apply the filter to all deployed Web applications running on the easyCredit-application JBoss instance.

Figure 36: Applying the servlet filter to the Web application archive of easyCredit

The web.xml file is embedded in the WEB-INF directory of the ec-credit-frontend-internetgast-v1.0.war file which, in turn, is part of the Enterprise Application Archive (EAR) file 4.edt_online.ear. The last adaption of the JBoss configuration included the creation of a JMS topic in order to pass the generated events to this topic at the moment they are instantiated. Therefore the JMS topic is created in the jbossmq-destinations-service.xml file within the ../deploy/jms folder of the JBoss server. Figure 37 illustrates this step of adding the newly created JMS topic to the JNDI reference list of JBoss.

Figure 37: Creation of the "EventCloud" JMS topic

Besides the integration of the servlet filter, the interceptor and the registration of the JMS topic at the JMS service provider in order to enable volitional
functionality of these classes, the deployment was executed by a self designed ant deployment script.

6.2.1 The ant deployment script

By using the ant deployment script technology it is possible to automatically deploy every utilized component on the JBoss server, whereby it enables the compilation and packaging into war, jar or rar files. For adopting this concept, an xml file containing the ant script has to be created. As seen on Figure 38, the design of this deployment descriptor consists of properties which are variables. These can be used for the reutilization within the build file. The target elements include multiple kinds of jobs which execute the building, compiling, deleting and copying of the introduced components.

```
<project name="easyCredit Ant" default="deploy" basedir="." >
  <property name="source.dir" value="de/easycredit/eqp/ejb2"/>
  <property name="web.dir" value="de/easycredit/eqp/web"/>
  <property name="lib.dir" value="C:/easycredit/JBoss-4.0.5.BA"/>
  <property name="bin.dir" value="target"/>
  <property name="META-INF.dir" value="META-INF"/>
  <property name="deploy.dir" value="/easycredit/jboss-4.0.5.BA/server/online/deploy/0.0b-intercept.ear"/>
  <property name="jarName" value="ejbInterceptor.jar"/>
  <property name="descriptionsName" value="easyCreditInterceptor"/>
  <target name="compile">
    <mkdir dir="${bin.dir}"/>
    <javac srcdir="${source.dir}" destdir="${bin.dir}" debug="true" deprecation="true">
      <classpath>
        <fileset dir="${lib.dir}" includes="**/*.jar"/>
      </classpath>
    </javac>
    <javac srcdir="${web.dir}" destdir="${bin.dir}" debug="true" deprecation="true">
      <classpath>
        <fileset dir="${lib.dir}" includes="**/*.jar"/>
      </classpath>
    </javac>
    <javac srcdir="${source.dir}" destdir="${bin.dir}" debug="true" deprecation="true">
      <classpath>
        <fileset dir="${lib.dir}" includes="**/*.jar"/>
      </classpath>
    </javac>
  </target>
```

Figure 38: Design of the applied ant deployment descriptor

By this, it is possible to dynamically add, alter, delete and deploy components just by referring the component sources like displayed in Figure 38. This code snippet illustrates inter alia the definition of the properties “ejb2”, “Web” and “message” which represent the packages containing the interceptor class, the servlet filter and the messaging classes. The job definition with the target name “compile” first creates a directory “target” where the particular .java classes are compiled to. Secondly the compilation of the classes is done.
6.2.2 Interceptor class

The “Ejb2Interceptor” interceptor class was created and included within the package de.easyCredit.cep.ejb2. As described in para. 2.4 and documented in the Figure 39, the interceptor class features the `invoke()` method which is composed of two identical parts. Each part executes the methods `extractDataFromInvocation()` – to extract the relevant data from the Invocation object, `createEvent()` – to create a HashMap containing the specific information needed to fill the CBE body – and the `sendJMS()` method – to send the generated HashMap to another class which hence builds a CBE-conform XML string. These very methods are being executed before and after the actual invocation of the requested session bean. To simplify this, a separation is represented by a line comment within Figure 39.

```java
public Object invoke(Invocation invocation) throws Exception {
    HashMap extractedInvocationData = new HashMap();
    extractedInvocationData = extractDataFromInvocation(invocation);
    //send JMS
    myJMSPublisher.sendJMS(createEvent(extractedInvocationData, "Start Situation"));
    mySystemPublisher.sendJMS(createEvent(extractedInvocationData, "Stop Situation"));

    //after method was called, on the way back the return is intercepted and the result set in object
    Object result = getNext().invoke(invocation);
    //extract data out of intercepted method invocation
    extractedInvocationData = extractDataFromInvocation(invocation);
    //send JMS as Stop Situation of the called method
    myJMSPublisher.sendJMS(createEvent(extractedInvocationData, "Stop Situation"));
    mySystemPublisher.sendJMS(createEvent(extractedInvocationData, "Stop Situation"));
    //return the result of the method invocation
    return result;
}
```

Figure 39: Basic structure of the Ejb2Interceptor

Before and after each request the same methods are being executed and differ just in the committed situation string, which is “Start Situation” as the request is not yet passed towards the called component and “Stop Situation” as the response of the originally requested session bean component is being intercepted. First of all, the `java.util.HashMap extractedInvocationData` is declared and instantiated representing the key-value pairs holding the relevant information, extracted from the context of each session bean request. As read from Figure 40,
the method extractDataFromInvocation() gathers information like the desired credit amount, clear credit or totals from out the invocation. As well as data about the requested session bean itself, its deployment environment, the particular method which is being called, the methodCaller of the request and so on.

```
//getArguments() {String value = buildString(extractDataFromInvocation());}
myCE = CreditEntityServiceInput(transferData.getArguments());
//getArguments() {String value = buildString(extractDataFromInvocation());}
myCE = CreditEntityServiceInput(transferData.getArguments());
//getArguments() {String value = buildString(extractDataFromInvocation());}
myCE = CreditEntityServiceInput(transferData.getArguments());
```

![Figure 40: Cutout of extractDataFromInvocation() method of the Ejb2Interceptor class](image)

The next step then passes this HashMap and the situation-string towards the method createEvent(). As illustrated in Figure 41, another HashMap is being declared and instantiated. This HashMap event represents the event as it is used to fill in the elements of the CBE structure. Therefore every relevant key-value pair of the transferred extractedInvocationData HashMap can be passed to the HashMap event. Furthermore event is being enriched with additional elements needed to complete the data structure of the CBE.
private HashMap<String, String> createEventData() throws Exception {
    HashMap<String, String> event = new HashMap<String, String>();
    try {
        event.put("version", "1.0");
        event.put("creationTime", formatter.format(new Date(System.currentTimeMillis())));
        event.put("location", java.net.InetAddress.getLocalHost().toString());
        event.put("application", "InterneParallels");
        event.put("threadId", CepFilter.getThreadSessionId());
        event.put("component", extractedInvocationData.getComponent().toString());
        event.put("componentIdType", "StatelessSessionBean");
        event.put("method", extractedInvocationData.getMethod().toString());
        event.put("situation", situation);
        event.put("globalInstanceId", "_" + (String)event.get("threadId") + System.currentTimeMillis());
        event.put("nameType", "IP4");
        if (!extractedInvocationData.containsKey("gesamtBetrag"))
            return;
        event.put("gesamtBetrag", extractedInvocationData.get("gesamtBetrag").toString());
        if (!extractedInvocationData.containsKey("KreditWunsch"))
            return;
        event.put("KreditWunsch", extractedInvocationData.get("KreditWunsch").toString());
        if (!extractedInvocationData.containsKey("metroKredit") || "null"
            event.put("metroKredit", extractedInvocationData.get("metroKredit").toString());
    }
    return event;
}

Figure 41: The Ejb2Interceptor method createEvent() fills a HashMap representing the event.

Finally the HashMap event contains the following elements used to fill the body of the CBE as already mentioned in para. 5.1.2:

- **version**: used to identify the version of the CBE.
- **creationTime**: representing the internal system time, the interceptor is running on.
- **location**: representing the name and the current IP address of the actual system.
- **application**: displaying the session bean application (the static non-BPEL workflow as mentioned in para. 6.1 which implements the interceptor.
- **threadId**: this parameter is obtained from and created by the servlet filter class CepFilter, as described in para. 6.2.3.
- **component**: the current requested session bean component within the deployment.
- **componentType**: describes the intercepted component type. Due to the fact that a session bean-interceptor was utilized, the **componentType** is always “StatelessSessionBean”.
- **method**: the particular method of the requested component.
• **situation**: as mentioned above, this value can either be “StartSituation” or “StopSituation” depending on if a request towards a component got intercepted or a response.

• **globalInstanceId**: this element is the worldwide global identifier of each generated CBE and consists of the variable `threadId` plus the variable `creationTime` in milliseconds.

• **locationTyp**: represents the protocol being used to call the component.

• **extId**: additional identifier for recognition of method responses.

• **gesamtBetrag/nettoKredit/kreditWunsch**: depending on the existence of these key-values pairs within the passed HashMap `extractedInvocationData` the values are added to the HashMap `event` whereupon “gesamtBetrag” denotes totals, “kreditWunsch” means desired credit amount and “nettoKredit” stands for clear credit. The usage of German appellations for those variables is based on the fact that the whole easyCredit application as a German deployment features German variable names and method names which the authors wanted to retain.

Finally the logger class methods `createlogBookEntr()`, `writeAll2Log()`, `writeEvent2Log()` and the `sendJMS()` of the JMS publisher classes are invoked, passing the generated HashMap `event`.

### 6.2.3 Servlet filter class

The servlet filter class `CepFilter` within the package `de.easycredit.cep.web` is responsible for the generation of a session Id per threat and the monitoring and tracing of the user movements within the easyCredit frontend by reading out each `HttpServletRequest` [Sun 2001a]. Therefore the servlet filter enables the possibility of accessing the values of a request respectively response of an individual user call of a Web-resource; similar to the interceptor chains depicted in para. 2.4, filters can, depending on the configuration, establish a “filter-chain” which is being passed through the request. Mapping rules can define which filter handles which request. A possible filter chain can be seen on Figure 42.
However, for this use case neither filter chains were established nor mapping rules were defined. The class CepFilter has a similar setup like the class Ejb2Interceptor. Beside the needed local variables an instance of the ThreadLocal field is instantiated as thisThreadSessionId. In the class CepFilter, ThreadLocal instances are private static fields that wish to associate state with a thread (e.g., a user ID or Transaction ID). This thisThreadSessionId could be obtained by other classes implementing the CepFilter class by calling the method getThisThreadSessionId().

To initialize the filter the init() method had to be executed by committing an object of the FilterConfig class [Sun 2001b] in order to pass the information during the initialization. Like demonstrated in Figure 43, the CepFilter contains a method which is being executed the moment a servlet request is made by a user. This method functions almost identical like the invoke() method of the interceptor described in para. 2.4 and para. 6.2.2.

As illustrated, the doFilter() method is handed over objects of the classes ServletRequest [Sun 2002d], ServletResponse [Sun 2002c] and FilterChain [Sun 2002c]. After declaring and instantiating the object HttpSession [Sun 2000], created by the getSession() method of the HttpServletRequest object the session Id of every active httpSession is being stored in the String variable sessionId. The set() method of the static class variable

Figure 42: Possible servlet filter chain with applied mapping rules see [Rittmeyer 2007]
thisThreadSessionId is called with committing this sessionId variable and put into the HashMap class variable called event.

```java
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, ServletException {
    try {
        HttpServletRequest req = (HttpServletRequest) req;
        HttpSession httpSession = req.getSession();
        String sessionId = httpSession.getId();
        thisThreadSessionId.setAttribute(sessionId);
        event.put("thisThreadSessionId", thisThreadSessionId.get());

        this.fillHashMap(req, "Start Situation");
        myJSMPub.sendJMS(event);
        mySysLogJSMPub.sendJMS(event);

        // ServletRequest
        myLogger.writeHttpSession2Log(event.toJSONString());
        chain.doFilter(req, resp);
    } finally {
        // ServletResponse
        this.fillHashMap(req, "Stop Situation");
        myJSMPub.sendJMS(event);
        mySysLogJSMPub.sendJMS(event);
        myLogger.writeHttpSession2Log(event.toJSONString());
        // adding logbook entries
        log.debug("CepFilter called");
        log.debug("HttpSession filled=" + getThisThreadSessionId());
    }
}
```

Figure 43: The doFilter() method of the servlet filter

After that, identically to the method body of the invoke() method of the interceptor class, a method is executed which is collecting and storing the necessary data as key-value pairs within the HashMap event in order to fill the CBE structure, followed by calling the sendJMS() method of the JMS publisher class and the writeHttpSession2Log() with the generated HashMap object event as parameter. As displayed in Figure 43, these methods are being executed twice: before the request to the particular servlet is passed to another filter in the filter chain, respectively the wanted servlet and before passing the response of the called servlet. Like explained in 6.2.2, the only difference lies within the string parameter passed to the fillHashMap() method. If fillHashMap() is called while a request is intercepted by the CepFilter class, the passed string is “StartSituation”. When the response of the requested servlet is intercepted, it is “StopSituation”. This concept is also completely identical to the concept of the interceptor class Ejb2Interceptor. Illustrated in Figure 44, the HashMap
object `event` is enriched with the necessary data, the CBE structure respectively the evaluated correlation engines require. Next to the already mentioned variables like version, application, location and so on, which are identical to the values set within the `createEvent()` method of the interceptor class, new key-value pairs or different values of the same key-string are added in the HashMap entry. These variables include:

- `getRequestedSessionId`: the session Id of the particular session
- `componentType`: this too, differs from the value of the `componentType`, set in the interceptor class. Due to the fact, that the `componentType`-variable mentions the type of the requested component which is being intercepted, the value is set to “JavaServerPage” because as described above the CepFilter class just intercepts requests on JSPs.
- `component`: Like the component variable added to the HashMap object `event` in `createEvent()` method of the Ejb2Interceptor, describes the requested component. This is, in case of this CepFilter, a deployed JSP. The variable is read out of the hidden field tag parameter “fromPage” every user-requested JSP contains. By adding this key-value pair to the HashMap object it is possible to define exactly what page the user is viewing at the moment and is therefore his movement forwards or backwards through the easyCredit online application is traceable.
public void fillHashMap(HttpServletRequest req, String situation) throws Exception
{
try
{
    event.put("getRemoteUserNumber ": req.getParameter("username"));
    event.put("getRequestedSessionId ": req.getSessionId());
    event.put("componentIdType ": "java:ServletPage");
    event.put("getServletPath ": req.getServletPath());
    event.put("getServletURL ": req.getServletURL());
    event.put("page ": req.getParameter("currentPage"));
    event.put("threadId ": req.getSession().getId());
    event.put("application ": "Internetgaststrecke");
    event.put("version ": "1.11");
    event.put("creationTime ": new Date(System.currentTimeMillis()));
    event.put("location ": java.net.InetAddress.getLocalHost().toString());
    event.put("locationType ": "IP4");
    event.put("globalInstanceID ": "event threadId".toString() + System.currentTimeMillis());
    if(event.get("page") != null)
    {
        event.put("component", "initPage");
    }
    else
    {
        event.put("component", event.get("page"));
    }
    if(situation == "Start Situation")
    {
        event.put("situation", situation);
    }
    else if(situation == "Stop Situation")
    {
        event.put("situation", situation);
    }
}

Figure 44: The fillHashMap() method fills HashMap object "event" with the necessary information, used to complete the body of the CBE

6.2.4 JMS publisher class

Both, the Ejb2Interceptor class and the CepFilter class instantiate objects of the class JMSPublisher of the package de.easycredit.message.

The main purpose of this class is the generation of the CBE structure via string operations as seen on Figure 45. The authors decided to build this CBE body not by using actual XML parsing techniques or XML string creator classes by reason of performance. It is also illustrated in Figure 45 that the elements of the basic structure of the CBE – required to parse the generated XML – are filled dynamically by adopting the particular data of the passed HashMap objects of the interceptor class respectively servlet filter class.
Implementation of the event generators by EJB interceptors and servlet filters

6.2.4.1 JMS Topic

The second part of the JMSPublisher class is defined by the method sendJMS(). As shown on Figure 46 this method generates every required standard objects in order to create a new JMS topic connection referencing the topic integrated in the jbossmq-destinations-service.xml file. By looking up the JNDI context directory a new topic connection to the topic “topic/EventCloud” is established.
Implementation of the event generators by EJB interceptors and servlet filters

Figure 46: the sendJMS() method passes the generated CBE-conform XML string to a JBoss topic

After establishing a connection to the respective topic, the generated CBE-conform XML-string is passed as MapMessage before the connection is destroyed afterwards.

6.2.5 Logfile class

The logfile class InterceptorLogger of the package de.easycredit.cep.message is deployed for debugging purposes only. As seen on Figure 47, the main method of this class write2Log() gets passed a String object message and a String object defining the destination where to write the message into. Surrounded by a try-catch block a new FileWriter instance is declared, initialized and used to create the BufferedWriter object bufferedWriter. This bufferedWriter adds the message which is passed to the method into the log file which exists at the specified destination.

Figure 47: Logfile class’s method write2Log()
Following files are created at the directory C:/easycredit/ejb2interceptors/logfiles/ due to the relative method invocations:

- **httpLogger.log:** the alteration of this file is handled by the method `writeHttpSession2Log()` which is implemented by the CepFilter class and just contains entries describing any possible information the CepFilter class can extract out of every intercepted JSP request and response.

- **interceptorLogger.log:** this is similar to the httpLogger.log file. The particular method `writeAll2Log()` is implemented by the class `Ejb2Interceptor` and contains any possible information the interceptor retrieves out of every intercepted session bean request or response.

- **JMSlogger.log:** this file contains all CBE-conform XML strings which are passed to the JMS topic “topic/EventCloud”.

### 6.3 Conclusion

Based on the independent component concept, featured by JBoss as mentioned in para. 2.2ff, the binding of the session bean interceptor `Ejb2Interceptor` respectively the servlet filter `CepFilter` into the already existent JBoss deployment of the easyCredit application constitutes a non-intrusive way to generate the CBE events. No class already developed and deployed by Senacor company had to be modified in any way. Therefore the utilization of the interceptor concept – for intercepting requests on the technical level – depict a proper way to generate events for every component defining the business logic whereas the servlet filter concept shows an ideal way to trace every movement of each single user during his activity within the frontend of the easyCredit-Internet application.

#### 6.3.1 JMS as standard for event messaging?

Because approximately 20.000 credit applications have to be handled per day with each application requesting about 50-100 components and services, depending on
the user’s inputs, between 20 and 30 requests are applied per second [Ammon 2006a]. Due to the above mentioned concept of generating an event for each request and the same amount per response of the particular requested components, this leads to a total event generation of circa 40-60 events per second. Based on the high performance JMS-messaging benchmark test made by the Crimson Consulting Group [Crimson 2003] a volume of 40-60 transmitted MapMessages might define no drawbacks in applying the JMS Messaging concept (see para. 2.3) for sending the CBE-conform XML-strings to the JMS topic.
7 Implementation of the enterprise cockpit and the BAM-views

The general precept of any product is that simple things should be easy, and hard things should be possible.

Alan Kay

Following paragraphs display the implementation of the tested CEP platforms. As test environment for all installation and deployment of all components, a single machine featuring a Pentium 2.4 Ghz processor and 2048 MB RAM was employed. A test version of the easyCredit application was deployed on this machine. The operating system was WindowsXP ServicePack 2.

7.1 Coral8

The CEP platform Coral8 allows an enterprise to build CEP applications within the Coral8 Studio and execute them on the Coral8 Server. Applications are built with the Continuous Computing Language (CCL), and compiled and deployed on the Coral8 Server via CCL Compiler. The Coral8 Studio allows the development of CCL statements within parametrizable modules in order to reduce complexity of CEP applications. The Coral8 Studio also provides central management capabilities for a distributed network or GRID of Coral8 servers. The use case was built in version 4.6.1, the current version is 4.8.1 [Coral8 2006a]. CCL implements standard SQL syntax which is added by CEP related expressions. In order to include historical data, Coral8 fully integrates databases into the platform. Therefore database queries can be written within CCL, which lets a resulting database table appear as the same as streaming data to a developer. Coral8 provides several adapters for an integration of the engine into a given enterprise environment. These adapters cover several message delivery types like JMS, Websphere MQ, JDBC, ODBC, TIBCO Rendezvous, SOAP, RSS/ATOM, E-mail etc. The Coral8 SDK, which is available in C, C++, Java, .NET, Perl, and Python, allows developers to build custom adapters, user-defined functions and aggregators. The enterprise edition of the Coral8 Server also supports SNMP,
which enables enterprise management frameworks and consoles such as HP OpenView, IBM Tivoli and CA Unicenter to monitor the Coral8 Server behaviors. The Coral8 platform is supported for the most popular operating systems, such as Windows 2003 Server, Windows 2000 Server and Windows XP, RedHat Enterprise, SuSE, Fedora and FreeBSD [Coral8 2006a], [Coral8 2006b], [Coral8 2006c]. Figure 48 gives an overview about the described components and their interaction.

![Coral8 platform](attachment:image)

Figure 48: Coral8 platform [Coral8 2006a]

7.1.1 **Input adapters**

As introduced in para. 5.3, the event cloud consists of JMS events with the CBE format. The reduction of event types results in the usage of only one input adaptor to integrate Coral8 into the enterprise environment. The JMS adaptor is an Out-Of-Process adaptor, which means, that the adaptor runs separately to the Coral8 server. The advantage of Out-Of-Process adapters is that a crash within the adaptor doesn’t affect the Coral8 server, whereby a crashing In-Process adaptor would result in a crash of the entire Coral8 server. The disadvantage is a slower
event delivery because of extra delivery overhead. The JMS adaptor is a runnable Java class which establishes a JMS session and subscribes to the given topic, then listens on that session and sends incoming messages as tuples into the stream. The incoming messages must have the MapMessage type, whereby the MapMessage keys must correspond exactly to the stream's tuple descriptor [Coral8 2006d].

A multitude of MapMessage formats would result in a similarly high amount of adaptor instances and corresponding input streams. The CBE format reduces the amount of adaptor instances to one, because the entering MapMessages have one common message format. The Java class has to be started manually by applying several parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic</td>
<td>The name of the topic to subscribe to</td>
<td>Required</td>
</tr>
<tr>
<td>Factory</td>
<td>Connection factory name</td>
<td>Required</td>
</tr>
<tr>
<td>factoryClass</td>
<td>Factory class</td>
<td>Optional</td>
</tr>
<tr>
<td>url</td>
<td>The CCL URL of the stream to write to</td>
<td>Required</td>
</tr>
<tr>
<td>Host</td>
<td>The host on which the JMS Provider is running</td>
<td>Optional</td>
</tr>
<tr>
<td>Port</td>
<td>The port number of the JMS Provider</td>
<td>Optional</td>
</tr>
<tr>
<td>c8.baseHostPort</td>
<td>Host and port of the Coral8 server</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 31: JMS adaptor parameters [Coral8 2006d]

7.1.2 Architecture and data flow

The CEP application is basically divided in three sub modules. The incoming events of the JMS adaptor arrive at the input stream. The XML events in the CBE format will be connected through the input stream to the Conversion module, which maps the XML events into the primitive data types of Coral8. The output of the Conversion module is the event stream ProcessEvent and the event stream ExtendedDataElements. The ProcessEvent stream consists of events with those CBE properties which describe the involved component and situation. The ExtendedDataElements stream contains the events which describe the business data. The ComponentMonitors module consists of multiple reusable ComponentMonitor sub modules. Each instance of this ComponentMonitor measures the performance of one stateless session bean. Therefore the generated events in the ProcessEvent stream will be filtered for events with the value
StatelessSessionBean as componentType attribute. The filtered events in the SessionBeans stream are the base for the ComponentMonitors module. The outputs of the ComponentMonitors module are the metrics of the BAM-view “performance monitoring of a service”, the corresponding drill-down features and alerts. The third module is the AmountMonitor module, which is responsible for the determination of the metrics for the BAM-view “loss by canceled credit applications”. The ProcessEvent stream is used to detect canceled applications and the ExtendedDataElements stream is used to enrich a canceled application with the related business data. The outputs of the AmountMonitor module are the metrics of the BAM-view “loss by canceled credit applications”, the corresponding drill-down features and alerts. The Coral8 Studio generates flow models of the application and presents it like Figure 49 to give an overview, how the incoming events run through the application to generate the resulting outgoing events. To give a better understanding, the names of the streams, modules and queries are inserted in the generated figure.

Figure 49: Coral8 data flow

7.1.3 Integration of Common Base Events

Coral8 supports XML as a native data type. The Coral8 implementation of XML is based on the SQL standard SQL:2006. Coral8 uses XPath [W3C 1999] to get access on XML elements and attributes. By usage of the Coral8 expression
XML\textsc{ExtractValue} and the appropriate path to a CBE property with XPath, all properties of the CBE are extracted and converted into the Coral8’s primitive data types.

Thus XPath allows the identification of elements in a hierarchy by applying the path to the element, which could look like \texttt{./subelement/subsubelement}.

The point indicates the beginning from the current root element, whereby child elements are determined by the slash and the name. Attributes of elements can be identified by adding the “@” symbol in front of the attributes name, so the attribute of a root element is identified by \texttt{./@attribute [Coral8 2006e]}.

For example the extraction of the \texttt{creationTime} property from the events containing CBE looks like:

\begin{verbatim}
XML\textsc{ExtractValue}(InputStream.CBE,'./CommonBaseEvent/@creationTime')
\end{verbatim}

The fact, that a stream, similar to a database table, has a predefined number of columns, doesn’t allow a schema definition with no predefined column number. Within a XML hierarchy, sets can be defined without a specified number of elements like for example:

\begin{verbatim}
<element><subelement></subelement><subelement></subelement></subelement></element>
\end{verbatim}

The CBE also has such a set of elements in terms of \texttt{ExtendedDataElements}, which may occur multiple times within one CBE. To provide the extracted data from the CBE in streams, a split of the CBEs into two streams has to be made, the first stream is called \texttt{ProcessEvent} with the elements occurring one time within the CBE and a second one called \texttt{ExtendedDataElements} with the \texttt{ExtendedDataElements}. The relationship between these two streams takes place with the event identifier \texttt{globalInstanceId}. The identifier \texttt{threadId} for a credit application instance is also in \texttt{ExtendedDataElements} to get an easier correlation to a credit application. Thus the relation from a \texttt{ProcessEvent} to \texttt{ExtendedDataElements} is 1 to \texttt{m}. 

7.1.4 **BAM-view: “Loss by canceled credit applications”**

As mentioned in para. 5.4.1, the determination of the metrics for the BAM-view takes place at the base of canceled applications. The detection of a canceled application will be done by the chosen rule “time exceeding of the beginning/end of a credit application” and in a parametrizable way in order to guarantee a fast reaction on process changes of the easyCredit application. The following parameters allow these reactions.

The definition takes place with the start component parameter $entry and the start method $enMethod. On further occurrences of start point compliant events, the stream will be checked whether this event starts a new credit application or it occurs within in an existing one. The definition of the end point of the credit...
application takes place with the end component parameter $exit$ and the end method $exitMethod$. By applying the scenario “start point not followed by end point”, a canceled application can be detected. The scenario is based on the parametrizable time window $appTime$. In Coral8 a sequence of events like “followed by” can be described with a semicolon. An exclamation point defines a non occurrence of an event; this is also called a negative event [Coral8 2007a], [Coral8 2007b]. Figure 53 shows, how to implement the pattern in CCL:

```
INSERT INTO CanceledApplicationPattern
SELECT EntryComponentMethod.threadId as threadId,
     EntryComponentMethod.location as location
FROM
  (SELECT InputStream.threadId as threadId,
      InputStream.location as location
   FROM InputStream,
   (SELECT *
    FROM InputStream
    WHERE component = $entry AND method = $exitMethod AND Situation = "StartSituation")
   AS FirstInput)
WHERE $appTime
     AND FirstInput.creationTime < FirstInput.threadId
     AND FirstInput.globalInstancesId = FirstInput.globalInstancesId
     AND FirstInput.threadId
     AND FirstInput.threadId = ExitComponentMethod.threadId;

Figure 53: Code example for the canceled application pattern
```

The detected canceled applications will be enriched with the exit of the application and with the relating business data. Therefore the ProcessEvent is filtered for the JSP and session bean which was last involved within the credit application. The related business data, in this case credit amount, is also filtered of the stream ExtendedDataElements. These enrichments take place by correlation of the streams by comparing their threadId. The resulting enriched canceled application includes the following columns:

```
<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ThreadId</td>
<td>string</td>
</tr>
<tr>
<td>2</td>
<td>error</td>
<td>string</td>
</tr>
<tr>
<td>3</td>
<td>amount</td>
<td>float</td>
</tr>
<tr>
<td>4</td>
<td>method</td>
<td>string</td>
</tr>
<tr>
<td>5</td>
<td>action</td>
<td>string</td>
</tr>
<tr>
<td>6</td>
<td>status</td>
<td>string</td>
</tr>
</tbody>
</table>

Figure 54: Schema of canceled application
```

Figure 55 shows the detection and enrichment of canceled applications.
The detected canceled applications will be aggregated in order to determine the metrics for the BAM-view. Therefore a count of the last involved application server and session bean will be made. A determination of the results’ maxima will be done in the next step. Those maxima will be merged with the aggregated losses. The aggregation takes place by determination of the summarized, average and maximum loss on the base of the last 24 hours. The resulting information build the metrics for the BAM-view “loss by canceled applications”.

The count of JSPs will be prepared for a drill-down view, therefore the number of cancelations and the value of the canceled credit related to a JSP will be calculated. Additionally the canceled applications will be prepared to be visualized in the drill-down view.

The query Filter: Alerts determines those occurrences which are interesting for decision makers. These conditions of interest are the exceeding of thresholds defined in the parameters $yellowThreshold and $redThreshold (see para. 5.2.1.2). These metrics will be sent out by an email adaptor. The view shows the steps taken towards the BAM metrics, drill-down metrics and alerts.
Implementation of the enterprise cockpit and the BAM-views

Figure 56: Aggregation of canceled applications

The resulting metrics in Figure 57 will be updated on every new occurrence of a relevant event. One part of the drill-down view gives a more detailed perspective, on which JSPs the cancelations were made.

Figure 57: The determined BAM metrics

Figure 58: Drill-down: Canceled applications per page

7.1.5 BAM-view: “Performance monitoring of a service”

The module ComponentMonitors contains multiple instances of the module ComponentMonitor. Because the monitored components are session beans, a ComponentMonitor instance just receives events describing session beans. In order to create a reusable monitoring module, it needs a parameter $component to identify component related events. The parameters $avgTime and $maxTime
are used as the time windows T for the patterns (see para. 5.4.2), which identify a correct method completion. The aggregation of the pattern results is based on the $observationPeriod parameter. The threshold parameters $redThreshold and $yellowThreshold are used as conditions for the alert generation.

As mentioned in para. 5.4.2, the behaviour of method calls is the base for the determination of the BAM-view metrics. Therefore completed method calls, yellow method calls and red method calls are measured. The measurement of a completed method will be done by a pattern which describes the scenario “method call followed by method response within time window $maxTime”. A red method is described with the pattern “method call not followed by method response within time window $maxTime”. The pattern for a yellow method describes the scenario “method call not followed by method response within time window $avgTime”. Figure 60 illustrates how to implement the pattern for yellow methods. Therefore the InputStream is filtered for component relevant events with the situation description StartSituation or StopSituation. If two events don’t meet with the condition of same threadId, component, method and extID, the method is rated with state yellow [Coral8 2007a], [Coral8 2007b].
Figure 60: CCL implementation of the yellow method pattern

The pattern’s results will be aggregated and merged in several steps in order to ascertain the metrics of the particular component. The following metrics describe the different method behaviors in relation to all method calls. Therefore the parameters $observationPeriod$ as ObservationPeriod, $avgTime$ as LimitationYellow and $maxTime$ as LimitationRed are outputted. The resulting values, based on the parameters are the minimum duration of a method call as MinDurationMethod, the average duration of a method call as AvgDurationMethod, the maximum duration of a method call as MaxDurationMethod, the number of incoming, completed, yellow and red method calls. The metrics Inc_Yellow_Rate and Inc_Yellow_Rate describe the percentage of yellow respectively red methods to the incoming method calls.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>String</td>
</tr>
<tr>
<td>ObservationPeriod</td>
<td>String</td>
</tr>
<tr>
<td>MethodsIncoming</td>
<td>Long</td>
</tr>
<tr>
<td>MethodCompleted</td>
<td>Long</td>
</tr>
<tr>
<td>ExecutionMethod</td>
<td>String</td>
</tr>
<tr>
<td>AvgDurationMethod</td>
<td>String</td>
</tr>
<tr>
<td>IncYellowMethod</td>
<td>String</td>
</tr>
<tr>
<td>IncRedMethod</td>
<td>String</td>
</tr>
<tr>
<td>LimitationYellow</td>
<td>String</td>
</tr>
<tr>
<td>YellowRate</td>
<td>Long</td>
</tr>
<tr>
<td>RedRate</td>
<td>Long</td>
</tr>
<tr>
<td>Inc_Yellow_Rate</td>
<td>Float</td>
</tr>
<tr>
<td>Inc_Yellow_Rate</td>
<td>Float</td>
</tr>
</tbody>
</table>

Figure 61: Metrics for the BAM-view
As shown in Figure 62, the alert is done by filtering for an exceeding of the thresholds $\text{redThreshold}$ and $\text{yellowThreshold}$ for the metrics $\text{Inc\_Red\_Rate}$ and $\text{Inc\_Yellow\_Rate}$.

Figure 62: Component monitor

7.1.6 Problems and conclusion

The extraction of XML elements might cause problems, if the elements are within a namespace. Usually the XPath access on elements works like „element/subelement“, this is not possible, if they are within a namespace. Instead elements must be addressed with *[[local-name()]='element']/*[local-name()]='subelement']. These steps were not documented, but communicated and solved by the support of Coral8. In addition, the time stamp of an event cannot be pulled out of the event itself if Out-Of-Process adapters like the JMS adaptor are used. With In-Process adapters it’s possible to take a specified column of the input stream as basis for the time stamp of the incoming events. At present Coral8 works to place this feature for the JMS adaptor in one of the next releases. This feature is crucial for the use of patterns, because its time windows always refer to the time stamp of the events involved, thus for example a network delay can lead to unexpected results, because the involved events are not evaluated according to its generation, but due to its arrival in the CEP platform. In order to map complex relations within business processes, the necessary event must have a high quality such as CBE and the complexity of the application rises fast. To store relations persistent, the relations must be stored in data bases and have to be correlated with the event stream. Other platforms like Tibco or Systar provide
event driven object models, wherewith processes can be mapped and cascading effects can be realized (see para. 7.3, 7.4). In Coral8 these relations must be implemented in CCL by the usage of stored relations in data bases. Furthermore user-defined queries on the event driven data model from the outside are available in other platforms like Tibco BE or Systar BusinessBridge NG, whereby queries on the data model can be made during operation according to user-defined criteria (see para. 7.3, 7.4). Because Coral8 works after the push principle, it can generate events on the arrival of other events. Thus a new CCL statement must be written and be attainable by a custom adaptor from the outside for each resulting stream. So for example for each restriction criteria of a drill-down view, a separate filter statement has to be written in CCL. Coral8 offers an API to build custom expressions or functions, which may solve the problem; however these functionalities were not observed within the thesis. Besides these problems Coral8 delivers all capabilities to develop CEP based BAM tools. Therefore it provides a rich event processing language, adaptors and APIs to integrate into enterprise environments and monitor them. Problems may occur on bigger projects such as monitoring projects for hundreds to thousands of different processes, which need a high flexibility and adaptability. The vision of BAM is to let the operational department build their own custom alerts, conditions of interest and BAM views without programming queries (see para. 3.3). In version 4.6 of Coral8 these functionalities have to be coded instead of using flexible interfaces. Version 4.8 includes the feature remote procedure call (RPC) integration in order to start queries remotely. This may allow a more flexible access to Coral8 [Coral8 2007d]. Process changes have either be corrected or prevented with a very flexible design. Therefore a very high modularisation is needed and may lead to less transparency and more complexity. The implementation of this use case already resulted in approximately 300 lines of code. In 2007 Coral8 released a BPEL based CCL code generator in order to generate BAM-views with one click. This can be an appropriate solution to the above described problems, but a more exact observation about the abilities was not made within this thesis [Coral8 2007c].
7.2 StreamBase

StreamBase is a computing platform which was designed to meet the performance requirements of high-volume, real-time streaming applications. Therefore it implements a Stream Processing Engine that uses the Inbound Processing Architecture. Compared to the database model where data is first stored and indexed and then subsequently processed by queries, StreamBase processes the inbound data while it is "in flight," as it streams through the server. The inbound processing done by StreamBase applies the business logic that the developer defined [StreamBase 2007a]. StreamBase offers a graphical and Eclipse based development environment, called StreamBase Studio which provides a graphical and text-based approach to build streaming applications. It lets developers design, test, and deploy streaming applications. The implementation of the use case was done in version 3.5, the current version is 3.7 [StreamBase 2007a]. Using the graphical approach lets developers build application diagrams by dragging and dropping visual components, streams of data and connecting them via arrows to implement the desired business logic. These arrows represent the flow of data throughout the application. A component or operator has at least one input port and one output port by which it is affiliated with each other. The operator has configuration capabilities in dependency of the operator’s type, so for example a filter component can be configured by applying several filter conditions. Syntax checking or coding help is provided automatically by the Studio. The resulting application diagrams are stored in xml files with a .sbapp extension. StreamSQL is the text-based approach to describe StreamBase applications, whereby applications are coded using a query language to describe the application and query streams of data. These StreamSQL applications are stored in text files with a .ssql extension. Many of the key words used in this approach are common to the SQL standard SQL:2003 like for example SELECT, WHERE and FROM with many extending CEP specific key words like CREATE STREAM or CREATE WINDOW. This way a collection of SQL-like statements can be defined as an application. StreamBase also allows a combination of the approaches by creating a module with either approach that can be used in applications designed
with the other approach [StreamBase 2007a], [StreamBase 2007b], [StreamBase 2007e]. In the end, both offered approaches of StreamBase provide the same capabilities of developing CEP applications. It depends on the developer on which way he wants to design an application. Figure 63 shows the StreamBase Studio:

The StreamBase server loads the application and computes the inbound data according to defined operators, and responding to any other requests, such as requests for data from Java, C++, or .NET clients. Once the server is started, StreamBase Studio provides a second perspective, "Test/Debug," that contains tools to enqueue or dequeue data on the running application, or run a debugger to pause the application and step through its processing units, or view performance statistics. StreamBase also provides APIs (Java, C++, and on Windows, .NET) to develop client applications that enqueue to or dequeue data from StreamBase applications [StreamBase 2007a] [StreamBase 2007c].
7.2.1 Input adaptors

The embedded JMS adaptor subscribes for messages from a JMS message bus. An embedded adaptor is an adaptor that runs in the same process as a StreamBase server. When it receives a message, the JMS Reader creates a tuple from the message, and then sends that tuple to the connected operator whereby it serializes the messages it receives. The class DefaultFromJMSMapMessageConverter converts JMS MapMessages to StreamBase tuples, DefaultFromJMSMapMessageConverter, whereby it pairs the fields held by a MapMessage with fields in a StreamBase schema that have the same names and compatible types. If a name doesn't occur within the MapMessage [Sun 2004] or the StreamBase Schema, then that field is dropped. The JMS Reader also provides facilities for handling other mappings, specified in the configuration file. For example, a name map can be used to map a field on a MapMessage to a field on a StreamBase Schema that has a different name. The JMS Reader consumes messages from JMS destinations. The manner in which these messages are acknowledged is configurable within the JMS Reader. The setting for the acknowledge mode can impact the performance of the reader. The JMS Reader supports acknowledge modes of AUTO_ACKNOWLEDGE and DUPS_OK_ACKNOWLEDGE. The JMS Reader makes no effort to prevent or in any way handle the delivery of duplicate JMS messages. The adaptor can be configured with a .sbconfig file which allows the configuration of the data mapping from the JMS’s MapMessage to the StreamBase’s tuples. The JMS provider is also described within the configuration file, so the access to the provider, the connection factory, the context factory, the acknowledge mode and the desired topics can be defined [StreamBase 2007c].

7.2.2 Architecture and data flow

Similar to Coral8, the CEP application is divided in three sub modules. The incoming events of the JMS adaptor arrive at the Conversion module, which shreds the XML events into the internal data types of StreamBase. The
Conversion module outputs the event stream ProcessEvent and the event stream ExtendedDataElements. The ComponentMonitor sub modules of the ComponentMonitors module each measure the performance of one stateless session bean. The AmountMonitor module measures the daily loss by canceled applications. Figure 64 shows the application diagram generating the desired BAM metrics.

![Application diagram]

**Figure 64: Application diagram**

### 7.2.3 Integration of Common Base Events

StreamBase doesn’t support XML as a native data type so far. In order to integrate XML, the events have to be handled as type `string` and StreamBase offers the Java file `XMLNormalizer.class`, which acts as an XML extractor. To use the `XMLNormalizer.class`, the jar file `XMLNormalizer.jar` including the `XMLNormalizer.class` has to be imported in the project. The jar file also includes `XMLNormalizerOperator`, which extends the `Operator` class, a provided class by the StreamBase API. This allows the integration of the `XMLNormalizerOperator` as a visual operator, whereby it uses the functionality of the `XMLNormalizer`. As shown in Figure 65, the `XMLNormalizer` functionality can be used as a graphical operator after importing the jar file.
The operator takes a XML well-formed string as input and extracts the included elements to columns of the type string. Basically the configuration of the operator takes place by identifying the CBE elements with the XPath notation [W3C 1999] and mapping them to the corresponding output columns [StreamBase 2007c]. The problem, that a XML string was not parseable, if it has an XML declaration 

```xml
<?xml version="1.0" encoding="utf-8"?>
```

, can be prevented by cutting off the declaration with the string function `substr()`, whereby the first 37 letters will be cut off. As already mentioned in para. 7.1.3, the CBE has to be split into two streams, in order to realize an undefined amount of incoming `extendedDataElements`. The two resulting streams are `ProcessEvent` and `ExtendedDataElements` and the `ProcessEvent` stream will be enriched with the `ExtendedDataElements`, which contain the `method` and the `extId`. As already mentioned the relationship between these two streams takes place with the event identifier `globalInstanceId`. The identifier `threadId` for a credit application instance is also in `ExtendedDataElements` to get an easier correlation to a credit application. Thus the relation from a `ProcessEvent` to `ExtendedDataElements` is 1 to m. Figure 66 shows the configuration parameters for the extraction of the CBE elements into `ProcessEvent`. The parameter `XML Element being parsed:` specifies the root element. The parameters in `List of elements to be returned` specify the elements which are wanted to be extracted. The identification of such an element is realized
with the XPath notation. The parameters in *List of output fields to be returned define the names of the output columns, whereby the position within the list must be the same as the corresponding position of the extracted XML element.

Figure 66: Parameters of XMLNormalizer for the extraction into ProcessEvent

Figure 67 shows the extraction of the extendedDataElements. If a CBE has multiple extendedDataElements the operator outputs the same number of tuples, whereby each tuple describes one ExtendedDataElements element.
In order to use the `creationTime` as a timestamp, the resulting element extracted in the XMLNormalizer has to be converted into the type `timestamp`. Therefore the string expressing the `creationTime` has to be cut first, because the `creationTime` is of the type `date`, standardized by W3C. In the type `date`, the date and the time of a day are separated by a `T`, like for example `1997-07-16T19:20+01:00` [W3C 1997]. StreamBase was not able so far to convert this `date` format. Therefore the separator `T` was replaced, which enables the conversion of the resulting date to the internal timestamp type of StreamBase. The described integration steps of the CBE are shown in Figure 68.
7.2.4 BAM-view: “Loss by canceled credit applications”

The parametrizable sub module AmountMonitor produces the metrics of the BAM-view, whereby it detects canceled applications with the pattern “time exceeding of the beginning/end of a credit application” (see para. 5.4.1.1). The following parameters allow a fast reconfiguration of the module on process changes of the easyCredit application.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>&quot;sys:eb:workflow:Internet:Gast:Local&quot;</td>
</tr>
<tr>
<td>enMethod</td>
<td>&quot;getProfit&quot;</td>
</tr>
<tr>
<td>exit</td>
<td>&quot;sys:eb:workflow:Internet:Gast:Local&quot;</td>
</tr>
<tr>
<td>exMethod</td>
<td>&quot;antrag:kredit:person&quot;</td>
</tr>
<tr>
<td>amountParameter</td>
<td>&quot;kredit:Wunsch&quot;</td>
</tr>
<tr>
<td>appTime</td>
<td>minutes(30)</td>
</tr>
<tr>
<td>yellowThreshold</td>
<td>200000</td>
</tr>
<tr>
<td>redThreshold</td>
<td>500000</td>
</tr>
</tbody>
</table>

![Figure 69: Parameters for the AmountMonitor](image)

A new credit application instance is identified by the parameters $entry$ and $enMethod$, representing the first called session bean. A credit application is finished by the correct execution of a session bean, identified by $exit$ and $exMethod$. The $amountParameter$ identifies an extendedDataElements event which includes the credit amount. The $appTime$ is the time window, in which a credit must be applied. Based on these parameters, canceled credit applications can be detected and aggregated to build the desired metrics. An exceeding of the $yellowThreshold$ or $redThreshold$ parameters is the condition of interest for the alert generation. The detection of canceled credit applications is difficult, because StreamBase does not yet offer functionalities to define complex scenarios or to recognize negative events. The case of not occurring events resp. negative events cannot be resolved with StreamBase. Mechanisms like exceeding time windows do not exist. Thus scenarios like “credit application entry followed by no credit application exit” cannot be described with a corresponding pattern. An alternative solution with StreamBase is the usage of the internal Query Table. It is basically a data base table which can be used in StreamBase CEP applications. In order to represent a credit application within a Query Table, the following schema elements are needed. The $threadId$
identifies a credit application. The location describes the application server on which the credit application is being processed. The component and method describe the last called method of a session bean during a credit application. The column page stands for the last visited JSP during a credit application. The amount is the current credit amount of a credit application.

![Figure 70: Schema of the Query Table Application](image)

The entering events are filtered in the operator Filter_Manipulation and implement a database manipulation depending upon their contents.

![Figure 71: Filter settings of the operator “Filter_Manipulation”](image)

The events matching the criteria for output port 1 perform a database entry for a new credit application. The events of output port 2 perform a database update which identifies that a credit application is finished. The events of port 3 update the Application table with the current called session bean, its method and its application server. Events of output port 3 update the table with the current visited JSP. The events of the ExtendedDataElements stream will be filtered for events containing the credit amount. Therefore an ExtendedDataElements event will be filtered for its name properties as seen below.
StreamBase offers an operator called Metronome which performs an output of the internal system time continuously. The continuous output triggers a database query on all tuples of the Application table. The resulting entries of the query will be filtered for the conditions, identifying a canceled application. Therefore an entry of the Application table is checked, if it contains a value in the column exit_creationTime and if the interval between the internal system time produced by the Metronome and the entry_creationTime is bigger than parameter $appTime$. If the condition is fulfilled, the application is canceled. The second output port determines successfully applied credits.

The determined canceled and successful applications will be deleted from the Application table in order to prevent a second determination of the application’s status. The determined applications will be aggregated to build the metrics described in para. 5.2.1.1. Therefore the credit amount of the last 24 hours will be summarized. Additionally the average and maximum amount of canceled credit applications will be determined. The determination of the most affected session bean, application server and JSP will be done in two. First the affected components will be counted and then the maximum of those numbers will be determined. The results will be joined in several steps and mapped onto readable names, because the operator join can just join two streams and enhances prefixes. The application diagram which implements the described actions, outputs the BAM metrics and canceled applications itself for drill-down purposes.
Additionally the AmountMonitor produces alerts in terms of emails, when the thresholds $redThreshold or $yellowThreshold are passed.

![Application diagram of the AmountMonitor](image)

**Figure 74:** Application diagram of the AmountMonitor

### 7.2.5 BAM-view: “Performance monitoring of a service”

For the implementation of a reusable monitoring module there is a need for several parameters. The parameter $component identifies component related events. Time windows for patterns, which identify a correct method completion, are based on the parameters $avgTime and $maxTime. The aggregation functions are based on the $observationPeriod parameter. The threshold parameters $redThreshold and $yellowThreshold are used as conditions for the alert generation.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>(none)</td>
</tr>
<tr>
<td>avgDuration</td>
<td>seconds(5)</td>
</tr>
<tr>
<td>maxDuration</td>
<td>seconds(10)</td>
</tr>
<tr>
<td>yellowThreshold</td>
<td>20</td>
</tr>
<tr>
<td>redThreshold</td>
<td>2</td>
</tr>
<tr>
<td>observationPeriod</td>
<td>120</td>
</tr>
</tbody>
</table>

![Parameters for the ComponentMonitor](image)

**Figure 75:** Parameters for the ComponentMonitor
The reconsideration that StreamBase doesn’t deliver functionalities to define patterns respectively negative events, results in the usage of a Query Table.

The Component table represents the methods of a component. The column id is the primary index. The component, extId, method and threadId identify a component’s method uniquely and thus they are used as secondary index for query purposes. The location stands for the application server, where the method is executed. The duration will be calculated on the base of the creationTime properties of the StopSituation event minus the StartSituation event. The state will be set on completed, yellow or red in dependency of the method call’s behaviour.

The incoming events of theProcessEvent stream are split dependent on their situation description, which is either a StartSituation or a StopSituation. Events describing a StartSituation generate a new query table entry, which represents a new method call of the event. The event’s attribute globalInstanceId will be set as the primary identifier “id” for the method and the attribute creationTime is written into the table column “duration”. The properties threadId, component, method, extId, location are written in the corresponding columns. Additionally the events are aggregated to the number of method calls based on the time window $observationPeriod. Events describing a StopSituation execute a query on the Component table entry, which has the same attributes threadId, method and extId. Based on the resulting id, the entry’s attribute duration is updated with the events creationTime minus the creationTime of the corresponding

![Component (Query Table)](image)

Figure 76: Component Query Table
StartSituation event which is stored in the table column duration. The entry’s column state will be set on status completed. Additionally the resulting entry will be deleted and aggregated to the number of completed method calls based on the time window $\text{observationPeriod}$. The metronome executes a query on all entries of the Component table. The entries are checked for the condition “time-duration > $\{\text{maxDuration}\}$”. Those entries, which fulfil the condition, are set to state red and will be deleted from the Component table. They will be also aggregated to the number of red methods. Entries, which do not fulfil the condition, are checked for “time-duration > $\{\text{avgDuration}\}$”. If they fulfil this condition, they are set to state yellow and will be also aggregated to the number of yellow methods based on the time window $\text{observationPeriod}$. The resulting red and yellow method entries will be routed to the output stream SuspiciousMethods. The aggregations will be merged to build the BAM metrics.

![BAMmetrics (Output Stream)](image.png)

**Figure 77:** BAM metrics

Figure 78 shows the operator Filter_Alert, which is applied on the BAM metrics and triggers emails, if the condition of interest $\text{YellowPercentage} > \{\text{yellowThreshold}\} \text{ OR } \text{RedPercentage} > \{\text{redThreshold}\}$ is fulfilled.
7.2.6 Problems and conclusion

In the author’s view the development of patterns with tables, which are queried by polling on the table, does not correspond to the CEP paradigm. The complexity of such tables and the associated queries rise fast, if there are more complex scenarios which have to be described. An operator which allows such a description would solve the problem. Besides this, StreamBase offers all capabilities to develop CEP based BAM tools. Therefore it provides a SQL based approach and a GUI based approach, adapters and APIs to integrate into enterprise environments and to monitor them. The projects can be deployed with one click and the test and debugging capabilities of the StreamBase Studio leverages results fast because all outputs of the operators and streams can be tracked in the debugging mode of the server. Problems may occur on bigger projects such as monitoring projects for hundreds to thousands of different processes, which need a high flexibility and adaptability. A vision of BAM is to let the operational department build their own custom alerts, conditions of interest and custom drill-down views without programming queries. These functionalities have to be coded instead of using flexible interfaces. The capability of using custom Java operators within StreamBase may solve the problem, but this could not be observed in this
thesis. Process changes have either be corrected or adopted with a very flexible design. Therefore a very high modularisation is needed and may lead to less transparency and more complexity. The graphical approach may help to execute these changes faster.

7.3 TIBCO

Unlike most other vendors, TIBCO does not focus in event processing as stand-alone activity despite the fact that TIBCO BusinessEvents is a stand-alone CEP product. TIBCO instead sees their BusinessEvents application as part of an enterprise-wide business optimization and of transcribing the concepts of IT-insight (see para. 3.1.4) and predictive business (see para. 3.1.5) [Vinayak 2006], [TIBCO 2004a], [TIBCO 2004b]. BusinessEvents has to be seen as part of a broader product set which is used to handle all necessary event-driven tasks. The consecutively paragraphs will provide an overview of the design of TIBCO’s architecture structure as well as a detailed implementation overview according to the requirements.

7.3.1 Architecture

The TIBCO architecture consists of multiple applications and components; each of them working independently as well as together. Those applications include inter alia TIBCO BusinessEvents [TIBCO 2006b], TIBCO Rendezvous [TIBCO 2006g], TIBCO EMS [TIBCO 2006h], TIBCO Business Works [TIBCO 2006i] and the TIBCO Designer. In the context of this thesis, the relevant components/applications involved the TIBCO Designer as an easy to use graphical user interface for creating integration projects, TIBCO BusinessWorks as an extensible integration platform for developing and executing the relevant processes and TIBCO BusinessEvents for processing complex events and for mapping the event’s data into the TIBCO specific data-model. TIBCO EMS provides all necessary adaptors and messaging technologies. For visualization purposes TIBCO RTView [TIBCO 2006j] was utilized. The following points will
describe the named TIBCO applications in a more detailed view besides the way they are realized.

7.3.1.1 General Architecture Concept Components

In the following the four TIBCO products are described in detail:

7.3.1.1.1 TIBCO BusinessEvents

BusinessEvents is the company’s event processing product. It was developed in conjunction with Vodaphone [Newswire 2006]. As Figure 79 shows that the development in BusinessEvents is based on a three level process. Every level consumes and distributes events through channels. TIBCO BusinessEvents includes predefined drivers for three kinds of (event) channels:

- JMS
- TIBCO Rendezvous
- Local

Each channel can have multiple destinations, whereby each destination is associated with a default simple event. Similarly, simple events are configured with a default destination channel. TIBCO BusinessEvents also includes local channels, which allow multiple rule sessions to pass events between each other. In view of the use case requirements, the JMS channel was the only channel which had to be set up in order to receive and publish events. The three main components of BusinessEvents consist of an UML based concept modeler where static relationships are being captured, state and object modelers, which graphically represent objects in the business and their dynamic relationships and a rules engine that provides inference capabilities [TIBCO 2006b].
This model-driven approach is being used to filter, collect and correlate events with respective on business processes. By applying event rules, certain situations can be identified and responses upon those situations can be executed. Therefore BusinessEvents is either able to adapt already running processes, design new processes and to notify the relevant people when human intervention is necessary [TIBCO 2006b], [TIBCO 2006c].

7.3.1.1.2 TIBCO Designer

The TIBCO designer is a graphical user interface for designing and creating the integration project configurations. The designer features the possibility to drag and drop components into a project and to build an Enterprise Archive (EAR) of the project. This EAR can then be used by TIBCO Administrator for deploying and running the application. The TIBCO Designer is not installed standalone, but in conjunction with other TIBCO products. When installing a product that uses TIBCO Designer like TIBCO EMS, TIBCO BusinessWorks or TIBCO BusinessEvents, palettes are added to TIBCO Designer. Depending on the installed product, it is possible, for example, to use the TIBCO Designer in order to model TIBCO BusinessWorks process definitions or create or modify adaptor
configurations accordingly to the requirements. Palettes are used to group configuration objects. The TIBCO Designer palette reference provides standard palettes that are always available from TIBCO Designer, like for example the enterprise archive palette which handles the building of the project. In this thesis the Designer version 5.5 was applied.

7.3.1.1.3 TIBCO BusinessWorks

TIBCO BusinessWorks is an integration platform that allows the development, deployment, and execution of integration projects. TIBCO BusinessWorks also includes an engine that executes the designed processes. Like mentioned in para. 7.3.1.1.2, BusinessWorks extends the pool of the provided pallet options.

TIBCO BusinessWork’s key components according to the use case requirements were:

- The TIBCO Designer GUI supports adaptor configuration, process design, and testing of the integration project.
- The TIBCO BusinessWorks engine runs the business processes in test mode and at run-time.
- TIBCO Administrator supports deployment, security administration, and monitoring and management of processes and machines.

7.3.1.1.3.1 TIBCO Enterprise Messaging Service

The TIBCO Enterprise Message Service (EMS) software lets application programs send and receive messages according to the JMS specification which was featured in this thesis. The TIBCO EMS server acts as an intermediary for JMS messages and manages its delivery to the correct destination. Using the message service allows an integration of the required applications in this thesis. Figure 80 illustrates an application producing a message, sending it by way of the server, and a different application receiving the message.
Like mentioned in para. 2.3, JMS supports two different messaging models. The point-to-point model (queues) and the publish and subscribe model (topics). Based on the specifications seen in Figure 81 of distributing the CBEs via JMS pub/sub is accomplished. Therefore the TIBCO EMS subscribes to the topic and acts instantaneously to delivered message.

7.3.2 Data import

As mentioned in the previous paragraphs, TIBCO provides a huge variety of input- and output adaptors. These can be applied particularly by using the provided palettes. The implementation of the integration of CBEs requires the “JMS Topic Subscriber” palette-item for opening a subscription to the defined JMS topic “EventCloud” of the JBoss application server and for test purposes the “File Poller” palette-item of the file palette. Figure 82 and Figure 83 illustrate the design of these implemented processes.
After configuring the data import palettes – like giving the path of the log file used by the “File Poller” respectively assigning the JMS palette to a defined JMS connection – the data palettes can be used to initialize the process for receiving the events; constituted by the green “play” icon. After initializing the data receiving process, the events are passed to the process responsible for the parsing of the received CBE-conform XML strings against the CBE schema. The “assign” function palette-item seen in Figure 82 is used to “mark” every line of the connected log file as a CBE-event, which is then being passed to the process responsible for the parsing as seen on Figure 84.
this, the event will automatically put into the accordant TIBCO-intern event input channel, seen in Figure 85.

![TIBCO process of mapping the relevant information on the “creditApplicationEVT.event”](image)

Figure 85: TIBCO process of mapping the relevant information on the “creditApplicationEVT.event”

Three kinds of events within TIBCO BusinessEvents can be processed. A “simple event” which is the representation of a single activity the occurred at a single point of time, a “timer event” which can be seen as a timer, and an “advisory event” which is a notice generated by BusinessEvents to report an activity in the engine, e.g. an exception. Except for the advisory event all event types are utilized in this thesis. This set of CBE-properties related to the given activity of the reception of a new message is a typical example of a simple event type [TIBCO 2006c].

7.3.3 Rules and concepts

In BusinessEvents activities within the enterprise applications are detected by rules which dictate the appropriate actions according to the particular situation. Actions can be the creation, modification or destruction of simple events, concept instances, time events, score cards, or a combination these objects. All rules exist inside of rule sets. Rule sets are groups of rules. At deployment time, rule sets can
be selected to use at runtime; an individual selection of rules is not possible. The BusinessEvents rule editor provides separate work areas for condition and action statements. Within the conditions work area, each statement has its own line and is joined by an unseen “AND” operator [TIBCO 2006c]. Within the actions work area, semicolons separate statements; one statement can span multiple lines; one line can include multiple statements. The statements are described, using TIBCO’s proprietary Java-similar EPL (see chap. 8ff). Every received event is inserted into the TIBCO intern “JMS input event channel” which enables the application of particular rules for each event type sent into the channel. These rules, like every other palette-item, can be set up just by dragging them onto the designer panel. Their configuration happens via individual parameterization respectively coding as seen at the example of the createNewCaConcept rule in Figure 86.

```
.Rule RuleSet CreateNewCaConcept

<table>
<thead>
<tr>
<th>Form</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Events/CreditApplicationEVT</td>
<td>creditapplicationevent</td>
</tr>
<tr>
<td>/Concepts/CreditApplication</td>
<td>creditapplication</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>creditapplicationevent.ID :: creditapplication.ID</td>
</tr>
</tbody>
</table>
| creditapplicationevent.Situation :: "StartSITUATION"
| creditapplicationevent.Page :: "InitPage" |

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreditApplication (a = instance AppliesTo(creditapplicationevent.ID));</td>
</tr>
<tr>
<td>if (a == null)</td>
</tr>
<tr>
<td>Instance.createInstance (&quot;Concepts/CreditApplication&quot;)</td>
</tr>
<tr>
<td>System.out.println(&quot;New application initialized: &quot; + creditapplicationevent.ID + &quot; - &quot; + creditapplicationevent.StartCT);</td>
</tr>
<tr>
<td>conceptstat(creditapplicationevent, ComponentType, creditapplicationevent, ComponentType);</td>
</tr>
</tbody>
</table>
```

Figure 86: Every rule can be applied to multiple event types respectively concepts

As seen in Figure 86, the rule createNewCaConcept is being triggered on the occurrence of an instance of the event type CreditApplicationEVT. This rule triggers its proprietary Java-based code in the action part, if the determined conditions of the received event are true. At this rule example seen on Figure 86, a new instance of CreditApplication concept is created. Concepts can have three characteristics:
Each concept is a definition of a set of properties that represent the data fields of an application entity. They can also describe relationships among themselves. For example, an order concept might have a parent/child relationship with an item concept or a credit application concept might be related to an applicant concept based on the shared property, customer_id. Furthermore concepts provide the possibility of including a state model [TIBCO 2006c]. The CreditApplication concept is similar to a class used in Object Oriented Programming. Therefore every credit application is represented by an instance of the concept CreditApplication. It can also be seen at the rule example createNewCaConcept that after the creation of the concept, the rule started() is being executed, getting passed the parameters componentType and component which represent the type of the component of which’s request initialized the received event and its identifier. Through modification the concepts by rules, the visualization via TIBCO RTView is possible. All relevant metrics like:

- the loss by canceled credit applications within the time interval t,
- the status of every active credit application within the easyCredit-Internet application,
- the status of every deployed StatelessSessionBean component,
- the average throughput time for every successfully completed credit application step, either per component, JSP or the whole process,
- the number of active users per deployed JSP, for frontend monitoring purposes

can be extracted from the active stats concepts within the project. Figure 87 illustrates the configuration respectively parameterization of the stats-concept by means of a screenshot.
Figure 87: Screenshot of the statistic-concepts within TIBCO Designer

The screenshot seen in Figure 87 displays the layout of the TIBCO Designer with focus on the Stats-concept, which is separated in five sections:

1. The Designer panel seen here is used to graphically implement every utilized component of the provided palette repository by dragging and dropping it onto the panel, followed by the particular configuration. As seen in Figure 87, the concepts Stats and StatsHolder were from the BusinessEvents palette implemented and configured.

2. This section displays the setup of the project. Every component is contained in the particular folder, concepts in the concept folder, processes in the process folder and so on. Cross-referencing of components within the project can be made by referencing to the folder name and the ID of the component.

3. All available palettes can be accessed here. As seen in the lower part of section three, various palettes are available.

4. Here all necessary configurations, parameterizations, naming, variables and other options can be set for each palette-item.
5. This section displays the function-argument-mapping. Here the relevant arguments and variables of each component within the project can be mapped.

7.3.4 Presentation

The presentation of all relevant statistical information in real-time is visualized using the TIBCO application RTView. Every statistic concept created or updated by triggered rules and queries of event patterns are sent into the TIBCO-intern JMS output channel. This output channel is connected to the RTView application which, in turn, can be configured in several steps to display all required data in real-time. The forms of the data respectively situation presentation can either be chosen or modified from pre-defined charts, views and dashboard portfolios, or can be completely developed from the scratch. Figure 88 displays a chart provided by TIBCO’s RTView.

![Figure 88: Example of visualization options featured by TIBCO RTView](image)

Besides using the dashboard- and presentation application RTView, it is possible due to the variety of provided output adaptors of TIBCO BE to send the correlated (event) data to any other application for presentation purposes, like to the JFreeChart-based cockpit as described in para. 7.5ff.

7.3.5 Conclusion

TIBCO offers a huge portfolio for implementing CEP-based BAM. According to the requirements of this thesis, four components were utilized; TIBCO EMS, TIBCO BE, TIBCO BW and TIBCO RTView. TIBCO BE correlated all events correctly, based on the requirements. Due to the implementation of the TIBCO specific visualization tool RTView, a presentation of the correlated data is
possible. RTView provides a variety of charts, dashboards and other visualization options in order to display the relevant metrics like “loss by cancelled credit applications” or performance and status of the deployed business logic components respectively JSPs. The messaging application TIBCO EMS provides multiple adaptors for all incoming and outgoing event-types. When increasing the amount of incoming CBEs, TIBCO’s BE processed well, according to the applied rules and concepts. With TIBCO BW it was possible to create, orchestrate and execute all relevant processes.

7.4 Systar / Claresco

"Competition is going to be defined by those companies that have best-in-class processes."

Michael James Melenovsky, Research Director Gartner

In this thesis the BusinessBridge portfolio is evaluated to reach the goal of enabling a seamless transition of business process and transaction for a better IT-insight in terms of bridging the IT with the business contents and activities (see para. 3.1.4). The following para. 7.4.1 will display the architecture structure Systar’s product BusinessBridge with regards to the requirements followed by a detailed implementation overview.

7.4.1 Architecture

The architecture of Systar’s BusinessBridge is based on and composed with multiple principle parts and components.

- The presentation component part is built around JSPs/JSFs [Sun 2006d]. These are designed, deployed and running on application servers. Like inter alia the WebSphere application server (WAS) [IBM 2007] or the JBossAS [RedHat 2007]. In order to visualize the relevant data, the JSPs have to be customized for every single concept or project which is done on templates which are based on industry experiences.
• For visualization purposes multiple JSPs are created respectively modified for developing the individual required point of view. The defined JSPs are published through a Web server which is compatible with the WAS like the Microsoft Internet Information Server (IIS) [Microsoft 2006a].

• The logical part is set up, built and executed with the BusinessBridge server. In this part, the rules and patterns have to be defined in order to realize the intended queries or filtering. This logic part can be accessed through an XML Web service which is based on Microsoft’s .Net Framework [Wikipedia 2007a] and the Microsoft Internet Information Server.

• The Datasource layer provides a non intrusive way of data acquisition. These data sources are able to wrap and manage communication with a great variety of application sources and systems. The captured data and events can be filtered to avoid processing non useful or non significant captured data. They are then mapped into the BusinessBridge’s common object format and injected into the next layer: the correlation engine. Datasources can either receive events, working in a “listen” mode, like what is used to read JMSLogger logfile or to subscribe to events sent by a JMS topic. Most data sources are bi-directional, which enables BusinessBridge NG to communicate with all layers in both ways. Systar has created more than 90 types of data source adaptors to date.

• The optional part of Business Bridge’s general architecture is the historic part. This repository is based on Relational Database Management Systems (RDBMS) and can be accessed through the Open Database Connectivity (ODBC) [Linux 2006] which provides a standard software API function for using RDBMS [Claresco 2007c]. This part was not utilized.

7.4.1.1 General architecture concept components

The Enterprise Information System (EIS) [IEEE 2004] provides services which support the executions of the business processes and offers various network connectivities. The EIS is connected with multiple data source services, which, in
Implementation of the enterprise cockpit and the BAM-views

return, feed BusinessBridge’s rules database, interface engines and object databases of the correlation engine. As seen on Figure 89, this server is connected with a Web interface via Web service interfaces. The Web interface is for visualization purposes only and can provide various information and views for the differing requirements.

Figure 89: Three tier architecture of Systar BusinessBridge [Claresco 2007a]

Figure 89 gives an overview of the three general layers Systar uses for retrieving, processing and displaying the right information at the right time to the right persons [Hayes-Roth 2007]. Each layer consists of multiple key components [Claresco 2007c].
• The information acquisition layer:
  This Systar windows-service acts as the data source service. As illustrated in Figure 89, it captures data in the client information system via a family of data sources. Via adapters those data sources are able to encapsulate and manage the communication with a variety of system and application assets within the Microsoft IIS. The captured data then are filtered in order to avoid the processing of non useful or non significant captured data and are then transformed into a common object format and injected into the correlation engine of the next higher layer.

• The correlation engine layer:
  This layer contains the Systar MS Windows service “correlation engine” which is a real-time based, object-oriented inference engine. The correlated input data which have been sent via the data source component is dynamically determined by the engine. Occurrences of situations are communicated to the upper presentation layer for visualization purposes per dashboards. Rule bases can be enriched, depending on the nature of solutions and provide the determination of relevant situations by the engine together with the creation and management of as well business as technical objects based on the specific requirements. Like mentioned above, the option of using historical data which are stored and managed by BusinessBridge in an Oracle-, DB2- or SQL server-repository was abandoned.

• The presentation layer:
  To display the correlated data properly to multiple different consumers, the BusinessBridge architecture features a presentation layer where dynamic Web applications are implemented to provide various views and dashboards. Open and standardized Java based technologies such as JSPs or JSFs [Sun 2006d] are used for the page rendering and design. Each main dashboard is a Web page. The logic content is obtained via access to objects and situations provided continuously by the correlation engine layer as described in relation to the requirements in para. 7.4.4.2.

• Other components:
Components like the provided Web service for simultaneous requests from multiple presentation layers, log services for automatically logging, administration services for centralized access to shared tools were not specially utilized or customized in this thesis, as well as the alarm service. This service is able to send notifications and alarms via SMS, email or phone on reaching certain threshold levels of single components. This fits in the concept of VIRT, but was not realized in this thesis (see para. 4.4, [Ammon 2006a], [Hayes-Roth 2007], [Claresco 2007c]).

7.4.1.2 Flows

As illustrated in Figure 90 data and information flowing through the BusinessBridge architecture can pass several individual steps and components throughout all three Systar’s BusinessBridge layers [Claresco 2007c].

![Data and information flow within Systar BusinessBridge architecture](image)

**Figure 90: Data and information flow within Systar BusinessBridge architecture [Systar 2007b]**

Step 1: Browser ➔ http Server
Through http browsers internal and external users can connect to the http server through https-protocol.
Step 2: WAS ➔ LDAP
In this step the access can be granted to individual user groups via Lightweight Directory Access Protocol (LDAP) [Virtanen 2003]. This optional step of a secured and authenticated access for multiple user groups was not configured and implemented.

Step 3: WAS —→ XML Web service
Every single BusinessBridge NG application opens a SOAP http connection to Web services. As seen in Figure 90, the Web services are based upon the Microsoft .Net Framework and the IIS. Web services collect data which have to be displayed on the tied Web pages.

Step 4: Web service —→ BBR Services
Within this step the data is passed onto the provided BusinessBridge Windows services (BBR Services). These include the administration services, the data source services as well as the correlation engine. This client/server communication inside BusinessBridge is based on standardized TCP/IP socket connections.

Step 5/6: Correlation Engine —→ ODBC Management —→ RDBMS
The BusinessBridge applications communicate with a dedicated database via the ODBC interface. For every ODBC connection a specific Data Source Name (DSN) [Wikipedia 2007b] is required. Via DSN, information of the specific database can be retrieved.

Step 7: GUI —→ Shared registry
To collect setup parameters from the BusinessBridge Server (the registry server), each GUI or client can:

- send queries to the BusinessBridge administration service.
- try to establish a direct access to the shared registry in case of a connection failure towards the administration service due to e.g. maintenance purposes. This access however is just granted, if the user account obtains the relevant account rights under the specific sub tree of the registry [Claresco 2007c].

Step 8: GUI —→ Shared files
Implementation of the enterprise cockpit and the BAM-views

Each GUI has to gain access to the specific files within the registry server. Therefore the sources of the BusinessBridge DataSource Builder define special DataSource scripts.

7.4.2 Adaptors

Systar’s BusinessBridge offers a variety of connection adaptors for receiving or sending data, information or events. The data connection is set up and configured in the BusinessBridge DataSource component. As seen on Figure 93, the BusinessBridge Log Reader and the JMS Gateway DataSource were implemented in this thesis to receive the created CBEs (see para. 5.3) via subscription to the relevant JMS topic or by processing the tied log file. Each adaptor is based on common standards which minimizes the intrusion into the target infrastructure. The adaptors are bidirectional and support active respectively passive collectors. Figure 91 gives an overview of the possibilities for data input or output [Claresco 2007a].

![Figure 91: BusinessBridge’s DataSource Adaptors [Claresco 2007a]](image)

7.4.3 Event Abstraction

BusinessBridge is the only evaluated real-time CEP-based BAM platform which supports the service of Expertise modules and functionalities based on a historical BAM knowledge-base [Claresco 2007a]. The expertise modules offer customers a
best practice approach and include knowledge that has been gained in over 300 implementations during the last years. Typical monitoring questions may be answered out of the box. Figure 92 displays graphically the usage of BusinessBridge’s Expertise modules.

![BusinessBridge BAM Expertise modules and templates for simplification of the BAM-view creation and event abstraction](image)

Figure 92: BusinessBridge BAM Expertise modules and templates for simplification of the BAM-view creation and event abstraction [Claresco 2007a]

The events are abstracted and correlated within BusinessBridge’s correlation engine. Rules for the event abstraction are defined in BusinessBridge’s Bridget Builder. Multiple different components of the Bridget Builder were utilized in this thesis [Claresco 2007c].

- **Alarms**: for defining alarm situations when reaching certain thresholds.
- **Bridgets**: which contain the Expertise modules. Based on these already predefined templates, the situation is implemented by customizing and expanding the Expertise modules in order to utilize the needed features. Next to the use case specifications, this approach offered many more options than needed.
- **Entities** consist of different entity types which either can be adopted for the specific use case from Expertise modules or new created for individual requirements.
- **Indicators** can be defined upon certain situations.
- **Rules** consist of the relevant rules sets for the different event types.

7.4.4 Implementation of the use case “easyCredit-Internet”
Following paragraphs illustrate the set up of implementation of the easyCredit-Internet case study, accordant to the requirements, using Systar’s BusinessBridge CEP-based BAM application.

### 7.4.4.1 Data import

For importing the data into BusinessBridge, the JMS input adaptor and a log file adaptor – for replaying CBEs sent to the JMS topic from a log file (see para. 6.2.5) – are implemented for this thesis. Like shown in Figure 93 the connected data sources are just added to the specific project. This way, they can be obtained as data source of the relevant project within the BridgetBuilder.

![Established data source connections with BusinessBridge’s DataSource Builder 4.1](image)

As seen in the screenshot, the above embedded data sources can be specified and configured via the Service Device Parameter mask. There the reference of the used log file is specified which contains the CBE entries, as well as the scanning mode or the time interval for handling the input rate of every contained CBE-conform XML string. On the other hand parameters of the connection type (queue or topic) or its JNDI name are set in this mask.
7.4.4.2 Pattern modeling and information retrieval via the Bridget Builder

The Bridget Builder is the part of the Systar’s BusinessBridge technology that constructs the relevant Bridget Models. This model uses events and conditions, gathered from the specified data sources (like described in para. 7.4.4.1) and transforms, filters and edits the raw event data into significant data for the supervision of the business processes and for the visualization via dashboards. This can be seen on Figure 94.

Properties of every incoming CBE via the designed DataSource dsEasyCreditCBEFile can be extracted from out the XML structure and mapped onto the relevant internal variables of the instances of CustomDataClass cdcCBEJMSEvent. This mapping is made possible by referring the matching CBE XML schema as a custom property of the particular data source. It is also possible to parse the incoming events against given schemas. It is modeled in the Bridget Designer how the relevant elements are mapped. To define the necessary pattern rules the “Domain tab” of the Bridget Builder has to be highlighted. All required pattern rules are defined in the BusinessBridge’s specific rule language.
like described in chap. 8ff. Figure 95 shows a clipping of some of the relevant rules for instantiation of an implemented entity class.

```java
When cdcCBEJMSEvent
Do

Use myEasyCredit As ecEasyCredit
Use appServer As ecappServer
myEasyCredit = Get ecEasyCredit (Name = cdcCBEJMSEvent.ThreadID)

LogMessage (ID, "cdcCBEJMSEvent.ThreadID = " + cdcCBEJMSEvent.ThreadID
   + " cdcCBEJMSEvent.ExtractValue = " + cdcCBEJMSEvent.ExtractValue
   + " cdcCBEJMSEvent.SituationCategory = " + cdcCBEJMSEvent.SituationCategory
)

If myEasyCredit.CreateTime_ST Is NullData Then
   Method myEasyCredit.Second_ST()
End If

myEasyCredit.SituationCategory = cdcCBEJMSEvent.SituationCategory
```

Figure 95: Snippet of the pattern rule triggering on arrival of a new CBE event in Business Builder with focus on the entity class declaration

On arrival of a new CBE message cdcCBEJMSEvent via the defined data source input, the new instance myEasyCredit of the internal BusinessBridge CBE ecEasyCredit class Entity is being declared. These entities can either be designed from the scratch or being derived and modified – like for this thesis – from an already predefined template repository, as mentioned in para. 7.4ff. Thus BusinessBridge provides multiple templates respectively Expertise modules (as seen in para. 7.4.3), an adequate base structure, containing various entities for nearly every situation, can be implemented and configured as highlighted in Figure 95. As already mentioned in para. 5.3ff, every entity instance has an individual identifier threadId. Every instance of the entity class ecEasyCredit represents a credit application with parameters depending on the received CBEs. After that a log message is created, consisting all relevant CBE information. Now a number of statements are executed on the myEasyCredit instance in order to extract the attributes resp. elements. By that the variables of each application instance will be set step by step. Due to the decoupled implementation of the rules, like seen in Figure 96, and the object model, such as entities of the type ecEasyCredit, the CEP application could be managed and maintained without much afford.
Implementation of the enterprise cockpit and the BAM-views

7.4.5 Presentation

The presentation layer is separated from its logic part and relies on a dynamic Web application structure. It makes use of Web technologies like the utilized JSPs. After evaluating the retrieved data, based on the implemented rules and events to determine the status of the easyCredit application respectively each discrete credit application, an enterprise cockpit can be utilized and deployed. Here either metrics for single deployed components can be visualized or an abstract view on the easyCredit business process as a whole can be provided. Drill-down options can be called on, in order to get a more detailed view; either on single active credit applications, single dialog steps or components containing the business logic. As seen on Figure 97, by importing the project interchange of the project, the step$PAY_diagram_ST can be created, using the correlated data passed by the Bridget Model. This way all dialog steps (defined in JSPs) of the user interaction layer of the easyCredit business process application can be displayed and monitored in real-time.
To drill-down for monitoring all momentarily active users of a dialog step a click on the appropriate step opens a new JSP which displays in detail how many users are viewing the dialog step at the moment with cross-reference on their particular credit application. Furthermore these interactive icons can indicate possible critical frontend situations by changing the color to yellow respectively red if predefined thresholds are reached while automatically sending a notification to the actual responsible person in charge. Based on individual threshold values for every component, a color indicates the status of every implemented stateless session bean.

7.4.6 Conclusion

Systar’s product BusinessBridge is the only product which is not explicitly rendered as CEP platform, but it is a BAM tool, based on CEP. For this master thesis all CEP functionalities are available and could be realized using BusinessBridge. Since the CEP-based BAM tool BusinessBridge provides a huge variety of already predefined templates respectively Expertise modules based on previous implementations, especially in the (online) banking environment [Claresco 2007b], the set up of the CEP-based BAM projects like for this thesis might be accelerated. Therefore the relevant Expertise modules have to be chosen and adopted according to the needed requirements. Since the whole framework
was already established by using predefined templates, the implementation or modification of all relevant components might be simplified. The detailed modifications and adoptions of the relevant entity classes, dashboards, data-input configurations or other components were conducted via masks or parameterization. However, prerequisite for such a simple and fast implementation is a comprehensive knowledge not only of the product’s architecture or EPL, but also of the available Expertise modules. Since the data presentation can be obtained using a standard Web browser, it offers compatibility assets and a huge availability for all relevant persons as well as from outside the company as from the inside. Because of Systar’s specialization in BAM, BusinessBridge offers a huge variety of visualization options which could not exercised totally in sense of this use case due time restrictions.

7.5 Java Dashboard

The enterprise cockpit is built as a single user application, whereby the implementation is done by using a custom adaptor for Coral8, JFreeChart and Swing. JFreeChart is an open source project, which was founded by Gilbert in February 2000 and is distributed under the terms of the GNU Lesser General Public Licence. It provides a Java chart library to develop charts in applications. Therefore it provides a consistent and well-documented API, a flexible design that is easy to extend and targets both server-side and client-side applications. It supports many output types, including Swing components, image files and vector graphics file formats [JFreeChart 2007]. As part of the Java Foundation Classes, Swing is a GUI toolkit for Java. It includes widgets such as text boxes, buttons, tables and lists. Generally, the Swing API is a complementary extension of the earlier released Abstract Window Toolkit. Since Swing components are written in pure Java, they run the same on all platforms, unlike the AWT which is tied to the underlying platform's windowing system [Wikipedia 2007e].
7.5.1 Architecture

The cockpit is designed to enable a quick replacement of the connected CEP platform. Because the cockpit just uses the abstract class `AdapterContainer` and the including abstract class `Adapter`, these two classes have to be extended with the functionality of the API in order to realize the data transfer from the CEP platform. In the class `Dashboard`, which initiates the cockpit application, one line of code has to be changed in order to plug in new adapters respectively a new CEP platform like `StreamBase`. The BAM-views of the cockpit are extended classes of the abstract class `Monitor` and are included in the class `MonitorContainer`. The class `Monitor` delivers a pointer to its parent container `MonitorContainer`. The `MonitorContainer` delivers a pointer to the abstract class `AdapterContainer`, which provides all adapters of the CEP platform via overriding of the particular methods [Sun 2006g]. Therewith, all BAM-view components have a connection to the custom adaptor, which enables them to retrieve the CEP platform’s data. The concrete implementation of the Coral8 adapters and BAM-views is explained in the next paragraphs.

7.5.2 Connection-Adapter – data mapping

The experimental implementation of the adaptor is done with the Coral8 API, which allows a subscription to defined streams in order to retrieve their data. Therefore it provides the class `com.coral8.toolbox.Toolbox` which provides a subscription `com.coral8.tuple.stream.Subscription` on a stream to an external application. This subscription allows the retrieving of messages from a stream in the form of the class `com.coral8.tuple.Tuple`. Thereby the elements of a `Tuple` are expressed in the internal data types of Coral8 such as `com.coral8.tuple.impl.TupleFieldValueFloat`. By using their `getValue()` method, these data types can easily be transformed into the primitive data types of Java. The adapters run as threads and process these incoming `Tuple` entries as they arrive. The processed data are stored in the container `HashMap`, whereby old data are replaced by current data. To propagate this transformed data, the adaptor provides the public method `getData()`, which
allows the BAM-view components to retrieve these data. The classes implementing the DrillDownAdapter store the data in an ArrayList; if an entry in the list exceeds a given time limit, it will be deleted. The AmountAdapter class transforms the message which includes the metrics for the BAM-view “loss by canceled applications”, the corresponding AmountDDAdapter delivers information about the affected applications. The class StepAdapter delivers the information about canceled applications related to a dialog step of the easyCredit application. The ComponentAdapter class transforms the message which includes the metrics for the BAM-view “performance monitoring of a service” and the corresponding ComponentDDAdapter delivers information about the affected methods.

7.5.3 Business monitor: Potential daily loss

The business monitor provides the BAM metrics for the view “Loss by canceled applications” (see para. 5.2.1.1). Therefore it visualizes the accumulated loss per day in terms of a bar chart. The bar chart is implemented by using the JFreeChart library. The colour of the bar indicates what the current status of easyCredit application is. If the daily loss passes a certain threshold, it changes its colour to green, yellow or red. Green indicates that the applications loss is within the given interval. Yellow indicates an approximation of the daily loss to the expected loss threshold. The colour changes to “red”, if this threshold is passed. In section “Details” the monitor also visualizes the other metrics which are determined in the CEP platform. Thus it shows the average loss by canceled applications, the maximum loss by canceled applications, the number of canceled applications and the exit points as information, why the applications could be canceled. Therefore it visualizes the session bean, where the most applications were canceled, and its number of cancelations. It also shows the application server, on which the most applications were canceled, and the number of cancelations. The business monitor runs as a thread and continuously updates the bar chart and the Swing elements by retrieving the desired metrics of the adaptor AmountAdapter. Figure 98 shows the business monitor and how it represents the particular metrics.
The button “Drill Down” opens a more detailed perspective on the easyCredit application. The drill-down view offers information about the dialog steps and their current status. Therefore it provides information, how many applications were canceled on a dialog step and the resulting lost sales volume. A table lists the particular canceled applications, which caused the entire status of the easyCredit application. It shows the threadId of the application, the credit amount, the payback volume, the last called session bean and the application server on which the application was processed. As shown in Figure 99, the button “Watch most affected Component” opens a technical view on the session bean, where the most applications were canceled.
7.5.4 Technical monitor: Session bean

The technical monitor is a reusable component, which visualizes the metrics of a session bean determined by the CEP platform (see para. 5.2.2.1). Therefore the monitor will be started with the fully qualified session bean name. Based on this parameter, the monitor runs as a thread and continuously retrieves the relating data from the ComponentAdapter class. The JFreeChart bar chart is used to show the percentages of yellow and red methods to all method requests. The percentage of yellow methods to the requested methods will be visualized in a yellow bar. A red bar shows the percentage of the red methods to all requested methods. The pane “Details” shows the monitoring parameters and the resultant metrics. The first visualized parameter is the “Observation Period”, the period on which the aggregations are based on. The parameters “Limitation Yellow” and “Limitation Red” are the thresholds for an exceeding of a method call.

The resulting metrics are “Incoming”, “Completed”, “Yellow States”, “Red States”, “Maximum Duration”, “Average Duration” and “Minimum Duration”. The parameter “Incoming” presents the number of requested methods. The line “Completed” shows the number of completed method calls. The “Yellow States” and “Red States” metrics display the method calls which did not complete within
the defined time intervals. The outputs in “Maximum Duration”, “Average Duration” and “Minimum Duration” exhibit the particular method completion durations of the session bean. Figure 100 shows a running monitor, which presents the performance of the session bean esys.ejb.workflow.InternetGastLocal. The button “Drill Down” starts a view on the particular “yellow” and “red” methods which caused the current status of the session bean.

![Image](image_url)

**Figure 100: Component Monitor for a session bean**

The drill-down view visualizes the name of the methods and their state changes to “yellow” and “red”. The element “Location” presents the application server on which the method was processed. Additionally it shows the affected credit application identified by its threadId.

![Image](image_url)

**Figure 101: Drill-down view on the session bean methods.**
8 Event Processing Languages

High thoughts must have high language.

Aristophanes, Frogs, 405 B.C.

This chapter gives an overview about the different event processing language approaches. A variety of CEP platform vendors will be classified depending on their programming language for event processing. The statements and results of the vendor’s representatives and business analysts will be evaluated and summarized. A clearly classification of all vendors to a particular approach is not possible, because some vendors provide different approaches within their platforms or the EPL is embedded into another language approach. In this thesis the EPLs are classified as “pseudo” SQL approach, special rule language or script language approach, Java approach, XML approach and GUI approach.

8.1 “Pseudo” SQL approaches

The main intention of vendors providing “SQL-based” approaches is that developers of relational databases can apply their knowledge based on SQL. These vendors offer language constructs of the SQL standard, which are well known in database expert groups. Common keywords as INSERT, SELECT, FROM and WHERE are used to filter and manipulate event streams and insert data from one stream into another one. They compare an event correlation with a join of database tables, whereby the join in the event stream platform takes place by joining streams [Coral8 2007a], [Coral8 2007b], [StreamBase 2007e], [Esper 2006a], [Esper 2006b], [Aleri 2006a]. The EPL of Coral8 is the continuous computing language CCL. Coral8 calls it “SQL-based”, because it provides these common keywords to process events. Additionally they offer a variety of non SQL expressions to accomplish the new paradigm of event processing. Therefore CCL has expressions to define windows CREATE WINDOW or KEEP 10 SECONDS, sequences or patterns MATCHING [10 Seconds: A, !B] or access on previous events PREV() [Coral8 2007a], [Coral8 2007b]. Figure 102 shows a processing statement.
The language StreamSQL of StreamBase is another proprietary approach, which is based on expressions, defined in the SQL standard. StreamSQL includes expressions like SELECT, FROM and WHERE, which have basically the same semantic as the corresponding SQL expressions. A complete statement needs the CEP enhancements like CREATE STREAM to define a stream or INTO to route events into a defined stream [StreamBase 2007c]:

```sql
CREATE STREAM completedApplication;
CREATE STREAM cancelledApplication;
SELECT * FROM read_all_rows
WHERE time-entry_creationTime>minutes(30) AND isnull(exit_creationTime) INTO cancelledApplication
WHERE notnull(entry_creationTime) AND notnull(exit_creationTime) INTO completedApplication;
```

The event query language EQL of the open source project Esper has a similar syntax as SQL. In addition, EQL offers capabilities for event stream analysis, especially the usage of event streams and a concept called views. The appendix .win:time(30 min) on a stream identifier creates such a view. Similar to tables in a SQL statement, views define the data of a stream which is available for querying and filtering. Other non SQL expressions are used to define scenarios of events such as every a=EventX -> every b=EventY(objectID=a.objectID). These stream analysis capabilities and scenario definitions can be combined into a statement like the following [Esper 2006a], [Esper 2006b]:

```
```
select a.id, count(*)
from pattern [ every a=Status -> (timer:interval(10 sec) and
not Status(id=a.id))]
group by id

The stream processing platform provider Aleri also features an EPL, which uses SQL keywords. The Aleri platform provides the implementation of CEP applications with the SQL based EPL Aleri SQL. Keywords like SELECT, WHERE, HAVING or GROUP BY are used to leverage the implementation of CEP applications for database developers [Aleri 2006a], [Aleri 2006b], [Yahoo Tech Group 2006a]. Many keywords defined in the SQL:2003 standard are used in the particular EPLs [Türker 2004, P. 181ff]. The semantic difference in the usage of SQL as a query language on a given set of data and an EPL as a query language on flowing event streams implies different usage of the particular keywords [Yahoo Tech Group 2006e]. In order to use aggregate functions like SUM or AVG or grouping functions like GROUP BY, the individual events must be collected by window techniques to get a set of data. Additionally one of the core functionalities of CEP, the recognition of event scenarios or patterns, is not covered by the SQL standard. These functionalities are managed by proprietary expressions of the respective vendors. The resulting queries or statements are implemented in a different syntax than the defined in the SQL standard. The commonness of the particular EPL query to a SQL query is usually only the SELECT FROM WHERE clause. Thus these EPLs are classified as “pseudo” SQL languages [Ammon 2006a].

8.2 Special rule languages or script languages

Another declarative language approach origins from the area of rules engines. Some vendors use similar syntaxes as business rules management systems, whereby they extend the common IF THEN blocks which describe causal dependencies, with additional language constructs in order to describe temporal relationships of events. So e.g. Apama extends the IF THEN block with the FOLLOWED-BY clause to describe the temporal order of events and the ALL
WITHIN clause to define time windows [CEP-Interest 2007], [Yahoo Tech Group 2006d].

IBM’s solution Active Middleware Technology (AMiT) uses a rule language which allows exact description of situations. The life span of the situation can be described exactly with the initiator and terminator. Additionally events can be classified as “relevant” or be “rejected”. The relevant events can be defined exactly by their attributes like e.g. second operand = event: “E3” threshold: “X > 7” or third operand = event: “E4” threshold: “X > 7”. Correlation conditions can be described like condition = “E3.X = E4.X”. So, situations can be described like the following:

```
operator = "sequence"
detection mode = "differed"
lifespan = "trading_day"
first operand = event: "stock-quote" as: "first-quote"
   threshold: "change > 0"
   quantifier: "each"
   consumption condition: "false"
second operand = event: "stock-quote" as: "second-quote"
   threshold: "change > 0"
   quantifier: "last"
   consumption condition: "true"
```

The Rules Manager, which is included in the Oracle 10g database management system, implements ECA rules with the IF THEN rule language terminology.
Action procedures can be implemented by SQL procedures. These rules and corresponding actions are stored in database tables. Figure 104 shows the definition of events, rules and actions [Oracle 2006]. Press releases of Oracle also introduced CEP capabilities of the Oracle Fusion middleware platform, but a nearer look on the EPL of this platform could not be done within this thesis [Oracle 2007].

![Event-Condition-Action Rule](image)

**Figure 104: Oracle Rule Manager architecture and code example [Oracle 2006]**

### 8.3 Java

The implementation in BusinessEvents of Tibco can be done via Java. Therefore Tibco provides the TibcoDesigner, which allows a design of an event driven UML model. The UML-based “state model” describes how applications and services interact as part of activities and processes. The creation, destruction or state change of these objects is done by events, which match to the rule condition implemented in Java. A creation, destruction or state change of an object is thereby a part of an action and represents a new event and may influence other objects. The implementation of these rules is done in Java. Describing an event driven UML model and writing rules in Java which can filter, correlate and aggregate events by applying constraints and threshold boundaries is the main part
of implementing CEP functionalities. Thus the Tibco EPL is classified as Java based [TIBCO 2006c].

\[
\begin{array}{|l|l|}
\hline
\text{EventProcessingLanguage} & \text{creditApplicationEvent} \\
\hline
\end{array}
\]

\begin{verbatim}
creditApplicationEvent.Method = "getProfile";
creditApplicationEvent.Situation = "StartSituation"
creditApplicationEvent.Component = "https://workflow.inventor5partLocal"

\end{verbatim}

**Figure 105:** Tibco rule for the creation of a new CreditApplication object

### 8.4 XML

Some platforms like Aleri, Coral8 or StreamBase store their applications in XML files, for example Aleri applications are stored in Aleri XML [Aleri 2006b]. The modules developed in Coral8 are stored in .ccl files, which are structured in XML. Thereby the CCL statements are embedded into Query tags [Coral8 2006b]. The applications built with StreamBase, are stored in .ssql and .sbapp files with a XML structure [StreamBase 2007a]. These files can also be manipulated with other applications [Aleri 2006b].

### 8.5 Graphical User Interface

Several application areas of CEP like BAM, exchange surveillance or algorithmic trading affect users without any programming knowledge, e.g. business analysts, sales [CEP-Interest 2007]. Vendors like AptSoft, StreamBase, Apama or Aleri try to bridge this gap by providing a GUI [Yahoo Tech Groups 2006d]. Therewith CEP applications can be built for example by dragging and dropping graphical components into a diagram and connecting them [StreamBase 2007b], [Aleri
The event driven object models of Tibco can also be built within graphical environments [Tibco 2006c]. AptSoft delivers graphical components, which can be reused in other components in order to build more complex queries [AptSoft 2006b], [AptSoft 2006c]. These graphical components can be configured or parameterized in order to perform a desired function like for example a filter function based on the configured condition. These parameters are subject of a proprietary syntax, so for example a filter condition in StreamBase looks like

```
TradeInformation.Volume > 300000
```

and in AptSoft a filter condition is implemented by applying conditions with text boxes and dropdown boxes as shown in the Figure below.

![Figure 106: AptSoft’s event processing GUI [AptSoft 2006c]](image)

The variety of different graphical components and their parameterization depends on the particular platform of a vendor [AptSoft 2006b], [AptSoft 2006c], [StreamBase 2007b], [Yahoo Tech Groups 2006c]. Except of AptSoft, generally these platform vendors provide an underlying or separate computing language which can be generated from the graphical designed applications. These can also be combined with graphical components [StreamBase 2007b], [Yahoo Tech Groups 2006c].

### 8.6 Summary and evaluation of the results

The opinions about the different EPLs on the present CEP market vary very much within in the CEP community, whereby the representatives of the individual vendors advocate their own ideologies. So the endeavors of StreamBase for SQL as the base for a standardization of EPLs caused many discrepancies in the CEP community. It was said that the vendors of CEP platforms chose their language for good reasons [Howard 2006]. Vendors striving for other goals disagreed that SQL is a good approach to process events at all. In their opinion the data set oriented processing of SQL does not fit to the event processing paradigm [Yahoo
Tech Groups 2006e]. They also claim that applications built in SQL based approaches are difficult to understand and to maintain. The development and maintenance of SQL based CEP applications can just be done by developers, not by business persons. In their opinion authoring tools with graphical components do not change this fact [Yahoo Tech Groups 2006f]. In the author’s opinion “pseudo” SQL approaches do not address the qualification of clerks, managers or other business related persons, whether the other approaches do not address these qualifications, too. Usually business related persons are not educated in programming or building applications. GUIs or almost natural languages do not change the fact, that an understanding of the CEP paradigm and programming and design concepts is necessary to build good CEP applications. Study courses like business IT were created to bridge these gaps of business and IT related persons. Additionally these vendors see problems in handling hierarchical event data. SQL based approaches usually use tuples as data processing units, which is very restricting. So simple events like stock tick events are easy to represent with a tuple. They doubt that more complex events with hierarchical structures like an "Order Submitted" event which contains multiple order items are manageable in tuples; they believe that events usually contain business objects as context information [Yahoo Tech Groups 2006f]. In the author’s view this problem is manageable with considerations which are related to normalizations in data bases (see para. 7.1.2, 7.2.2). So events like the CBE which have a hierarchical structure can be handled with stream splitting, whereby the attribute, identifying an event, is the “foreign key” for the stream containing the related context information (see para. 7.1.2, 7.2.2). To attain more clarity in the topic of how a language approach can solve CEP problems Etzion offered two possible CEP related problems [Yahoo Tech Groups 2006g]. Vendors providing SQL based approaches like Hagman of Coral8 and Kulleen of StreamBase offered solutions to these problems [Yahoo Tech Groups 2006h], [Yahoo Tech Groups 2006i]. Cameron of AptSoft also offered a solution, which is designed graphically [AptSoft 2006c]. Regrettably these endeavours came to no results, because not enough different solutions were posted in order to evaluate differences, weaknesses and opportunities [Yahoo Tech Groups 2006j]. It can be said recapitulating about the
recent statements that there is still no sufficient progress in the evaluation about the respective differences in the usability and behaviours of an EPL. Works like [Coral8 2007e], [Bry 2006] or this thesis contribute and lead in the direction of a more exact rateability of EPLs by describing their requirements.
9 Summary

The market for event processing is growing and, at some point, it will explode

Philip Howard, Research Director Bloor Research

As demonstrated in the use case, the possibilities offered by the implementation of event processing concepts like CEP-based BAM adapted from the correlated low-level events without semantics, help companies to gain a better IT-insight. Based in what the enterprise wants to accomplish and if it acts in fast moving sectors like algorithmic trading, this could mean to redesign the whole enterprise infrastructure in terms of EDA. Both ways help the company in accomplishing the concept of becoming a predictive, agile enterprise providing a high level of IT-insight, independent of the size. Like described in para. 3.1.5, this goal enables the enterprise not only to be flexible enough to react on challenges and chances, but to start to predict those situations in advance and therefore become proactive rather than reactive. By the real-time correlation and monitoring of as well business as technical events of the same event cloud, the prevailing gap between the business and the IT departments is not that distinct anymore, since both parties obtain their needed information based on one and the same events.

9.1 Principle design of future CEP/BAM/BPM platforms

Based on these future concepts, new CEP/BAM/BPM platforms could be designed or adopted as seen on Figure 107. This Figure shows an example how the platform of the NextGeneration easyCredit instant-online system could look like.
As seen above, this platform contains a workflow modeler who’s main purpose is the design of executable business processes. The event modeler is responsible for an automatically event generation within these processes. He also detects and designs relevant patterns of events for every part of the enterprise.

9.2 Conclusions of the event generation concepts

Although events can be generated within every part of the IT-infrastructure of an enterprise automatically, it is often very difficult to create useable and relevant (low-level) events and to do this without changing the already deployed, implemented and tested legacy application, SOA-services or other components. Another, more easier way might constitute the generation of events via the BPEL-driven business process model, as described in para. 3.2. Many CEP platforms already provide the necessary adaptors to receive these events the moment steps of the business process are executed (see chap. 7ff). Fundamental in order to apply this concept is although the modeling and execution of the business process using a business process execution language. For Java EE enterprise environments
of today’s companies such as TeamBank the AOP based interceptor concept may be utilized for generating service level-events and the servlet filter concept for user interface events. This realization might represent an acceptable non-intrusive way in matters of flexibility, because interceptors can be applied to every deployed component type, business process component or method.

9.3 Summary of the evaluated platforms

For realizing CEP-based BAM, an enterprise needs exact exigencies about the processes it wants to monitor. By understanding this concept, platforms like the following can help companies to realize this goal of reaching a new level of IT-insight. For reaching this goal, various ways of implementations are possible, depending on the platforms. Therefore it has to be appointed exactly what demands have to be made at the platform. Based on that, a proper platform can be chosen. The prerequisites on a platform based on the use case are absorption of JMS events, capabilities to handle XML and the correlation of the events. The aggregated results of the correlated events shall be displayed in a dashboard. The necessary functionalities as well as a sufficient stage of maturation of the language constructs and scopes of the underlying EPLs of the tested platforms suffice the requirements. Exclusively Systar provided a dashboard solution within its platform. The other CEP platform vendors provide APIs and output adaptors, which enable to connect to a dashboard of the customer’s choice (best-of-breed approach). TIBCO for instance provides the dashboard solution RTView as a non-browser based application (desktop client) or the browser-based dashboard application BusinessFactor resp. Spotfire in its product portfolio. However, all providers offer the option to attach dashboards from other vendors like for example the implemented Java dashboard.
Acknowledgement and devotion

Here the authors would like to thank every involved parties for the exemplary support.

In particular the authors want to thank the TeamBank Company for providing a version of their core business process application, the easyCredit-Online application which was the base of the whole use case. Explicitly the authors want to thank Dr. Torsten Greiner from TeamBank and Mr. Rolf Kintscher from Senacor Company. Without their help and know how this use case could not have been realized.

Furthermore special thanks go to the companies TIBCO and Systar for providing their application portfolio for correlating events in real-time, BAM and other software components which helped to realize the use case with all its requirements.

Explicitly the authors want to thank Mr. Wolfgang Kelz, Mr. Andreas Gerst and Mr. Andreas Sittler from TIBCO for the support. Besides that, the authors want to thank Mr. Roland Heinen and Mr. Stefan Giegerich from Claresco Company for investing that much time in supporting the authors, even on weekends.

Farther, the authors want to give the companies StreamBase and Coral8 the props for the invitations to workshops respectively meetings and for providing developer licenses and support.

Last but not least the authors want to thank TietoEnator for the sponsoring.
List of literature

[@ctiveFRIEND 2005]
http://www.santix.de/uploads/media/Pressemeldung_April_2005_Siemens-santix-@ctiveFRIEND.pdf. downloaded 2007-03-13

[Adobe 2003]

[Aleri 2006a]

[Aleri 2006b]

[Ammon 2006a]

[Ammon 2006b]

[Ammon 2006c]

[Ammon 2006d]
[Apache 2007]

[AptSoft 2006a]

[AptSoft 2006b]

[AptSoft 2006c]

[Barzdins 2004]

[Bates 2007]

[BEA 2005]

[Böge 2004]

[Boone 2003]

[Brock 2007]
[Bry 2006]

[Burke 2003]

[Buschmann 1996]

[Carnegie 2006]

[CEP-Interest 2007]

[Monson-Haefel 2001]
Monson-Haefel, R.; Chappell, D.: Java™ Message Service. O'Reilly Media

[Chisty 2006]

[Cisco 2006]

[Claresco 2007a]
Claresco: Guide to WebRecording BusinessBridge_NG.pdf. 2007. provided by Claresco

[Claresco 2007b]

[Claresco 2007c]
[Codd 1998]

[Coral8 2006a]

[Coral8 2006b]

[Coral8 2006c]

[Coral8 2006d]

[Coral8 2006e]

[Coral8 2007a]

[Coral8 2007b]

[Coral8 2007c]

[Coral8 2007d]
[Coral8 2007e]
Coral8: CEP Design Patterns. 2007.

[Covelight 2006]

[Cover Pages 2003]

[Crimson 2003]

[CSC 1995]
CSC: CSC Catalyst Methodology. 1995. CSC Inc.

[Esper 2006a]

[Esper 2006b]

[Etzion 2004]

[Fleury 2004]

[Gamma 1995]
[Greiner 2006a]
Greiner, T.: Example of Business Process designed in ARIS. Provided by Torsten Greiner.

[Greiner 2006b]

[Grob 1999]

[Gunderson 2005]

[Gunderson 2006]

[Haase 2002]

[Handelsblatt 2006]

[Hayes-Roth 2004]

[Hayes-Roth 2006]
[Hayes-Roth 2007]
Hayes-Roth, F.: Value Information at the Right Time (VIRT): Why less volume is more value in hastily formed networks. 2007
www.nps.edu/cebrowski/Docs/VIRTforHFNs.pdf. downloaded 2007-04-08

[Hill 1994]

[Howard 2006]

[IBM 2003a]
http://www.ibm.com/developerworks/autonomic/books/fpy0mst.htm#HDRAPPA. downloaded 2006-11-12

[IBM 2003b]

[IBM 2004]

[IBM 2004b]
IBM: Service-Oriented Architecture expands the vision of Web services. 2004.

[IBM 2006]

[IBM 2007]

[IEEE 2004]
[Interca Mondiale 2005]

[IT-Wissen 2007]

[Jobst 2006]

[JBoss 2004]

[JBoss 2005a]

[JBoss 2005b]

[JBoss 2007]

[JCP 2007]

[JFreeChart 2007]

[Kalnins 2003]

[Kochar 2005]

[Lakkaraju 2005]
Lakkaraju, S., Kumar, A.: Converting a POJO to a persistent entity. 2005.

[Linux 2006]
Linux/Unix project: ODBC basics and definition by easysoft. 2006.

[Luckham 2002a]

[Luckham 2002b]
Luckham, D.: The Rapide Event Pattern Language. clause 8.5ff. 2002

[Luckham 2004a]

[Luckham 2004b]

[Luckham 2004c]
http://complexevents.com/?p=27. downloaded 2006-09-10

[Luckham 2004d]

[Luckham 2005a]
[Luckham 2005b]

[Luckham 2007]

[Maréchaux 2006]

[McCoy 2002]

[Mercury 2006]

[Michelson 2006]

[Microsoft 2005]

[Microsoft 2006a]

[Microsoft 2006b]

[Natis 2003]
[Newswire 2006]

[OASIS 2006a]

[OASIS 2006b]

[OASIS 2006c]

[OASIS 2007]

[OMG 2004]

[Oracle 2004]

[Oracle 2006]

[Oracle 2007]
[PetriNets 2004]

[Quantz 2005]

[Ranadive 2006]

[RedHat 2007]

[Rittmeyer 2007]

[Rupp 2005]

[Scheer 2000]

[SearchCRM 2007]

[Seeley 2006]

[Senacor 2007]

[Service-Architecture 2005]
services/articles/business_process_execution_language_for_Web_services_BPEL
4ws.html. downloaded 2007-03-13

[SGI 2004]
http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi/linux/bks/SGI_Admin/books/ESP_Ug/sgi_html/ch07.html.
downloaded 2006-09-14

[Silberbauer 2007]

[SLA 2005]
downloaded 2007-02-03

[Sliwa 2003]

[Smith 2006]

[SOA Magazine 2005]
http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gei1089758,00.html. downloaded 2007-03-06

[Stanford 2002]
http://pavg.stanford.edu/ID/. downloaded 2006-02-10

[StreamBase 2007a]
downloaded 2007-05-04

[StreamBase 2007b]

[StreamBase 2007c]

[StreamBase 2007d]

[StreamBase 2007e]

[Sun 2000]

[Sun 2001a]

[Sun 2001b]

[Sun 2002a]
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Transaction3.html. downloaded 2007-02-02

[Sun 2002b]
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Transaction4.html. downloaded 2007-02-02

[Sun 2002c]

[Sun 2002d]

[Sun 2002e]

[Sun 2004]

[Sun 2006a]

[Sun 2006b]

[Sun 2006c]
SUN Microsystems.: EJB 3.0 specification API. 2006.

[Sun 2006d]
SUN Microsystems.: The Java EE 5 Tutorial. 2006.

[Sun 2006e]
http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp. downloaded 2007-02-02

[Sun 2006f]

[Sun 2006g]
http://java.sun.com/docs/books/tutorial/java/IandI/override.html. downloaded 2007-03-02

[Systar 2007a]

[Systar 2007b]
Systar: BusinessBridge Administration Guide. 2007. Provided by Systar

[Talkback 2006]

[TIBCO 2004a]

[TIBCO 2004b]

[TIBCO 2005]

[TIBCO 2006a]

[TIBCO 2006b]

[TIBCO 2006c]

[TIBCO 2006d]
[TIBCO 2006e]

[TIBCO 2006f]

[TIBCO 2006g]
*TIBCO*: TIBCO Rendezvous. 2006.

[TIBCO 2006h]
*TIBCO*: TIBCO EMS. 2006.

[TIBCO 2006i]

[TIBCO 2006j]
*TIBCO*: TIBCO RTView. 2006.

[Türker 2004]

[Vinayak 2006]

[Virtanen 2003]

[W3C 1997]
http://www.w3.org/TR/NOTE-datetime. downloaded 2007-05-04

[W3C 1999]
http://www.w3.org/TR/xpath. downloaded 2007-04-22

[W3C 2001]

[W3C 2004]
World-Wide-Web Consortium: Web Services Architecture W3C Working Group
downloaded 21.09.2006

[W3C 2006]
World-Wide-Web Consortium:: Extensible Markup Language (XML) 1.0 (Fourth
2007-02-02

[Wikipedia 2007a]

[Wikipedia 2007b]

[Wikipedia 2007c]

[Wikipedia 2007d]

[Wikipedia 2007e]

[Wikipedia 2007f]
Wikipedia free encyclopedia: C-Level position. 2007.

[Wily 2007]
Wily: CA Wily SOA Manager — Monitor and Manage Web Services
Manager.html. downloaded 2007-04-20

[Wolff 2004]
http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212769,00.html
. downloaded 2006-12-14

[Yahoo Tech Groups 2006a]
http://tech.groups.yahoo.com/group/CEP-Interest/message/453. downloaded 2007-05-14

[Yahoo Tech Groups 2006b]
http://tech.groups.yahoo.com/group/CEP-Interest/message/766. downloaded 2007-05-14

[Yahoo Tech Groups 2006c]
Yahoo Tech Groups: CEP-Interest · Event Processing Technologies. 2006
http://tech.groups.yahoo.com/group/CEP-Interest/message/990. downloaded 2007-05-14

[Yahoo Tech Groups 2006d]
Yahoo Tech Groups: CEP-Interest · Event Processing Technologies. 2006
http://tech.groups.yahoo.com/group/CEP-Interest/message/816. downloaded 2007-05-14

[Yahoo Tech Groups 2006e]
Yahoo Tech Groups: CEP-Interest · Event Processing Technologies. 2006
http://tech.groups.yahoo.com/group/CEP-Interest/message/674. downloaded 2007-05-14

[Yahoo Tech Groups 2006f]
Yahoo Tech Groups: CEP-Interest · Event Processing Technologies. 2006
http://tech.groups.yahoo.com/group/CEP-Interest/message/682. downloaded 2007-05-14

[Yahoo Tech Groups 2006g]
Yahoo Tech Groups: CEP-Interest · Event Processing Technologies. 2006
http://tech.groups.yahoo.com/group/CEP-Interest/message/678. downloaded 2007-05-14

[Yahoo Tech Groups 2006h]
Yahoo Tech Groups: CEP-Interest · Event Processing Technologies. 2006
http://tech.groups.yahoo.com/group/CEP-Interest/message/717. downloaded 2007-05-14

[Yahoo Tech Groups 2006i]
http://tech.groups.yahoo.com/group/CEP-Interest/message/760. downloaded 2007-05-14

[Yahoo Tech Groups 2006j]
http://tech.groups.yahoo.com/group/CEP-Interest/message/847. downloaded 2007-05-14
List of figures

Figure 1: Similar to Java EE 5 application server architecture [Sun 2006d] .......... 4
Figure 2: EJB roles similar to [Boone 2003] ............................................................ 5
Figure 3: JBoss application server architecture (derived from [JBoss 2005a]) ...... 9
Figure 4: A typically Invoker Chain (derived from [Rupp 2005]) ....................... 10
Figure 5: Message delivery (similar to [Haase 2002]) ............................................ 11
Figure 6: Example of point-to-point messaging [Haase 2002] ............................... 12
Figure 7: Publish and subscribe messaging [h hase to Haase 2002] ....................... 13
Figure 8: Example of an interceptor chain (similar to [Rupp 2005]) ................. 15
Figure 9: Example of a client interceptor ......................................................... 16
Figure 10: Complex events as an abstraction of at least to other events
            [Luckham 2005a] .................................................................................. 19
Figure 11: Complex event pattern matching leads to generation of
           complex event instance. (similar to [Luckham 2005a]) ........................... 21
Figure 12: Possible event cloud of an online banking website [Luckham 2005a] 23
Figure 13: Connection of business and IT via the event cloud
            (similar to [Jobst 2006]) ....................................................................... 24
Figure 14: The event tornado provides additional events from business process
           components and SOA services for a better IT-insight [Ammon 2006a] ...... 27
Figure 15: Example of business process modeling using ARIS at the example
           of an Order Request [Barzdins 2004] ...................................................... 30
Figure 16: Business process example of credit allocation within a SOA
           architectural concept [Ammon 2006d] .................................................... 31
Figure 17: ARIS model of TeamBank’s „easyCredit-Internet“ business
           process [Greiner 2006a] ......................................................................... 32
Figure 18: Business Process Monitoring using the example of
            "easyCredit-Internet" EPC [Greiner 2006b] .............................................. 35
Figure 19: Example of a real-time enterprise, using BAM [TIBCO 2004b] ........ 37
Figure 20: A simplified architecture for VIRT [Hayes-Roth 2007] .................... 39
Figure 21: Different information requirements need different BAM
           dashboards for visualization [Clarseco 2007c] ....................................... 40
Figure 22: Integration with traditional methods (like e.g. peer-to-peer connections,…) [Ammon 2006d] ................................................................. 41
Figure 23: Example of a multi-tier application-server based integration [Ammon 2006d] ........................................................................................................ 42
Figure 24: Example of JSPs of the easyCredit application taken from www.easycredit.de .................................................................................... 44
Figure 25: Use case diagram of the monitoring generated loss by cancelled credit applications of the next generation easyCredit-Internet ............... 45
Figure 26: Pipeline model of the TeamBank Company [Ammon 2006b] ........ 47
Figure 27: A typical customer process of a retail bank [Ammon 2006d] ....... 48
Figure 28: The SOA challenge at the design example vertical and horizontal coupling of services of a customer credit process [Ammon 2006d] ......... 49
Figure 29: Event infrastructure relationships and terminology [IBM 2003b] .... 54
Figure 30: Exemplary business alert .............................................................................................................................. 66
Figure 31: Exemplary technical alert ............................................................................................................................. 71
Figure 32: Architecture for the BAM implementation ...................................................................................................... 87
Figure 33: The three-layer architecture of TeamBank’s easyCredit application [Ammon2006a] .................................................................................................................................. 91
Figure 34: Example of the server architecture of the TeamBank Company........ 92
Figure 35: Integration of the session bean interceptor into the JBoss configuration ........................................................................................................ 92
Figure 36: Applying the servlet filter to the Web application archive of easyCredit............................................................................................................ 93
Figure 37: Creation of the "EventCloud" JMS topic ................................................. 93
Figure 38: Design of the applied ant deployment descriptor ............................... 94
Figure 39: Basic structure of the Ejb2Interceptor .............................................. 95
Figure 40: Cutout of extractDataFromInvocation() method of the Ejb2Interceptor class ........................................................................................................ 96
Figure 41: The Ejb2Interceptor method createEvent() fills a HashMap representing the event ................................................................. 97
Figure 42: Possible servlet filter chain with applied mapping rules see [Rittmeyer 2007] ................................................................................................. 99
Figure 43: The doFilter() method of the servlet filter ............................................. 100
Figure 44: The fillHashMap() method fills HashMap object "event"
    with the necessary information, used to complete the body of the CBE .... 102
Figure 45: createMessage() method of the JMSPublisher class for
    constructing the CBE body ............................................................................. 103
Figure 46: the sendJMS() method passes the generated CBE-conform
    XML string to a JBoss topic ................................................................................ 104
Figure 47: Logfile class’s method write2Log() ....................................................... 104
Figure 48: Coral8 platform [Coral8 2006a] ................................................................. 108
Figure 49: Coral8 data flow ...................................................................................... 110
Figure 50: The schema for the ProcessEvent stream ............................................... 112
Figure 51: The schema for the ExtendedDataElements stream ................................ 112
Figure 52: Parameters of the AmountMonitor ........................................................... 112
Figure 53: Code example for the canceled application pattern ................................... 113
Figure 54: Schema of canceled application ............................................................... 113
Figure 55: Detection and enrichment of canceled applications ................................. 114
Figure 56: Aggregation of canceled applications ...................................................... 115
Figure 57: The determined BAM metrics .................................................................. 115
Figure 58: Drill-down: Canceled applications per page .............................................. 115
Figure 59: Parameters of the ComponentMonitor module ........................................ 116
Figure 60: CCL implementation of the yellow method pattern .................................. 117
Figure 61: Metrics for the BAM-view ......................................................................... 117
Figure 62: Component monitor .................................................................................. 118
Figure 63: StreamBase Studio [StreamBase 2007b] .................................................. 121
Figure 64: Application diagram ............................................................................... 123
Figure 65: Java Operator ......................................................................................... 124
Figure 66: Parameters of XMLNormalizer for the extraction into
    ProcessEvent ........................................................................................................ 125
Figure 67: Parameters of XMLNormalizer for the extraction into
    ExtendedDataElements ....................................................................................... 126
Figure 68: Application diagram for the extraction of the CBE .................................. 126
Figure 69: Parameters for the AmountMonitor ....................................................... 127
Figure 70: Schema of the Query Table Application ........................................... 128
Figure 71: Filter settings of the operator “Filter_Manipulation” ....................... 128
Figure 72: Filter settings of the operator “Filter_creditAmount” ....................... 129
Figure 73: Filter settings of “Filter_cancelledApplication_OR_successfulApplication” ....................... 129
Figure 74: Application diagram of the AmountMonitor .................................... 130
Figure 75: Parameters for the ComponentMonitor ........................................... 130
Figure 76: Component Query Table .................................................................. 131
Figure 77: BAM metrics .................................................................................... 132
Figure 78: Application diagram of the ComponentMonitor ............................... 133
Figure 79: Layout of TIBCO’s BusinessEvent application. The three models work together in processing abstract events for BAM decisions [TIBCO 2006b] .................................................................................. 136
Figure 80: Message Delivery using TIBCO EMS [TIBCO 2006d] ....................... 138
Figure 81: Utilization of the Publish/Subscribe JMS messaging model using TIBCO EMS [TIBCO 2006d] .................................................................................. 138
Figure 82: Data import from the CBE log file using the File Poller palette-item ........................................................................................................ 139
Figure 83: Data import via subscription to a JMS topic using TIBCO’s JMS Topic Subscriber palette-item .............................................................................. 139
Figure 84: The parsing process parses every received xml string against the CBE schema ......................................................................................................... 139
Figure 85: TIBCO process of mapping the relevant information on the “creditApplicationEVT.event” .............................................................. 140
Figure 86: Every rule can be applied to multiple event types respectively concepts ........................................................................................................ 141
Figure 87: Screenshot of the statistic-concepts within TIBCO Designer ............ 143
Figure 88: Example of visualization options featured by TIBCO RTView ........ 144
Figure 89: Three tier architecture of Systar BusinessBridge [Claresco 2007a] .... 147
Figure 90: Data and information flow within Systar BusinessBridge architecture [Systar 2007b] .................................................................................. 149
Figure 91: BusinessBridge’s DataSource Adaptors [Claresco 2007a] ............. 151
List of figures

Figure 92: BusinessBridge BAM Expertise modules and templates for simplification of the BAM-view creation and event abstraction

[Claresco 2007a] ........................................................................................................ 152

Figure 93: Established data source connections with BusinessBridge’s DataSource Builder 4.1 .......................................................... 153

Figure 94: Data input and property extraction of received CBEs within the Bridget Builder .................................................................................................................. 154

Figure 95: Snippet of the pattern rule triggering on arrival of a new CBE event in Business Builder with focus on the entity class declaration. 155

Figure 96: Part of an event pattern designed with BusinessBridge’s Bridget Builder .................................................................................................................. 156

Figure 97: Visualization of the easyCredit application frontend using the correlated events of BusinessBridge .................................................................................. 157

Figure 98: Business view on the easyCredit application .............................................. 161

Figure 99: Drill-down view on the easyCredit application ............................................. 162

Figure 100: Component Monitor for a session bean ...................................................... 163

Figure 101: Drill-down view on the session bean methods ............................................. 163

Figure 102: Coral8 code example ............................................................................. 165

Figure 103: Apamas Rule language [CEP-Interest 2007] ............................................. 167

Figure 104: Oracle Rule Manager architecture and code example

[Oracle 2006] ........................................................................................................ 168

Figure 105: Tibco rule for the creation of a new CreditApplication object.............. 169

Figure 106: AptSoft’s event processing GUI [AptSoft 2006c] .................................... 170

Figure 107: A reference model of the adoption of a CEP/BAM/BPM platform

[Ammon 2006a] ........................................................................................................ 174
List of tables

Table 1: Legend for Figure 29 [IBM 2003b] ................................................................. 54
Table 2: Metrics for the easyCredit process ................................................................. 62
Table 3: Average duration metric for the easyCredit process ......................................... 62
Table 4: Application step metric for the easyCredit process ......................................... 63
Table 5: Session bean metric for the easyCredit process ............................................. 63
Table 6: Application server metric for the easyCredit process .................................... 64
Table 7: Parameters for a component monitor ............................................................. 68
Table 8: Request metric .............................................................................................. 69
Table 9: Completed request metric ............................................................................. 69
Table 10: Yellow request metric .................................................................................. 69
Table 11: Red request metric ...................................................................................... 70
Table 12: Processing duration metric .......................................................................... 70
Table 13: Threshold parameters .................................................................................. 70
Table 14: Event attributes ............................................................................................ 73
Table 15: CBE elements .............................................................................................. 74
Table 16: ComponentIdentification elements ............................................................... 75
Table 17: Situation elements ...................................................................................... 75
Table 18: Situation type: StartSituation ...................................................................... 75
Table 19: Situation type: StopSituation ...................................................................... 76
Table 20: Complex type RequestSituation [IBM 2003] .............................................. 76
Table 21: ExtendedDataElement expressing the method ............................................. 77
Table 22: ExtendedDataElement expressing the additional identifier ......................... 77
Table 23: ExtendedDataElement expressing the credit amount .................................. 77
Table 24: ExtendedDataElement expressing the credit decision .................................. 78
Table 25: ExtendedDataElement expressing the status of employment ..................... 78
Table 26: CBE with selected elements ....................................................................... 78
Table 27: Exemplary event for a canceled credit application ....................................... 84
Table 28: Exemplary component report ...................................................................... 87
Table 29: Design pattern structure elements of Gamma and Buschmann

[Silberbauer 2007] ................................................................................................. 88
Table 30: Elements of the Design Pattern description of Buschmann

[Silberbauer 2007]........................................................................................................... 89

Table 31: JMS adaptor parameters [Coral8 2006d]......................................................... 109
## List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARIS</td>
<td>Architektur integrierter Informationssysteme</td>
</tr>
<tr>
<td>B2B</td>
<td>Business-to-Business</td>
</tr>
<tr>
<td>BAM</td>
<td>Business Activity Monitoring</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>BPM</td>
<td>Business Process Management</td>
</tr>
<tr>
<td>BPMI</td>
<td>Business Process Management Initiative</td>
</tr>
<tr>
<td>CBE</td>
<td>Common Base Event</td>
</tr>
<tr>
<td>CEP</td>
<td>Complex Event Processing</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>ECAA</td>
<td>Event, Condition, then Action and else Action</td>
</tr>
<tr>
<td>eEPC</td>
<td>extended Event Processing Chain</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Bean</td>
</tr>
<tr>
<td>EP</td>
<td>Enterprise Plattform</td>
</tr>
<tr>
<td>EPC</td>
<td>Event Processing Chain</td>
</tr>
<tr>
<td>EPP</td>
<td>Enterprise Portal Platform</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>J2SE</td>
<td>Java 2 Standard Edition</td>
</tr>
<tr>
<td>JDK</td>
<td>J2SE Development Kit</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>JSR</td>
<td>Java Specification Request</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Maschine</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WSDM</td>
<td>Web Service Distributed Management</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Metadata Interchange</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
</tr>
</tbody>
</table>
Appendix A – Common Base Event Schema [IBM 2003a]

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:cbe="http://www.ibm.com/AC/commonbaseevent1_0_1"
    targetNamespace="http://www.ibm.com/AC/commonbaseevent1_0_1"
    version="1.0.1"
    elementFormDefault="qualified">
    <xsd:complexType name="CommonBaseEventType">
        <xsd:sequence>
            <xsd:element name="contextDataElements" type="cbe:ContextDataElementType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="extendedDataElements" type="cbe:ExtendedDataElementType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="associatedEvents" type="cbe:AssociatedEventType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="reporterComponentId" type="cbe:ComponentIdentificationType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="sourceComponentId" type="cbe:ComponentIdentificationType" minOccurs="1" maxOccurs="1" />
            <xsd:element name="msgDataElement" type="cbe:MsgDataElementType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="situation" type="cbe:Situation" minOccurs="1" maxOccurs="1" />
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="skip" />
        </xsd:sequence>
        <xsd:attribute name="version" use="optional">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="16"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="globalInstanceId" use="optional">
            <xsd:simpleType>
                <xsd:restriction base="xsd:ID">
                    <xsd:minLength value="32"/>
                    <xsd:maxLength value="64"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:schema>
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="extensionName" use="optional">
<xsd:simpleType>
<xsd:restriction base="xsd:Name">
<xsd:maxLength value="64" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:attribute>
<xsd:attribute name="localInstanceId" use="optional">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="128" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:attribute>
<xsd:attribute name="creationTime" type="xsd:dateTime" use="required" />
<xsd:attribute name="severity" use="optional">
<xsd:simpleType>
<xsd:restriction base="xsd:short">
<xsd:minInclusive value="0" />
<xsd:maxInclusive value="70" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:attribute>
<xsd:attribute name="msg" use="optional">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="1024" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:attribute>
<xsd:attribute name="priority" use="optional">
<xsd:simpleType>
<xsd:restriction base="xsd:short">
<xsd:minInclusive value="0" />
<xsd:maxInclusive value="100" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:attribute>
<xsd:attribute name="sequenceNumber" use="optional">
<xsd:simpleType>
<xsd:restriction base="xsd:long">
<xsd:minInclusive value="0" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:attribute>
<!--
This attribute has been replaced for version 1.0.1 of the Common Base Event Schema. For version 1.0.1 the attribute categoryName should be filled in with a value that was assigned to the attribute situationType

```xml
<xsd:attribute name="situationType" use="optional">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:maxLength value="512" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
```

```xml
<xsd:attribute name="repeatCount" use="optional">
    <xsd:simpleType>
        <xsd:restriction base="xsd:short">
            <xsd:minInclusive value="0" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
```

```xml
<xsd:attribute name="elapsedTime" use="optional">
    <xsd:simpleType>
        <xsd:restriction base="xsd:long">
            <xsd:minInclusive value="0" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
```

```
<xsd:complexType>
    <xsd:element name="CBE" type="cbe:CommonBaseEventType" />
</xsd:complexType>
```

```
<xsd:complexType name="ComponentIdentificationType">
    <xsd:attribute name="component" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:maxLength value="256" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="subComponent" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:maxLength value="512" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="componentIdType" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:maxLength value="32" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>
```
<xsd:attribute use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="512" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
<xsd:complexType name="MsgDataElementtype">
  <xsd:sequence>
    <xsd:element name="msgCatalogTokens" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="value" use="required">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:maxLength value="256" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:complexType>
    </xsd:element>
    <xsd:group ref="cbe:msgIdgroup" minOccurs="0" maxOccurs="1" />
    <xsd:group ref="cbe:msgCatalogGroup" minOccurs="0" maxOccurs="1" />
  </xsd:sequence>
  <xsd:attribute name="msglocale" use="optional">
    <xsd:simpleType>
      <xsd:restriction base="xsd:language">
        <xsd:maxLength value="11" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:element name="msgCatalogId" minOccurs="1" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="128" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
<xsd:element name="msgCatalogType"
  minOccurs="1" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="32" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="msgCatalog" minOccurs="1"
  maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="128" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:group name="msgIdGroup">
  <xsd:sequence>
    <xsd:element name="msgId" minOccurs="1"
      maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:maxLength value="256" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="msgIdType" minOccurs="1"
      maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:Name">
          <xsd:maxLength value="32" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="AssociatedEventType">
  <xsd:choice minOccurs="1" maxOccurs="1">
    <xsd:element name="associationEngineInfo">
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:restriction base="cbe:AssociationEngineType">
            <xsd:choice minOccurs="1" />
          </xsd:restriction>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="associationEngine" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:maxLength value="32" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
<xsd:restriction base="xsd:NMTOKEN">
  <xsd:minLength value="32"/>
  <xsd:maxLength value="64"/>
</xsd:restriction>
</xsd:element>
</xsd:choice>
<!-- This association would contain a serialized version of the GloballyUniqueIId for all the resolvedAssociatedEvent objects -->
<xsd:attribute name="resolvedEvents" type="xsd:NMTOKENS" use="required"/>
</xsd:complexType>
<xsd:complexType name="AssociationEngineType">
  <!-- This id would contain a serialized version of the GloballyUniqueIId for all the resolvedAssociatedEvent objects -->
  <xsd:attribute name="id" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:minLength value="32"/>
        <xsd:maxLength value="64"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="type" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:Name">
        <xsd:maxLength value="64"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="name" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:Name">
        <xsd:maxLength value="64"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:element name="associationEngine" type="cbe:AssociationEngineType"/>
<xsd:complexType name="ExtendedDataElementType">
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="1">
      <!-- Additional choices here -->
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="values" minOccurs="1" maxOccurs="unbounded">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="1024"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
<xsd:element name="hexValue" type="xsd:hexBinary" minOccurs="1" maxOccurs="1" />
<xsd:element name="children" type="cbe:ExtendedDataElementType" minOccurs="0" maxOccurs="unbounded" />
</xsd:choice>
<xsd:attribute name="name" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="64"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="type" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="64"></xsd:maxLength>
      <!-- Added type noValue when only a children is specified. -->
      <!-- With a new type noValue, so the value is ignored and only the children is considered. V1.1 Approved -->
      <xsd:enumeration value="noValue"></xsd:enumeration>
      <xsd:enumeration value="byte"></xsd:enumeration>
      <xsd:enumeration value="short"></xsd:enumeration>
      <xsd:enumeration value="int"></xsd:enumeration>
      <xsd:enumeration value="long"></xsd:enumeration>
      <xsd:enumeration value="float"></xsd:enumeration>
      <xsd:enumeration value="double"></xsd:enumeration>
      <xsd:enumeration value="string"></xsd:enumeration>
      <xsd:enumeration value="dateTime"></xsd:enumeration>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
<xsd:enumeration value="boolean"></xsd:enumeration>
<xsd:enumeration value="byteArray"></xsd:enumeration>
<xsd:enumeration value="shortArray"></xsd:enumeration>
<xsd:enumeration value="intArray"></xsd:enumeration>
<xsd:enumeration value="longArray"></xsd:enumeration>
<xsd:enumeration value="floatArray"></xsd:enumeration>
<xsd:enumeration value="doubleArray"></xsd:enumeration>
<xsd:enumeration value="stringArray"></xsd:enumeration>
<xsd:enumeration value="dateTimeArray"></xsd:enumeration>
<xsd:enumeration value="booleanArray"></xsd:enumeration>
<xsd:enumeration value="hexBinary"></xsd:enumeration>
<xsd:minLength value="1"></xsd:minLength>
</xsd:restriction>
</xsd:simpleType>
</xsd:complexType>

<xsd:complexType name="ContextDataElementType">
  <xsd:choice minOccurs="1" maxOccurs="1">
    <xsd:element name="contextValue" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:maxLength value="1024"></xsd:maxLength>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="contextId" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:minLength value="32"></xsd:minLength>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
<xsd:attribute name="name" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="64"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>

<xsd:attribute name="type" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="64"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>

<xsd:complexType name="CommonBaseEventsType">
  <xsd:sequence>
    <xsd:element ref="cbe:associationEngine" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="cbe:CBE" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="CBEs" type="cbe:CommonBaseEventsType" />

<xsd:complexType name="Situation">
  <xsd:sequence>
    <xsd:element name="situationType" type="cbe:SituationType" minOccurs="1" maxOccurs="1" />
  </xsd:sequence>
  <xsd:attribute name="categoryName" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="StartSituation" />
        <xsd:enumeration value="StopSituation" />
        <xsd:enumeration value="FeatureSituation" />
        <xsd:enumeration value="DependencySituation" />
        <xsd:enumeration value="RequestSituation" />
        <xsd:enumeration value="ConFigureSituation" />
        <xsd:enumeration value="ConnectSituation" />
        <xsd:enumeration value="CreateSituation" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="SituationType" abstract="true">
  <xsd:attribute name="reasoningScope" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:maxLength value="64"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="StartSituation">
  <xsd:complexContent>
    <xsd:extension base="cbe:SituationType">
      <xsd:attribute name="successDisposition" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="64"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="situationQualifier" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="64"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="StopSituation">
  <xsd:complexContent>
    <xsd:extension base="cbe:SituationType">
      <xsd:attribute name="successDisposition">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="64"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="situationQualifier">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="64"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:attribute
  name="successDisposition" use="required">
  <xsd:simpleType>
    <xsd:restriction
      base="xsd:string">
      <xsd:maxLength
        value="64"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>

<xsd:attribute
  name="situationQualifier" use="required">
  <xsd:simpleType>
    <xsd:restriction
      base="xsd:string">
      <xsd:maxLength
        value="64"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>

<xsd:extension base="cbe:Situati
type">
  <xsd:attribute
    name="successDisposition" use="required">
    <xsd:simpleType>
      <xsd:restriction
        base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute
    name="situationDisposition" use="required">
    <xsd:simpleType>
      <xsd:restriction
        base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:extension>
</xsd:complexType>
<xsd:complexType name="RequestSituation">
  <xsd:complexContent>
    <xsd:extension base="cbe:Situati
type">
      <xsd:attribute
        name="successDisposition" use="required">
        <xsd:simpleType>
          <xsd:restriction
            base="xsd:string">
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute
        name="situationDisposition" use="required">
        <xsd:simpleType>
          <xsd:restriction
            base="xsd:string">
        </xsd:restriction>
      </xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:attribute name="successDisposition" use="required">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:maxLength value="64"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>

value="64"></xsd:maxLength>
        </xsd:restriction>
    </xsd:attribute>
</xsd:complexType>
</xsd:complexType>
</xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AvailableSituation">
    <xsd:complexContent>
        <xsd:extension base="cbe:SituationType">
            <xsd:attribute name="operationDisposition" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength value="64"></xsd:maxLength>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="successDisposition" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength value="64"></xsd:maxLength>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:attribute
name="availabilityDisposition" use="required">
  <xsd:simpleType>
    <xsd:restriction
      base="xsd:string">
      <xsd:maxLength
      value="64"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>

<xsd:attribute
name="processingDisposition" use="required">
  <xsd:simpleType>
    <xsd:restriction
      base="xsd:string">
      <xsd:maxLength
      value="64"></xsd:maxLength>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>

<xsd:complexType name="ReportSituation">
  <xsd:complexContent>
    <xsd:extension base="cbe:SituationType">
      <xsd:attribute name="reportCategory"
        use="required">
        <xsd:simpleType>
          <xsd:restriction
            base="xsd:string">
            <xsd:maxLength
            value="64"></xsd:maxLength>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="CreateSituation">
  <xsd:complexContent>
    <xsd:extension base="cbe:SituationType">
      <xsd:attribute name="successDisposition" use="required">
        <xsd:simpleType>
          <xsd:restriction
            base="xsd:string">
            <xsd:maxLength
            value="64"></xsd:maxLength>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:sequence>
  <xsd:any namespace="##any" minOccurs="1" maxOccurs="1" processContents="skip" />
</xsd:sequence>
</xsd:extension>
</xsd:complexType>
</xsd:schema>
Appendix B – Reference model „loss pattern“ [Silberbauer 2007]

Example

A credit application is a pretty extensive order process. A customer has to reveal numerous personal data about his financial circumstances in order to decide whether a bank credit can be granted and under which conditions. This process will not always be completed successfully which will lead to the loss of a potential credit contract. For the operators of such a platform it would be interesting to detect these losses and figure out the causes as accurate as possible. Those reasons could be, e.g.:

- The customer considers the credit application as too long-winded.
- The customer decides spontaneously that he does not want to have a credit anymore.
- The customer does not have all required data available, e.g. about rental income, cancels the application and intents to restart the application later.
- The customer does not have enough time to complete the application and wants to restart it later.
- The customer first wants to discuss with his spouse whether the credit is really necessary.
- The system is overloaded.
- The system fails and throws an exception which forces the process to abort.
- A server- or client-side hardware problem occurs.

Of course, it is not possible to recognize each detail of an abort, especially when the reason is founded in the psyche of the customer. However it would be helpful to determine where in the order process the cancellation happened. Thus, e.g. design errors of the order process or programming errors can be located.

Context

Customers use web applications to order products. The order process encompasses several web pages. The actual order is released at the end of this process. During this process there can be various reasons to cancel.
Problem

The operator of an internet sales platform is interested in determining cancellations of the order process and to obtain further information. The following questions are in particularly relevant:

- Where did the order cancellation occur?
- How often was it canceled?
- What is the economic dimension that the losses caused?

Solution

The web application fires events to the CEP platform for all relevant user actions. These events contain the following information:

- Key identifying an order
- Location in the program where the events were fired
- Current order amount

If no further events are fired during a defined time slot (activityTimeout), a “loss” is determined.

The losses are aggregated over a particular time slot (observationPeriod), regarding number and amount. Additionally both will be grouped by their locations.

Structure

Different types of events are necessary for the solution. In the following these are ordered according to their abstraction level:
On the lowest abstraction level there is the event type \textit{UserActivity}. Corresponding events are fired at user activities during the order process. They include:

- \textit{sessionId}: key identifying the order process,
- \textit{location}: location in the program where the event was fired and
- \textit{amount}: current amount of the order.

The event type \textit{Loss} is based on \textit{UserActivity}. It defines the actual loss of an order and contains the following attributes:

- \textit{sessionId}: the key of the order process,
- \textit{location}: location of the last fired \textit{UserActivity} event and
- \textit{amount}: the current amount of the order.

A BAM view receives an aggregated prospect of the losses in two different ways:

The event type \textit{AggregatedLoss} includes for a defined observation period (\textit{observationPeriod}):  

- \textit{number}: the number of all losses and
- \textit{amount}: the total amount of orders which were lost.

Both values are additionally grouped by their locations. Thus the so-called \textit{AggregatedLossPerLocation} has this attributes:

- \textit{location}: location the losses occurred,
- \textit{number}: number of losses per location and
- \textit{amount}: total amount of the orders which were lost per location
Dynamics

Each significant user activity, particularly click events and user inputs, invokes a UserActivity event.

Once for a certain time slot (activityTimeout) no more UserActivity events have been invoked, a Loss event is fired.

The time slot, while an order process is active and a loss can be determined, starts with the first UserActivity event of an order, i.e. the first time a certain sessionId appears. The time slot ends with the UserActivity event of a predefined location, namely the so-called lastLocation. This is the last UserActivity event of an order. It has to be sent as soon as an order is completed successfully.

Each loss event causes a new AggregatedLoss event and also a new AggregatedLossPerLocation event, i.e. the aggregated values will be refreshed on every single update immediately.

Implementation

In order to implement the pattern the following steps have to be proceeded:

- **Fire an event of the type UserActivity on each essential user activity in the program.** The invocation of the events either can be realized directly in the existing code or can occur non-intrusive, e.g. by interceptors in Java or by other aspect-oriented approaches depending on the programming language.

- **Normalize the location’s value.** The type of location is string. It indicates where a UserActivity event was fired, e.g. in which website, in which method/function, in which class or on which server. The composition of the location is specific to the context. It is recommended to build the location in terms of an URI.

- **Define the parameter lastLocation.** It reflects where the order is completed successfully. lastLocation has to be unique and exists typically once; there are no different last locations.
• *Define the parameter activityTimeout.* Choose this time slot large enough, so that the next *UserActivity* event occurs before the timeout happens. Otherwise a "loss" would be propagated. So the *activityTimeout* is the maximum time interval between two *userActivity* events.

• *Define the observationPeriod.* It specifies the time slot, on which the aggregated loss events refer to. Typical values for this parameter are a day, but also an hour, depending on how often "losses" usually occur, how strong "losses" fluctuate and how fast you like to react on changes.

• *Realize the different complex event types Loss, AggregatedLoss and AggregatedLossPerLocation in the CEP platform.*

**Example Resolved**

This solution shows how the single complex event types are defined using the Coral8 syntax. They are based on *UserActivity* events. They are fired directly by the web application.

First, the Loss-Event:

```sql
INSERT INTO Loss
SELECT    Old.sessionId AS sessionId,
          Old.location AS location,
          Old.amount AS amount
FROM      UserActivity AS Old, UserActivity AS New
MATCHING  [$activityTimeout: Old, !New] ON Old.sessionId =
          New.sessionId
WHERE     Old.location != $lastLocation;
```
The MATCHING clause realizes the timeout. If no new UserActivity event of the same session occurs over the time slot activityTimeout, a Loss event will be fired. Once an event arrives with the location corresponding to the lastLocation of an order, this order is finished successfully.

The implementation of AggregatedLoss and AggregatedLossPerLocation are quite simple. Their data is relevant to the time slot that is set by the parameter observationPeriod.

```
INSERT INTO AggregatedLoss
SELECT COUNT(sessionId) AS number, SUM(amount) AS amount 
FROM Loss KEEP $observationPeriod;

INSERT INTO AggregatedLossPerLocation
SELECT location AS location, COUNT(sessionId) AS number, SUM(amount) AS amount 
FROM Loss KEEP $observationPeriod
GROUP BY location;
```

**Variants**

*Difference amounts instead of absolute amounts*

Sometimes it is not possible or inefficient determining the current total amount if it consists of several single amounts in order to fire a UserActivity event, e.g. by
Appendix B – Reference model „loss pattern“ [Silberbauer 2007] 227

Interceptors. Especially, this fact is related to UserActivity events, which do not designate a change of the amount. Hence it is recommendable to paste the difference amount into the event instead of the absolute amount. This more primitive event type is called UserActivityBasic event. The CEP-Platform derives the actual UserActivity event from these UserActivityBasic events.

The following listing shows how to create the UserActivity event out of a UserActivityBasic event according to the Coral8-Syntax:

```
INSERT INTO UserActivity

SELECT UserActivityBasic.sessionId AS sessionId,
      UserActivityBasic.location AS location
FROM UserActivity KEEP 1 ROW PER sessionId KEEP $sessionTimeout, UserActivityBasic;
```

A UserActivity event is based on a UserActivityBasic event, which contains the difference amount, and the previous UserActivity event of the order, containing the current total amount. The new total amount is determined by the sum of these two values.

In particular you should consider the following aspects while implementing:

- The first event of an order has no precedent event. Thus the current total amount is 0. In the listing above the COALESCE function is used for this case.
- And: There is no need to keep the last UserActivity event in memory forever, but only for the time slot the order process is still active. Therefore the so-called sessionTimeout is introduced. It specifies the maximum time the last UserActivity event is retained.
Different activity timeouts

Instead of only one `activityTimeout` you can define different timeouts depending on the step of the order. This considers the aspect that not each step of an order takes the same length of time. A global timeout has to be geared to the most time-consuming step of the order whereas the individual timeouts can be parameterized to the actual duration of the step of an order. Hence a “Loss” can be detected faster. This might be necessary if a reaction to a Loss should occur in real-time.

Exceptions initiate loss events directly

If an application throws an exception and leads to the cancellation of the session, a Loss will be detected after the timeout as described. The exception leads to the cancellation of the order process, the user cannot proceed, the `activityTimeout` occurs and thereupon a Loss event will be fired. Alternatively Exceptions could fire Loss events immediately. This would have two benefits:

- The Loss can be recognized faster, because it happens immediately and not after the timeout.
- The event attribute `location` can be specified more accurate. Instead of using the location of the precedent `UserActivity` event, the exception itself can be used to specify the location.

Ignore Losses with small amounts

Maybe you do not want to take into account losses of orders having only small amounts or even those which have no amount at all. In that case define a parameter `amountLimit` and consider this parameter by specifying a Loss. In Coral8 you must add the following WHERE clause to the implementation of the event type Loss:

```
WHERE    amount >= $amountLimit
```
Additional Aggregations of the Loss event

The two composed Loss events named AggregatedLoss and AggregatedLossPerLocation are not the only useful aggregations. Here is a further example:

- AggregatedLossPerAmount: Determine the Number of losses grouped by ranges of amounts.

Known Uses

The Teambank uses the Loss pattern in its easyCredit platform. This is an online platform for simple and fast granting of bank credits. The Loss pattern is quite useful to ensure high availability, to detect aborts of high number or amount quickly and to allow proceeding appropriate reactions immediately.

Consequences

The Loss pattern provides fundamental benefits:

It is possible to analyze and to correct weak points of an order process. If losses appear frequently at a specific location it leads to the assumption that the user might be deterred of entering the data. It might be that the input is too extensive or too long-winded. Consequently, the operator can remove – if possible – or at least reduce the corresponding user inputs or he can move them to a later point in the process where the user is less likely to abort.

Appropriate measurements against a concrete Loss can be triggered. Immediately after a certain Loss corresponding arrangements can be initiated, e.g. if applicable the operator can personally get in touch with the user or can apply appropriate advertising.

Bugs can be detected. Bugs in a program that lead to the cancellation of an order process are also leading inevitably to a loss. Not only exceptions can be detected,
that usually are written in log files anyway, but also e.g. infinite loops will be recognized as losses.