



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

DEPARTAMENTO DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES
E DE COMPUTADORES

Relatório Final do Projecto de Mestrado

**BAM Framework for OMS using Open Source
Technologies**

João Pereira

Dissertação para obtenção do grau de Mestre em Engenharia Informática e de
Computadores

Orientadores:

Carlos Gonçalves

Tiago Esperança

Presidente de Júri:

Manuel Barata

Arguente:

Luís Morgado

Vogais:

Carlos Gonçalves

Tiago Esperança

November 29, 2010



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

DEPARTAMENTO DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES
E DE COMPUTADORES

Relatório Final do Projecto de Mestrado
BAM Framework for OMS using Open Source
Technologies

João Pereira

Dissertação para obtenção do grau de Mestre em Engenharia Informática e de
Computadores

Orientador:

Carlos Gonçalves: _____

Co-Orientador:

Tiago Esperança: _____

Presidente de Júri:

Manuel Barata: _____

Arguente:

Luís Morgado: _____

November 29, 2010

Acknowledgments

First of all to my family, thanks for being there when i needed you and for the unconditional support throughout my academic life. As such this project is dedicated to them.

I wish to express my gratitude to the project mentors, Carlos Gonçalves and Tiago Esperança, for all the help and advices that allowed me to complete this project.

Finally to all my friends and others that crossed my academic life and contributed to its success.

Resumo

No início da década de 90, as empresas começaram a sentir a necessidade de melhorar o acesso à informação das suas actividades para auxiliar na tomada de decisões. Desta forma, no mundo da informática, emergiu o sector *Business Intelligence* (BI) composto inicialmente por *data warehousing* e ferramentas de geração de relatórios. Ao longo dos anos o conceito de BI evoluiu de acordo com as necessidades empresariais, tornando a análise das actividades e do desempenho das organizações em aspectos críticos na gestão das mesmas.

A área de BI abrange diversos sectores, sendo o de geração de relatórios e o de análise de dados aqueles que melhor preenchem os requisitos pretendidos no controlo de acesso à informação do negócio e respectivos processos. Actualmente o tempo e a informação são vantagens competitivas e por esse mesmo motivo as empresas estão cada vez mais preocupadas com o facto de o aumento do volume de informação estar a tornar-se insustentável na medida que o tempo necessário para processar a informação é cada vez maior. Por esta razão muitas empresas de software, tais como Microsoft, IBM e Oracle estão numa luta por um lugar neste mercado de BI em expansão.

Para que as empresas possam ser competitivas, a sua capacidade de previsão e resposta às necessidades de mercado em tempo real é requisito principal, em detrimento da existência apenas de uma reacção a uma necessidade que peca por tardia. Os produtos de BI têm fama de trabalharem apenas com dados históricos armazenados, o que faz com que as empresas não se possam basear nessas soluções quando o requisito de alguns negócios é de tempo quase real. A latência introduzida por um *data warehouse* é demasiada para que o desempenho seja aceitável. Desta forma, surge a tecnologia *Business Activity Monitoring* (BAM) que fornece análise de dados e alertas em tempo quase real sobre os processos do negócio, utilizando fontes de dados como *Web Services*, filas de mensagens, etc.

O conceito de BAM surgiu em Julho de 2001 pela organização Gartner, sendo uma extensão orientada a eventos da área de BI. O BAM define-se pelo acesso em tempo real aos indicadores de desempenho de negócios com o intuito de aumentar a velocidade e eficácia dos processos de negócio. As soluções BAM estão a tornar-se cada vez mais comuns e sofisticadas.

Abstract

In the early 1990s enterprises began to feel the necessity to have better access to information for decision making, through visibility at all times into activities at all levels of business. For this matter emerged the discipline Business Intelligence (BI) composed by data warehousing and reporting tools. A developer had to extract business data, transform it into structured data and load it into a data warehouse for reporting and analysis.

Through the years BI evolved according to the companies needs. Activities analysis and monitoring business performance turned to be critical aspects in the company management.

Nowadays BI covers areas like data warehousing, data integration technologies, query, reporting, and analysis tools which fulfil the requirements of better and controlled access to business information and processes. Today almost every sector of commerce is event-driven. Time and information are competitive advantage and due to that fact, managers of event-driven enterprises are becoming increasingly stressed as time necessary to process data is increasing and simultaneously the information amount becomes unbearable. For this reason many software heavyweights, such as Microsoft, IBM and Oracle, have joined the struggle for a piece of the BI expanding market.

Staying ahead of the competition requires the ability to predict and respond to trends in the market in real-time, rather than react. BI products have the reputation of working only with historical data gathered from data warehouses. To be able to catch business activities issues before or right after they occur companies can't rely only on traditional BI systems. The latency introduced by moving data into a data warehouse is too high to be able to respond in real-time. BAM technologies can provide real-time business analysis and alerts on information from sources like Web services, message queues, etc.

The Business Activity Monitoring (BAM) concept emerges in July 2001 by Gartner enterprise, and it is an event-driven extension of BI. BAM defines the concept of providing real-time access to critical business performance indicators to improve the speed and effectiveness of business operations. BAM solutions are becoming increasingly more common and sophisticated.

Contents

Acknowledgments	ii
Resumo	iv
Abstract	vi
Introduction	x
1 Business Activity Monitoring	2
1.1 The Concept	2
1.2 Why Business Activity Monitoring?	5
1.3 Monitoring	5
1.3.1 Complex Event Processing	6
1.3.2 Choosing the Messaging Protocol	8
1.3.3 Real-Time Data Storage	8
1.4 Analysing	11
1.4.1 Constraints	12
1.5 Reporting	12
1.5.1 Dashboards	13
1.5.2 Alerting	17
1.6 Doing it	18
1.7 Planning	18
1.8 State of the Art	18
1.9 Summary	19
2 Pre-Requisites and Studying Platforms	22
2.1 The Order Management System	22
2.1.1 JBoss Application Server	22
2.1.2 JBoss Enterprise Service Bus	23
2.1.3 JBoss Messaging	23
2.1.4 JBoss Business Process Management	24

2.1.5	Oracle Database Management System	24
2.1.6	Finding Connection Points	25
2.1.7	JMS Notifications	28
2.2	Architecture and Platforms	32
2.2.1	Complex Event Processing Platforms	32
2.2.2	Presentation using Dashboards	43
2.2.3	Application Business Integration	48
2.2.4	Real-Time Data Storage	50
2.3	Summary	53
3	Prototype Development	54
3.1	Development	55
3.1.1	The Architecture	55
3.1.2	The Foundations	56
3.1.3	The Event Engine	56
3.1.4	The User Console	59
3.1.5	Data Storage	66
3.2	Summary	71
	Conclusions	72
	Acronyms	74
	References	76
	List of Figures	82
	List of Tables	84
	List of Listings	85
	A Use Cases	88
	B Sequence Diagrams	94
	C Project Planning Maps	98

Introduction

This project aims to develop a completely open source oriented BAM solution, which is targeted to run over an Order Management System (OMS). The targeted OMS framework was developed by Xpand IT and is focused on a typical services company. This project was developed on an internship between Xpand IT Solutions and ISEL (Instituto Superior de Engenharia de Lisboa) under the master's degree project at ISEL.

At the beginning, BAM concept research and technological comparisons have been performed, in order to determine the solution main modules and the chosen open source tools, frameworks and technologies, for later BAM solution development. When necessary, performance tests were conducted as well. System requirements have been target of a careful analysis, characterised by the definition of use cases, robustness and sequence diagrams to ease communication, so the entropy was reduced.

This project report is divided in three main sections; the first one studies this particular discipline of business intelligence exploring the BAM concept, analysing its current adequacy to enterprise needs and to technology evolution; the second one presents some of the choices made on architectural and technological aspects, as well as, the project development and plan; the third one presents the prototype developed.

Chapter 1

Business Activity Monitoring

1.1 The Concept

The Business Activity Monitoring (BAM) concept was created by the organisation Gartner Research[1] and consists on providing an enterprise solution primarily intended to provide as possible as real-time insight into daily business activities, detecting events, filtering them and triggering business process management solutions in order to react instantly.

BAM stands for a class of applications that observe the behaviour of business activities and provide ways to monitor heterogeneous business applications, aggregated data, provide alerts and measures relevant business metrics called Key Performance Indicator (KPI). This allows a quick and effective analysis and presentation of near real-time information about activities within the organisation. BAM provides visibility into business processes and event-context correlations extremely quickly, providing accurate information about the status and results of various operations, processes, and transactions. The near real-time monitoring of operational processes is often accepted as the key benefit of BAM against conventional Business Intelligence. A basic BAM system can be represented by the diagram in Figure 1.1. This analytical insight helps enterprises to react, to make better informed business decisions and quickly address problematic areas by helping to identify operational inefficiencies, predicting potential problems and improving business operations performance. Most of the decisions and actions taken using BAM are reactive in nature, which means that the user acts after a business situation has occurred. However, using predictive BI techniques in conjunction with BAM makes it possible to detect patterns in business operations and predict business issues before they occur (e.g. fraudulent behaviour) and allow risk preventing measures to be taken (e.g. cancelling a credit card). Therefore and additionally to monitoring, BAM systems should be able to send automatic alerts that can be used to notify decision makers about changes in the business scenarios that may require an action.

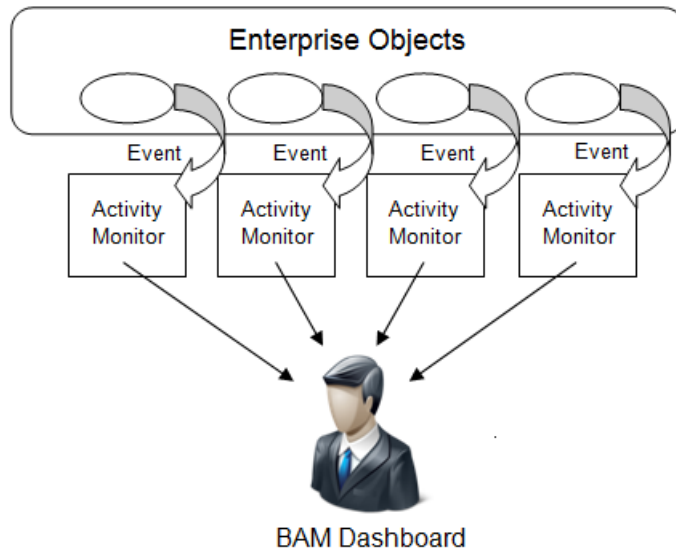


Figure 1.1: The basic BAM solution; reads events from enterprise applications but doesn't interfere with the communication between them. Activity Monitors use the events to measure the status which is displayed in a dashboard. Basic BAM uses single events to trigger rules that notify situations that might affect business.

BAM applies operational business intelligence and application integration technologies to automated processes to continually refine them, based on feedback that comes directly from knowledge of operational events. Events must be captured and processed with minimum latency and therefore, BAM is tightly integrated with its operations sources and optimised for event processing and correlating event information with historical or contextual information.

BAM exposes the visibility requirements of the end-to-end business process. The various roles within the business interact with the business process as well as the data requirements for the interactions.

BAM solutions can also integrate with online analytical processing (OLAP) and data mining tools to provide monitoring over analysed information, for example if a data mining tool reports fraudulent accounts to a BAM tool, it will probably freeze the accounts and alert interested business users. This behaviour allows BAM to make not only in-line analysis but also off-line ones. As shown in Figure 1.2 a data warehouse may serve as a historical source for a BAM system. BAM systems don't replace data warehousing. A data warehouse (DW) is a place to accumulate and aggregate data, typically for historical analysis. A data warehouse generally contains permanent data storage.

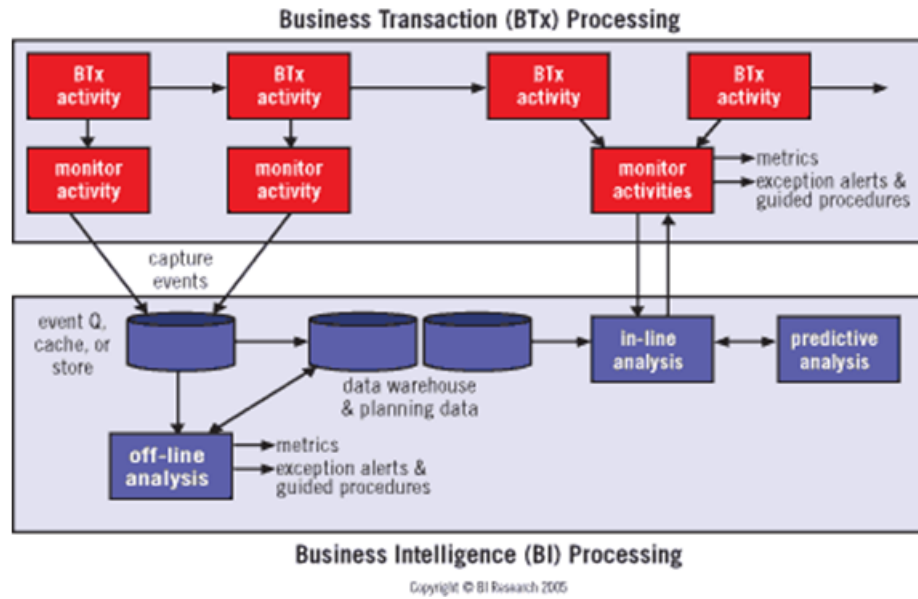


Figure 1.2: Integration of BI and business transactions; Data warehouse as a source for BAM solutions;
Source: [2, Colin White]

BAM is often associated more with technology infrastructure than business needs and therefore is often called as operational dashboard builder. A dashboard means more to a business manager than an acronym such as BAM. Dashboards will be approached later in this chapter.

BAM systems exhibit the following characteristics [3]:

- **Event-driven processing model** that captures events in real time from multiple sources - message queues, Web services, databases, context-oriented sources, etc.
- **Perform dynamic modelling** by integrating event and contextual information on the fly, producing a stream of analytical models automatically updated on subsequent event or contextual information, for decision support.
- **Robust business rules** that let users define alerts, targets, and thresholds for key performance indicators. Generated decisions are transformed to actions which are captured within the BAM system and forwarded directly into the underlying systems for execution. Responds are integrated again in to the decision in order to improve the decision support.
- **A business-user-friendly dashboard** that updates metrics as events flow through the system and puts metrics into context by relating them to business objectives.
- **A collaborative work flow system** that lets one set up formal and informal processes by which users can collaborate and discuss results.

This challenges BAM ability to combine high-speed streaming of events with the dynamic modelling needed for event-context correlation.

1.2 Why Business Activity Monitoring?

Enterprises constantly look for ways to increase speed and flexibility inside changing markets. Nowadays, to make it possible, they use a variety of business applications, such as customer relationship management (CRM), SAP, and order management, purchased or developed internally over time. These applications frequently use distinct technologies (from COBOL to C# and Java) and run on heterogeneous operating systems. Nevertheless, a typical enterprise deals frequently with processes that are still based on human actions, such as phone calls, faxes, and e-mail. It turns to be increasingly difficult to see the activity in the business in such a complex environment. This way, it is crucial for enterprises to make quick decisions so they can take advantage of market opportunities or to prevent losses.

BAM can be used as a monitoring solution by IT managers who want to cut the cost of their distributed IT environments while improving service quality. BAM provides management oversight of the business critical operations improving its effectiveness through access to real-time KPI's. BAM offers a vertical view of high-performance workstations and applications and leads to ways to optimise performance.

Having BAM concept and utility clarified it is possible to identify the concept main components as monitoring events, analysing data fetched, and reporting it.

1.3 Monitoring

Monitoring involves tracking and collecting information about a business operation, which is easier if the operation being tracked has been implemented in terms of the business activities required to carry out the operation. This granularity enables the monitoring task to track just those activities that are important from a performance perspective. Otherwise the monitoring task can only track the complete business operation.

One might have the impression that speeding up a data warehouse to a near real-time processing can be called, BAM solution, which is completely wrong. To better understand it there are some characteristics [4] that makes that impossible such as:

- **DW can not be rule driven** like BAM due to data reloading and cube processing which makes it impossible to dynamically change business rules.
- **Data modification in the DW** even if its made in near-real time, no one in the enterprise

will notice that it happened. In BAM analytics run on the events as they are generated.

- **Loading large volumes of data into a DW can delay events** and reports can only be produced after that. This makes it impossible to warn users about trend exception when they occur. In BAM, exception are acted upon as soon as they occur due to its highly optimised latency between event and action.
- **BAM handles information of specific business processes.** DW deals with a wider range.

1.3.1 Complex Event Processing

In many applications, events have to be inferred in real-time and the applications simply can't wait for the raw data to be persisted and analysed. BAM is an event driven solution which is capable of monitoring a stream of business transactions and providing near real-time reports and dashboards. Business processes can be quite complex since they require communication between multiple entities. This style of analytical processing involves what is known as stream analytical because the events are analysed as they flow across networks and systems. Stream analytics requires massive amounts of event processing and the underlying technology that supports it is known as complex event processing (CEP). Thus CEP is an essential component of BAM solutions.

Applications have implemented functionality for inferring events from existing data for a very long time. The current data avalanche is rapidly changing design-detection algorithms and the need for a configurable and flexible way to detect patterns is becoming more vital. CEP is a parallel running platform that analyses and processes events. It deals with the event-driven behaviour and so the task of processing multiple events. According to Colin White[5], president of BI Research, the trend of the market is to move towards real-time processing involving stream analytics and complex event processing for certain types of businesses. CEP solutions can analyse and correlate multiple streams of current data looking for patterns and trends. Therefore BAM turns into an intermediate layer between business users and information flow in business processes as shown in Figure 1.3. CEP engine deals with heterogeneous events from various sources. Events such as messages passing between applications over a middleware system, or transactions detected within a database log file, are processed by the CEP engine. Additionally new events can be inferred by correlating data from multiple primary data sources. From a black box view, a CEP engine takes as input a set of streams, database files, or Java Message Service (JMS) objects like queues and topics (to be explained in the next chapter), and as output produces event pattern matching and event correlations. The CEP platform can gather events fired by business process and also by external sources.

There are quite a few CEP products in the market nowadays and there is a need to filter

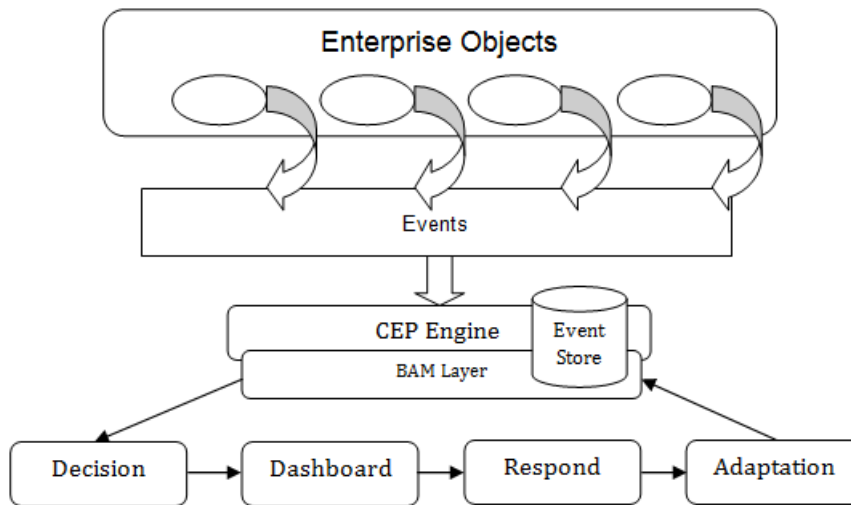


Figure 1.3: BAM as a layer over CEP engine

them in order to choose the most effective. When filtering this products it should be taken into consideration the following characteristics [6]:

- **The performance of the non-core parts** of the system (e.g. plugins) can also have a dramatic impact on the overall performance of a CEP application.
- **Infrastructure fit:** refers to how well the platform fits into their overall environment since some are difficult to connect with (e.g. Can the CEP engine connect to the messaging bus?).
- **Application development model:** refers to the ease of actually building and deploying applications in the system.
- **Total cost of ownership (TCO):** Even the platform itself must be deployed and managed which as a cost.

1.3.1.1 Patterns

Live analytic applications allow to filter, aggregate, slice and dice data and require the ability to run dynamic queries against the real-time data being managed in the CEP engine.

The event modeller defines event patterns for a BAM view on the basis of an Event Processing Language (EPL). Nowadays there is no standard EPL [7] although there are some solutions for event processing languages. Some provide SQL-like (Structured Query Language) language, rule-based EPL or abstract user interface that generate code like Java. Patterns are found when events occur regularly and always lead to the same result. After a pattern has occurred CEP is able to create a complex event which may trigger proactive reactions. The CEP engine queries

the event streams according to predefined CEP patterns for detecting relevant complex events. These events are displayed within a BAM component in dashboard views.

1.3.2 Choosing the Messaging Protocol

Choosing the correct messaging protocol for the data stream, can fully leverage the high-performance of a CEP engine. There are some criteria to be considered [8] when choosing a messaging bus for the CEP needs, such as:

- **Maximum Latency:** is a concern for developing a successful application, even if the total average latency of a system is low, if it periodically experiences lags, that can cause a serious competitive disadvantage.
- **Managing network bandwidth efficiently.**
- **Quality of Service (QoS) for data streams:** QoS refers to a service contract made between two entities. Each data stream has unique attributes or characteristics.
- **Number of messages per second:** Use cases requiring CEP typically demand that messages be transmitted at a rate of 1,000/second to 500,000/sec. This is understandable considering that the CEP application typically integrates with edge devices or, as in trading applications, process a large amount of market data from multiple exchanges to make trend inferences.
- **Supporting heterogeneous platforms:** data streams from different sources. While different data streams can use their own messaging protocols, this poses a needless headache for application architects by having multiple technology stacks for their data stream implementations. Messaging protocols like JMS are supported on major enterprise platforms.

"The end result of an ideal event-processing application is that the business logic is a clean, declarative expression of the relevant business rules, yielding a system infinitely easier to develop, maintain, and extend."

[6, Chait, D.]

1.3.3 Real-Time Data Storage

Although BAM is not a data warehouse, for most solutions, storing event data, even if temporary, is important because it allows further operations and analysis. Storing data is also useful for the application that is using the data, which exits for any reason, when comes back the current state of the data can be retrieved where it left off.

A message between applications is related to a set of structured data that might contain information about a customer's order as shown in Table 1.1. The fields Customer, URL and Item are often related with characteristics that specify the data and by which data is analysed. One might want to know how many orders came through each website, whereas another might want to know how many specific items were ordered. The fields Quantity and Price contain the actual measurements. There can be more than one value in the same time interval sample. This is referred to as multi-dimensional data which are several columns of data related by a common time stamp. This nature influences how data will be analysed and visualised. Monitoring applications

Timestamp	Customer	URL	Item	Quantity	Price
10:34	Mary	www.a.com	Perfume	1	25
10:35	John	www.b.com	Toy	2	5

Table 1.1: Set of Structured Data

are typically developed with independent processes running on one or more systems [9]. Each process performs a separate task such as communication to other systems, I/O, control logic, etc. For matters of analysis each data row must be stored long enough to compare with previous data.

For this reason much real-time data arrives asynchronously meaning that events from different sources may occur at different times and out of sync. Real-time data comes in and is gone by the time the next record shows up, therefore all real-time data must contain a time stamp which is necessary to perform trend analysis.

In this situation the system needs to maintain a current value table (CVT) [10]. The CVT is seen as the central data component and can be used in a wide range of applications. It allows other applications to share a common data repository and have direct access to the most up-to-date values used between them. This way, operations such as alarm detection, user interface updates, process logic, etc. are handled by separate processes that share the same data repository. The CVT is indexed by the columns of data that uniquely identify the source of the data, providing access to the latest data for each uniquely indexed record. However it is a complex task to maintain a CVT with a relational database. It would be necessary to perform continuously a query against a large set of data to determine the latest value for a specific source. Instead, the ability to create a CVT in memory is an important feature for the BAM solution.

Like the CVT the in-memory cache of prior data must be indexed as well. This system provides efficient access to a time window of stored data which is essential for performing analytics on the data in real-time such as detection of anomalous changes in trends.

Real-time data from most BAM sources is also in row form so it seems natural to archive data in a relational database and this way reporting tools to process historical data and generate reports

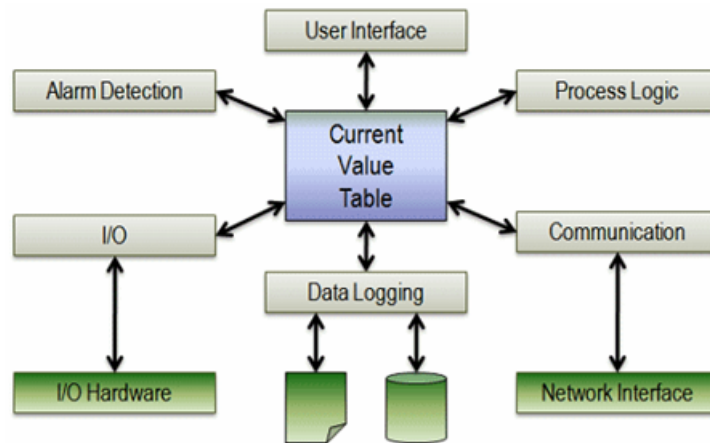


Figure 1.4: Placing the CVT in context of the application functional blocks. Source: [9]

could be used.

The display can obtain historical data from the database making SQL queries or metrics

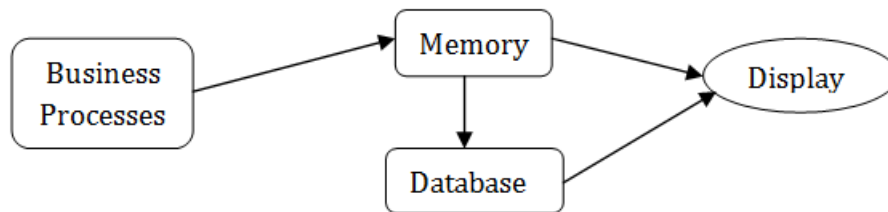


Figure 1.5: Real-time data flow with relational database as persistent storage

from the data server memory which can also be archived. The memory cache provides efficient access to short-term data for analysis while database provides access to long-term data for longer periods. The problem is that it is not possible to perform time-range aggregations on historical data in a way that is portable across all databases. Because of it memory cache is used to perform such functions. Another way to do this is to use OLAP database that perform time-series functions in-memory, although OLAP queries syntax and cube configuration is quite complicated.

A Relational Database Management System (RDBMS) offers too many features, is often too large and complicated. On the other hand the file system is underpowered since it offers few or none of the integrity, concurrency and performance advantages of a database system. There are some products specialised in time-series data storage. Usually these products are able to compress a set of data into a single record if that data is unchanged. Average is a possibility

to reduce the amount of memory required to store large amounts of such data. Since data in BAM is typically multi-dimensional this introduces a complexity that makes them difficult to use. Multiple related measures need to be stored together in a single record. For this reason only few products have penetration in BAM solution. Some open source products are: Round Robin Database (RRDtool) and Oracle Berkeley DB Java Edition.

1.4 Analysing

The analysis step processes the information collected by the monitoring step and creates a set of metrics documenting the performance of the monitored business activities [2]. This analysis can be done synchronously and in-line as a part of the main business process, or it can be done outside of the main business process. In this later case, the monitoring information can be routed to an asynchronous in-line process for analysis, or it may be stored in a message queue or persistent store for off-line analysis by an independent application. The method chosen will depend on how the timing requirements of the business application.

Analysis and visualisation of real-time data are highly related. There are commonly used analytic functions like averages, totals, least-square trends, etc. Much useful information can be extracted using simple aggregations and breakdowns. A CEP engine is a valid solution in various situations, such as fraudulent detection. Nevertheless, a large percentage of BAM requirements are a tight coupling between analytical functions and graphical objects.

Functions provide aggregation and breakdown of multi-dimensional data and graphical objects directly display the results of the analysis. This analysis functions are, therefore, transformations of tabular data from one form to another. This way, imagining a store's sales data in a table structure, is possible to transform that table into a smaller one breaking down the number of sales of each day by each employee and fed it into a bar chart to display the information. The advantage is to eliminate or reduce programming involved in a sophisticated BAM solution development. Common patterns are: aggregation and breakdown - group by, and baseline trends [10]:

- The group by operates on data tables, aggregating the data in different ways (sum, count, average, min, max, etc) and showing them directly on a bar chart, table, area graph, map, etc.
- The baseline trends in BAM applications refer to typical range where data should stay within for the current time period as shown in Figure 1.6. Data being monitored may fluctuate wildly in circles up and down during the day, for example the number of sales in a Web site will vary according to the time of the day and day of the week. Alerts can be

configured to fire only when the value of interest goes outside the baseline.

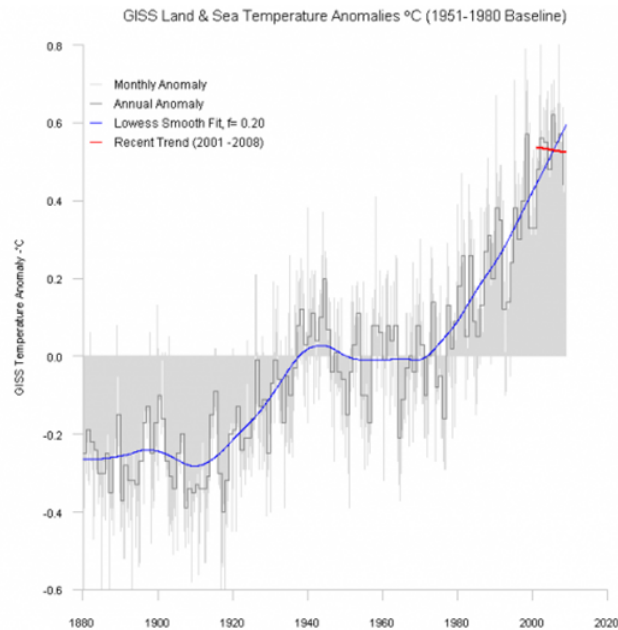


Figure 1.6: Baseline trend; source: <http://chartsgraphs.wordpress.com/2009/01/19/GEORGE-WILLS-INTERPRETATION-OF-GLOBAL-TEMPERATURE-TRENDS-IS-FLAWED/>

1.4.1 Constraints

A BAM tool should allow monitoring for a certain policy which is not to be violated by the business policy of the enterprise. For example if there is a sale made and the business process automatically order a restock event, there are some situations that might not be necessary to order like if there is still a high stock of that particular stock. In that case, would be interesting to enforce the constraint: Don't order stock until it reaches a critical level. The BAM tool interface should provide changing constraints on the fly. This way complexity is kept out of the business processes. [11]

1.5 Reporting

The reporting step involves displaying or delivering the results of the analysis step. This may be done via a portal, dashboard, e-mail, instant message, etc, depending on the preference of the user and how quickly the information must be acted upon. In some situations, an alert may also be sent to highlight important information.

The decision and action step that follows the report of information is the most critical aspect. This enables business users to optimise business operations and align those operations with the goals of the company.

In Business Activity Monitoring the process being monitored is typically abstract. Dashboard is definitely the most common way for actual BAM solutions to report its monitoring results such as work flow statuses, work flow instances, KPI's results, etc. Business processes are often represented as "bubbles" with an icon inside indicating the process nature. The state of each process can be represented as values, colours or icons on the bubbles. Values representing the time difference between process steps can be shown on the lines connecting the bubbles.

A display used for visualisation of real-time data must be designed with reference to data elements whose values will be shown in the objects on the display. In order for a display to be used multiple times against different sources of data, those references need to be parametrised. When a user selects an object in a display or a row in a data table, there needs to be an easily configurable way for the system to pick up the value of the parameter driving the selected object, and pass that value down as a parameter to a new display invoked in the drill down process. How to develop parametrised displays that can be reused in a variety of situations is the real problem.

1.5.1 Dashboards

One of the most common features of BAM solutions is the presentation of information on dashboards that contain KPIs used to provide assurance, visibility of activity and performance.

In the 90s, companies began experimenting ways to give business users direct and timely access to critical information. This way emerged the idea of having a "Business Cockpit" to drive the organisation, inspired in automobile and aircraft data presentation panels. However, in contrast to an aircraft or automobile, each organisation has a set of KPIs that differs significantly between organisation. At the dawn of the 21st century, BI converged with performance management to create the performance dashboard.

For enterprises, dashboards characteristics and benefits turned out to be pretty clear [3, 12]:

- **Interactive and visually appealing:** Should allow the user to drill down and get to details, root causes, etc, in an ergonomically and visually effective screen view with different aspects of the business.
- **Status on the fly:** provides a clear real time picture of the strategic objectives and what

is required in all areas to achieve their goals. Provides alerts in addition to the visual presentation on the dashboard (e.g. sound, e-mails, pagers) to draw immediate attention.

- **Refine strategy:** performance dashboards are like a steering wheel to make a series of minor corrections. Information being presented must be entirely accurate in order to gain full user confidence. It should allow each user to customise the metrics.
- **Increase visibility:** access to daily operations and performance by collecting relevant data and forecasting trends based on past activity. Must display critical KPIs for effective decision. The dashboard should allow users to review the historical trend for a given KPI.
- **Consistent View:** consolidate and integrate business information using common definitions, rules, and metrics. This creates a single version of business information.
- **Self-service:** easy access to information, eliminating reliance on the IT department to create custom reports.
- **Team work:** encourage different departments to work more closely together. Collaboration would serve as a communication platform.
- **Motivation:** compels people to work harder out of pride and desire for extra pay when compensation is tied to performance results.
- **Personalised:** The dashboard presentation should be specific to the user's domain of responsibility, privileges and data restrictions.
- **Intuitive:** Users learning curve should be a short one.
- **Secured and Scalable through the Web:** should be accessed through the Web with data encryption to secure sensitive data transmission. A huge number of users may access the dashboard without crashing the system or causing it to slow down below predefined value.

Dashboards are more than just a screen populated with fancy performance graphics: it is a business information system designed to help organisations optimise performance and achieve strategic objectives. It provides a set of indicators about the state of a process (e.g. stock, sales by shop in a specific month, energy consumption, etc) as shown in 1.7. An enterprise dashboard is designed to be an effective tool to be deployed at various levels within the organisation. It should not be used as a simple report distribution tool for KPI viewing as it can also warn the user in an effective manner when any relevant metrics are out of acceptable boundaries, fulfilling BAM's alerting purpose.

In general, organisations use dashboards to monitor operational processes. Some authors use

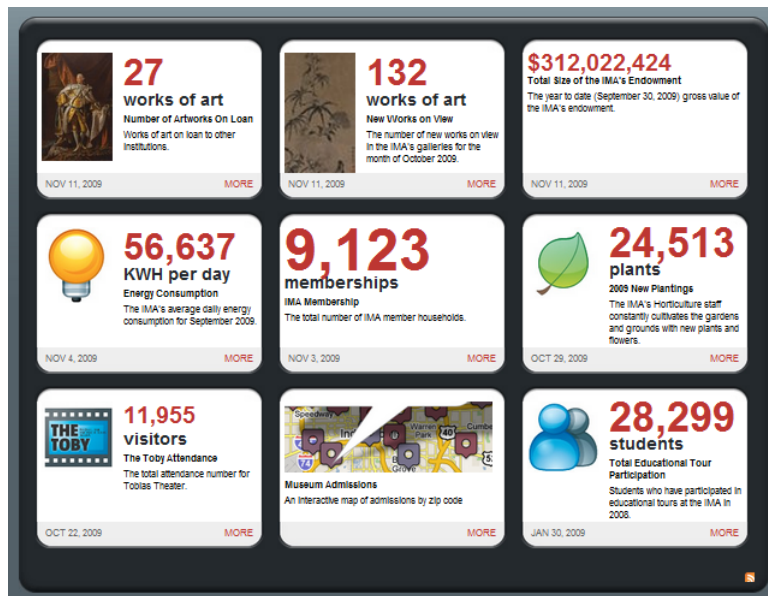


Figure 1.7: A museum Dashboard example; source: <http://dashboard.imamuseum.org/>

the term scoreboard when referring to the displays used to monitor tactical and strategic goals. Dashboards and scorecards are visual display mechanisms within a performance management system that convey critical performance information at a glance.

Dashboards allow supervisors to monitor performance through events generated by key business processes displayed visually, using charts or simple graphs.

A scoreboard informs if the company is meeting its objectives in terms of vision and strategy providing information like for example, call center sales or frequency of potential customer contacts. Scorecards usually display monthly summarised data for business executives who track strategic and longterm objectives, or daily and weekly snapshots of data for managers who need to chart the progress of their group or project toward achieving goals.

The combination of both information presentation forms, as in Figure 1.8, generates a full and controlled understanding about the enterprise performance.

1.5.1.1 Layers

A performance dashboard is a multilayered application. Each successive layer provides additional details, that enable users to understand the problem better and identify what is necessary to address it [3]:

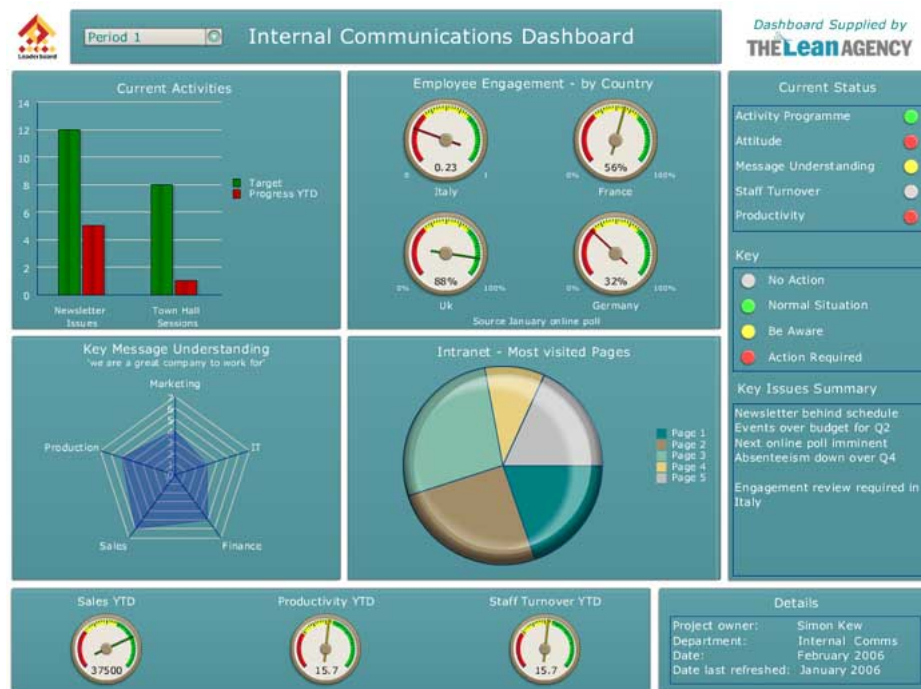


Figure 1.8: Performance dashboard example; source:<http://www.enterprise-dashboard.com/2006/10/11/communications-performance-management-dashboard-using-xelsius-to-display-internal-communication-kpis/>

- **The Summarised Graphical layer** provides a summarised view, usually graphical, of the status of key performance metrics and exception conditions. Exceptions can be in the form of alerts that pop up on users screens or arrive via e-mail, etc.
- **The Multidimensional layer** provides the data behind the graphical metrics and alerts. Users navigate the data by dimensions and hierarchies (applying slice and dice, drill down or up, etc).
- **The Reporting layer** lets users view detailed reports and transaction records. The resulting report or query results are then usually displayed in a separate window, which users can view or print.

1.5.1.2 Types of Dashboards

There are three types of performance dashboards [3, 12]:

1. **Operational or performance dashboards** monitor core operational processes and deliver detailed summarised information. Operational dashboards emphasise monitoring more than analysis and management

2. **Tactical or divisional dashboards** monitor departmental processes and projects that are of interest to a limited group of people of the organisation. Operational managers require dashboards that display performance metrics and numbers specific to their areas of responsibility. Usually updated daily or weekly with both detailed and summary data. They tend to emphasis analysis more than monitoring or management.
3. **Strategic dashboards** monitor the execution of strategic objectives. The goal is to get the entire organisation marching in the same direction. Usually updated weekly or monthly, provides a powerful tool to communicate strategy, gain visibility into operations, and identify the key drivers of performance and business value. Emphasise management more than monitoring and analysis.

Although BAM systems usually use a dashboard to present data, BAM is distinct from the dashboards used by BI. BI dashboards refresh at predetermined intervals by polling or querying databases, whereas BAM events are processed in real-time or near real-time and are pushed to the dashboard. Depending on the refresh interval selected, BAM and BI dashboards can be similar or vary widely.

1.5.1.3 Portal vs Dashboard

A dashboard is an application with a collection of metrics, results, alerts, etc, whereas a portal is a collection of applications presented together within a personalised framework [12]. For this reason a dashboard could be part of a portal, but not the other way around. A dashboard is intended for the presentation of organisational and individual performance metrics and alerts. On the other hand, a portal might have a dashboard, an events calendar, weather conditions, etc.

1.5.2 Alerting

There is no doubt that alerts are an integral component in BAM concept since they turn a graphical information monitor into an organisational manager. A rules engine helps in managing the business rules allowing the dashboard to drive the alerts by monitoring the rules. Using complex rules helps in grouping them such that if any is triggered, the same action is instantiated.

For a better KPI monitoring, different users have distinct domain responsibility and each one of them may create personal alerts whose the recipient would be that specific individual only. The ability to create public and critical alerts may be confined to a restricted user base like an administrator who may select any number of users and user groups to receive a given alert.

The effectiveness of a rules engine in a BAM is all about its flexibility on adapting to any possible business rule required.

1.6 Doing it

By cooperating with the department or even with the management level of an enterprise, the event modeller has to define which BAM view has to be monitored in a dashboard, which alerts are to send to which roles in the organisation and which actions shall be started automatically if a certain event pattern occurs.

Once BAM views are defined, the event modeller looks for the needed event types and their instances flowing through the event streams of an enterprise or which are saved in an event store. It is a highly skilled task to define the right event patterns for a near real-time BAM view. The event modeller has to know the different event sources like messages types in the Java Message Service (JMS) which are realised on the basis of a publish/subscribe model. He has to install the corresponding event adapters delivered by the CEP platform as "out of the box" pre-built features or the event modeller has to care about the development of not yet existing adapters.

1.7 Planning

This project plan was divided into three different stages. The first stage refers to the BAM concept investigation, state of the art definition and platform analysis. According to planned this first stage took approximately 2 months to accomplish.

The planned second stage is about performing a system requirement analysis and defining the solution foundations, such as base concepts and choosing the supporting technologies, based on the solution's needs. This stage took about three weeks to accomplish, one more than planned.

The third and last stage of development took about three months to accomplish and has to do with the prototype development process.

The project planning chart is available in appendix for consult.

1.8 State of the Art

Ever since the BAM concept was born in 2001, enterprises have gradually shown interest in this kind of BI solutions. Organisations are beginning to understand that they need to use these technologies in order to control their business processes in a more effective way.

In this late years a few other concepts like CEP and business dashboards also emerged, which made BAM solutions a viable scenario in enterprise business management. Processing event streams requires robust and quick solutions. For this reason many analysts claim that BAM and CEP are complementary.

Existing BAM solution functionalities consist mainly of monitoring user-defined KPI's and visualisation of real-time colourful graphs. These functionalities use a first generation BAM environment which is not enough for a real-time enterprise. These solutions work mainly on reporting instead of analysing, and in addition only past activities are considered without predictions for future business.

Today vendors have some interesting CEP, and dashboard tools. Some have also tried for a complete set of tools to build BAM solutions. Progress Software's Apama[13], Coral8 (merged with Aleri)[14], Oracle[15] and SL - real time visibility[16], are some of the vendors that have a complete set of tools for BAM solutions development. However, these are not open source platforms, which is the main interest of this project. As an open source CEP solution, there are only few available like Esper and Pion, nevertheless Esper Tech is widely referred as *"the only open source option"*[17].

Today's CEP platforms in the market are mainly divided into three Event Pattern Language (EPL) different approaches [7]:

- SQL-like (e.g. Coral8, Esper, Oracle, StreamBase);
- Rule-based (e.g. AMiT from IBM or Reaction RuleML from RuleML);
- Abstract user interface and java-like code generation (e.g. Tibco, AptSoft).

On dashboard domains, performance dashboards are one of the latest creations of business intelligence. These solutions were built on years of technical and process innovation. Performance dashboards meet the users requirements when speaking of enterprise business intelligence and more specifically by reporting and analysis capabilities within an intuitive dashboard interface. Performance dashboards aim to deliver access to the right data to the right people at the right time to optimise decisions and accelerate results. There are some interesting dashboard builders in the market such as Progress Software, Coral8 but once again, they are not open source solutions. Pentaho also shows a powerful dashboards platform, however it integrates Pentaho's enterprise edition which is commercial.

1.9 Summary

After studying the Business Activity Monitoring concept, as a conclusion it is possible to say that a BAM solution provides three main features to the user:

- Near real-time business key performance indicators monitoring. This feature usually uses charts to display the information in order to ease its reading by the user. This is called the dashboard component of a BAM solution.
- Alerts to notify target users about important previously defined business behaviours. This may require event correlation and pattern detection. Notification can be any way to alert someone but in general is also email oriented. This is often called the alarmist component of a BAM solution.
- Detailed report generation with system behaviour important information. Since we're dealing with a BAM solution and not a data warehouse one, it is not expected from to deliver long reports. According to the BAM concept it is acceptable for it to deliver at maximum the last 24 hours behaviour report.

Due to the fact that there are no free open source solutions in the market, there is no way to compare directly the BAM prototype against other existing solutions. Nevertheless, this BAM solution tends to implement main features that other BAM solutions provide.

Chapter 2

Pre-Requisites and Studying Platforms

Before studying which are the best open-source platform available for the production of a BAM solution, it is necessary to introduce the conditions on which the product will be running. This BAM solution is targeted to run over a Order Management System (OMS). The targeted OMS framework was developed by Xpand IT and is focused on a typical services company.

This chapter will describe the needs and requirements that the OMS has and which will influence the platforms selection.

The studied platforms characteristics will also be described, as well as their advantages and disadvantages and matched against each other in order to choose the platform to be later used in the solution implementation.

2.1 The Order Management System

An OMS is a software system for order processing within an organisational business process which empowers the coordination and execution of an order, given the priorities, error handling and data transformation, measuring the results and informing the interested systems. The targeted OMS uses jBPM, JBoss ESB, JBoss Messaging technologies and Oracle DBMS. Let's start by understanding the OMS's supporting technologies with a general study for each one of them.

2.1.1 JBoss Application Server

JBoss Application Server (JBoss AS) is a free software/open-source Java EE-based application server which makes it cross-platform. Its a Java EE certified platform for developing and deploying enterprise Java applications, Web applications, and Portals.

JBoss AS supports the addition of some Modules/Enterprise Applications with integration purposes, including clustering, caching, and persistence. JBoss AS was developed by JBoss, now a division of Red Hat.

For more information about JBoss AS please consult [18].

2.1.2 JBoss Enterprise Service Bus

The JBoss Enterprise Service Bus (JBoss ESB) project provides a lightweight standalone ESB server. It can be used as a standalone server or it can be used to integrate several JBoss application server. JBossESB is part of a Service Oriented Infrastructure (SOI). As shown in Figure 2.1,

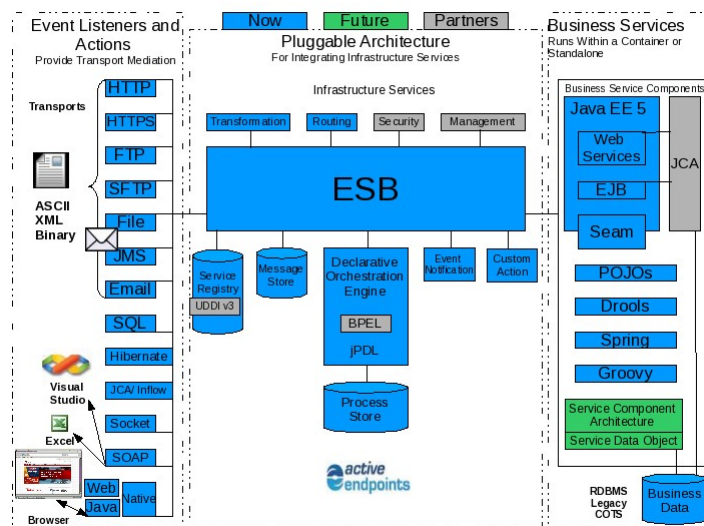


Figure 2.1: JBoss ESB architecture

Source: JBoss at <http://www.jboss.org/jbossesb/>

JBoss ESB uses a flexible architecture based on Service Oriented Architecture (SOA) principles such as loose-coupling and asynchronous message passing, emphasising an incremental approach to adopting and deploying a SOI.

For more information about JBoss ESB please consult [19].

2.1.3 JBoss Messaging

JBoss Messaging is composed of several services working together to provide Java Message Service (JMS) API level services to client applications. JBoss Messaging is JBoss's JMS implementation

and therefore is a Message-Oriented-Middleware (MOM). JBoss Messaging only runs with Java 5 or later.

For more information about JBoss Messaging please consult [20].

2.1.4 JBoss Business Process Management

BPM deals with the management of business with the approach to increase the efficiency, flexibility and technology integration of the business. Business processes describe the unique way of doing business. Today they are seen as the most valuable asset of a corporation.[21]

Business process management (BPM) promotes a model-based approach to task definitions: it encourages a top-down approach to service orchestration and requirements. Since the process are defined by the businessmen this should ensure that they drive the business the way they want. BPM attempts to improve processes continuously.

JBoss Business Process Management (jBPM) is a flexible, extensible framework for process languages. JBoss Process Definition Language (jPDL) is one process language that is built on top of that common framework. It is a process language that expresses business processes graphically in terms of tasks, wait states for asynchronous communication, timers, automated actions and other components.

jPDL has minimal dependencies and can be used as easy as using a Java library. In addition, it can also be used in environments where extreme throughput is crucial by deploying it on a Java Enterprise Edition (JEE) clustered application server. It can be configured with any database and can be deployed on any application server.

Unlike Business Process Execution Language (BPEL), which is tightly coupled to Web Services and Web services invocation (in BPEL, every activity has to be implemented as a Web service), jBPM is more of a component framework, allowing direct invocations of the Java handlers (similar to an ESB service execution pipeline).

For more information about JBoss jBPM please consult [22].

2.1.5 Oracle Database Management System

Oracle Database Management System (DBMS) is a database management system produced and marketed by Oracle Corporation. Oracle offers the following editions:

- **Enterprise Edition:** Has no memory and no Central Processing Unit (CPU) limits. It can use clustering with Oracle Real Application Clusters (RAC) software.

- **Standard Edition:** Typically used for servers running from one to four CPUs. has no memory limits, and can utilise clustering with Oracle RAC.
- **Standard Edition One:** Has some additional feature-restrictions. Oracle Corporation markets it for use on systems with one or two CPUs. It has no memory limitations.
- **Express Edition:** To be distributed on Windows and Linux platforms. It uses a maximum of 1GB of memory, is restricted to the use of a single CPU, and a maximum of 4 GB of database size.
- **Oracle Database Lite:** intended for running on mobile devices. Oracle DBMS enterprise edition protects from server failure, site failure, and human error. It enables data warehousing, online analytic processing, and data mining.

For more information about Oracle DBMS please consult [23].

2.1.6 Finding Connection Points

Through an OMS it is possible to decompose an order, allowing to automatically determine the systems to be used, as shown in Figure 2.2, sending specific messages to each one and this way controlling the assumed commitments. To be able to structure the BAM solution being

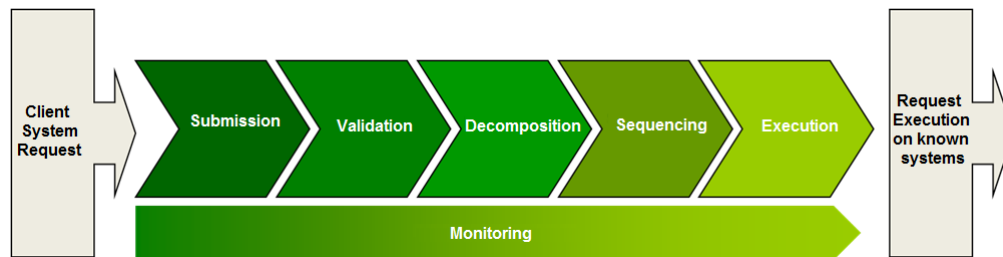


Figure 2.2: High level perspective of an Order Management System. Source: XPand-IT documentation. **Submission** - Receives the client request and produces the representation of one or more actions taken by the client application. **Validation** - Guarantees the quality of data being submitted. **Decomposition** - The request is transformed into smaller granularity technical requests, which are going to be executed in the respective systems. **Sequencing** - Orders the technical requests so the execution success is guaranteed, as well as the final state coherence. **Execution** - Represents the technical request execution on the target system. Triggers the execution state update as well as the conclusion notification to the order source system.

developed and to define the essential requirements, that the technologies being concerned in the development should fulfil, there are a few questions to be answered:

- Available connection points for event gathering and processing;
- Estimated event flow to define CEP engine minimum performance required.

2.1.6.1 Event Sources

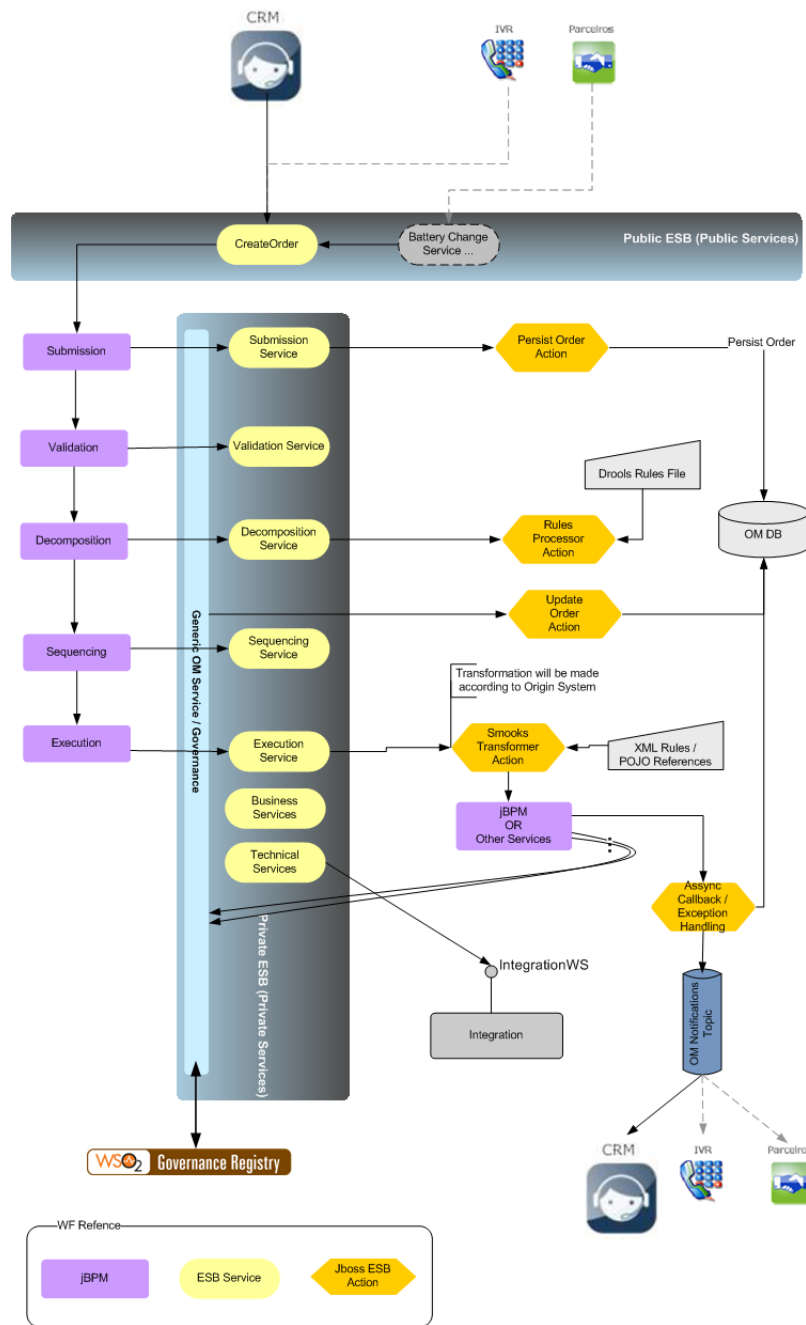


Figure 2.3: OMS architecture. Source: XPand-IT documentation.

As shown in Figure 2.3, there is a clear connection point for event fetching which is the

OMS Notifications Topic. There are other connection points that are not represented in the architecture diagram such as JMS Notification Queues along with the order flow. Both produce various types of events which are to be studied later in this report.

2.1.6.2 OMS Event Flow

In the OMS built by Xpand-IT, it is expected for the worst case performance that the event flow generates around 100.000 events per day. For a normal behaviour, the expected are 10.000 events per day. This means that since a day has $(60 \times 60) \times 24 = 86400$ seconds, there will be approximately $\frac{100.000}{86400} \approx 1.16$ events per second in the worst case.

2.1.7 JMS Notifications

The Java Message Service (JMS) defines the standard Enterprise Messaging also known as MOM which is recognised as an essential tool for building enterprise applications. The JMS API provides a powerful tool for enterprises by combining Java technology with enterprise messaging. Enterprise messaging provides a reliable and flexible service for the asynchronous exchange of critical business data and events within an enterprise. JMS provides two ways of asynchronous messaging: Queues (point to point - PTP) and Topic (Publish/Subscribe Model - multiple Senders and multiple Receivers). Table 2.1 presents a corresponding between point-to-point model preferred interfaces and Publisher/Subscriber model preferred interfaces.

JMS Common Interfaces	PTP Domain Interfaces	Pub/Sub Domain Interfaces
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver	TopicSubscriber

Table 2.1: Corresponding PTP interfaces with Publisher/Subscriber interfaces.

2.1.7.1 Point to Point

Point-to-point (PTP) systems are about working with queues of messages. When one process needs to send a message to another process, PTP Messaging can be used. However, this may or may not be a one-way relationship. The client to a Messaging system may only send messages, only receive messages, or send and receive messages. At the same time, another client can also send and/or receive messages. In the simplest case, one client is the Sender of the message and the other client is the Receiver of the message.

To achieve this behaviour a message publisher acquires a reference to a JMS Queue and when a message arrives, it persists in a Queue until either it times out, or until some receiver comes along to retrieve the message (one-to-one).

2.1.7.2 Publishing and Subscribing

When thinking of a messaging system the wanted behaviour is an operation where a single publisher sends each message to multiple subscribers with a single method call. In JMS behaviour there is a catch which is the existence of a messaging server, also known as JMS provider, between the publisher and the subscriber as shown in picture 2.4. To achieve this behaviour a message publisher acquires a reference to a JMS Topic or Queue on a server, and sends messages. When a message arrives, the JMS Topic provider notifies all message consumers subscribed (one-to-many). By contrast, in point-to-point messaging, a persistent message sits in a Queue until

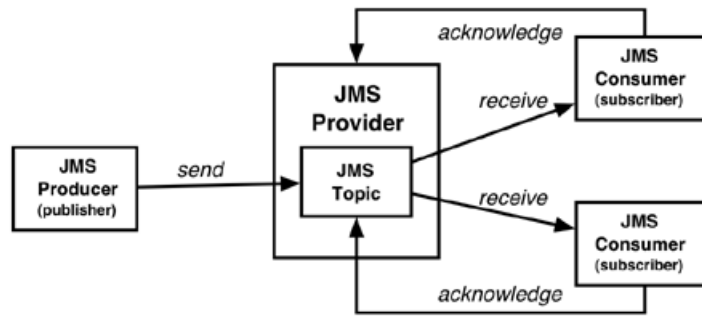


Figure 2.4: In JMS, publish/subscribe messaging uses a JMS-managed object called a Topic to manage message flow from publishers to subscribers. Source: SUN Microsystems at <http://java.sun.com/developer/EJTechTips/2003/tt0415.html>.

either it times out, or until some receiver comes along to retrieve the message (one-to-one). The JMS provider (optionally) receives acknowledgement of the message receipt each time it sends the message.

In JMS, messaging can be object-oriented, transactional, synchronous or asynchronous and integrated with underlying third-party products. A message may be sent to a message consumer (QueueReceiver or TopicSubscriber) that is not running at the time the message is sent. The JMS provider stores messages that can't be delivered because the subscriber is unavailable. The stored messages are delivered the next time the subscriber connects. This ensures delivery of all messages published after the client subscribes to a Topic. A function that sends a message returns as soon as the message is delivered and the message receipt can be acknowledged explicitly or automatically.

According to specification [24] there might be latency in the system, meaning that the exact messages seen by a subscriber may vary depending on how quickly a JMS provider propagates the existence of a new subscriber and the length of time a provider retains messages in transit. When a new subscriber is created, it may receive messages sent earlier because a provider may still have them available.

Persistent messages in publish/subscribe messaging are provided by "durable subscriptions". A durable subscription can be used to preserve messages published on a topic while the subscriber is not active. At the cost of higher overhead, a subscriber can be made durable. A durable subscription as an unique identity that is retained by JMS. Subsequent subscriber objects with the same identity resume the subscription in the state it was left in by the prior subscriber. If there is no active subscriber for a durable subscription, JMS retains the subscription's messages until they are received by the subscription or until they expire. The JMS provider stores mes-

sages that can't be delivered to a subscriber because the subscriber is somehow unavailable. The stored messages are delivered the next time the subscriber connects. This ensures delivery of all messages published after the client subscribes to a Topic. This enables subscriber applications to operate disconnected from the JMS provider for periods of time, and then reconnect to the provider and process messages that were published during their absence.

Each JMS durable subscription is identified by a subscription name (sub-Name), which is defined when the durable subscription is created. A JMS connection also has an associated client identifier (clientID), which is used to associate a connection and its objects with the list of messages (on the durable subscription) that is maintained by the JMS provider for the client. The sub-Name assigned to a durable subscription must be unique within a given client ID.

If a subscription isn't durable, any messages published while the subscriber is down are never delivered to the subscriber.

Clients that desire concurrent delivery can use multiple sessions. In effect, each session's listener thread runs concurrently. While a listener on one session is executing, a listener on another session may also be executing. Note that JMS itself does not provide the facilities for concurrently processing a topic's message set (the messages delivered to a single consumer). A client could use a single consumer and implement all the multithreading logic needed to concurrently process the messages; however, it is not possible to do this reliably, because JMS does not have the transaction facilities needed to handle the concurrent transactions this would require. [25]

2.1.7.3 Message Types

A JMS Message is composed of three parts: Header, Properties and Body. JMS provides five forms of message body each defined by a message interface:

- **StreamMessage:** a message whose body contains a stream of Java primitive values. It is filled and read sequentially.
- **MapMessage:** contains a set of name-value pairs where names are Strings and values are Java primitive types. The entries can be accessed sequentially by enumerator or randomly by name.
- **TextMessage:** contains a `java.lang.String`.
- **ObjectMessage:** contains a Serializable Java object. If a collection of Java objects is needed, one of the collection classes provided in JDK 1.2 can be used.
- **BytesMessage:** contains a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

Although JMS allows the use of message properties with byte messages, they are typically not used, since the inclusion of properties may affect the format.

2.1.7.4 Sessions restrictions

According to the JMS specification [24], a session is designed for serial use by one thread at a time. A JMS Session is a single threaded context for producing and consuming messages. Consequently, if multiple threads simultaneously access a session or one of its consumers or producers the resulting behaviour is undefined. In addition, if multiple asynchronous consumers exist on a session, messages will be delivered to them in series and not in parallel. The only exception to this occurs during the orderly shutdown of the session or its connection.

It is erroneous for a client to use a thread of control to attempt to synchronously receive a message if there is already a client thread of control waiting to receive a message in the same session.

If a client desires to have one thread producing messages while others consume them, the client should use a separate session for its producing thread. Once a connection has been started, all its sessions with a registered message listener are dedicated to the thread of control that delivers messages to them.

To take advantage of multiple threads with JMS, use multiple sessions. A client may create multiple sessions. Each session is an independent producer and consumer of messages. For Publisher/Subscriber, if two sessions each have a TopicSubscriber that subscribes to the same Topic, each subscriber is given each message. Delivery to one subscriber does not block if the other gets behind. On the other hand, JMS does not specify the semantics of concurrent QueueReceivers for the same Queue; however, JMS does not prohibit a provider from supporting this. Therefore, message delivery to multiple QueueReceivers will depend on the JMS provider's implementation. Applications that depend on delivery to multiple QueueReceivers are not portable.

2.1.7.5 QueueBrowsers

A client uses a QueueBrowser object to look at messages on a queue without removing them [26]. This might be useful when we want to use existing queues in the target system and we don't want to consume those events, so the system keeps working as if nothing happened. Nevertheless, there is an important catch, which is the fact that every time the client goes offline, all messages that arrive are consumed by the target system's queue listeners and the messages are forever lost. In addition there is no guarantee in the specification that the QueueBrowser listeners are notified first than the system queue consumers, which might cause messages to be missed. Due to all the solution is to create client dedicated queues and tell the target system to insert a copy of event messages in this new queue.

2.2 Architecture and Platforms

The diagram shown on Figure 2.5 represents a high level architecture to support the BAM solution to be developed. First of all, since JBoss AS is the application server being used in

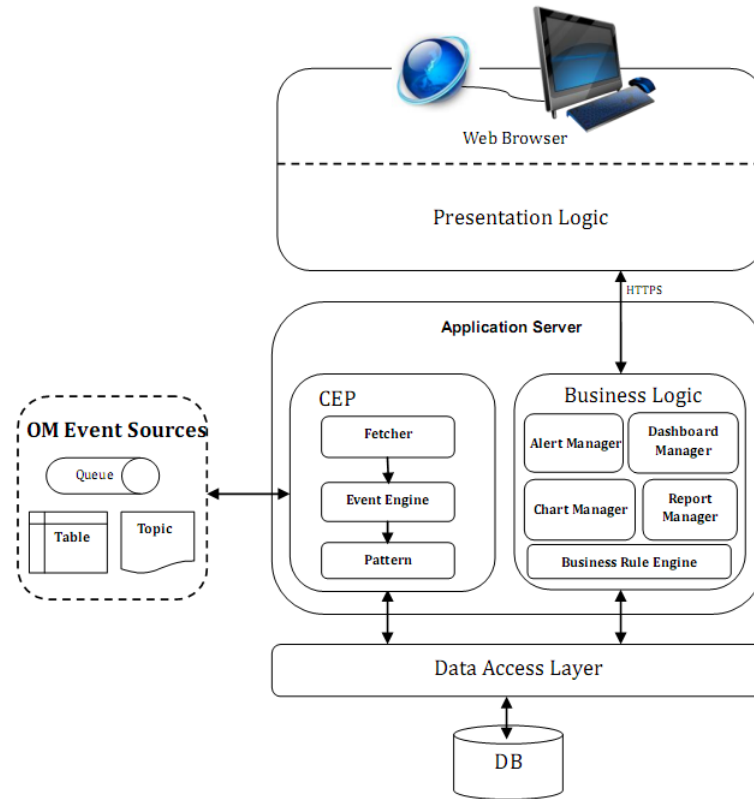


Figure 2.5: The first BAM solution architecture

the OMS development and for a matter of coherence, it will be also the chosen AS for this BAM solution. This architecture is based on all information searched and gathered on chapter 1. Through this architecture it is possible to identify the three main modules that constitute a BAM solution and that require a deeper study. Those modules are: the event engine, which is responsible for fetching events from the OMS sources, the web application presentation, using dashboards and intuitive User Interface (UI), and finally the web application business integration, which is responsible for all the application business behaviour.

2.2.1 Complex Event Processing Platforms

This section is about to explore the possible solutions for the project's CEP component shown in 2.6.

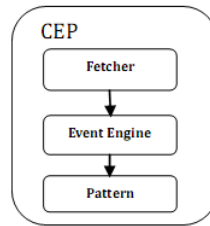


Figure 2.6: Solution’s CEP solution component architecture.

2.2.1.1 Esper Tech

Esper is an Event Stream Processing (ESP) and event correlation engine (CEP, Complex Event Processing), available for Java as Esper, and for .NET as NEsper. Esper is designed to build CEP and ESP applications. Esper is an open-source software.

Complex Event Processing, or CEP, is a technology to process events and discover complex patterns among multiple streams of event data. ESP stands for Event Stream Processing and deals with the task of processing multiple streams of event data with the goal of identifying the meaningful events within those streams, and deriving meaningful information from them.

Esper and NEsper enable the development of applications that process large volumes of incoming messages or events. Esper and NEsper filter and analyze events in various ways, and respond to conditions of interest in real-time.

Targeted to real-time Event Driven Architectures (EDA), Esper is capable of triggering custom actions written as Plain Old Java Objects (POJO) when event conditions occur among event streams. It is designed for high-volume event correlation where millions of events coming in would make it impossible to store them all to later query them using classical database architecture.

Esper’s POJO based programming model and core API enable an existing application to be enriched with Event Stream Intelligence (in Figure 2.7) using some lightweight containers, object oriented programming techniques and XML document management. Esper can be embeddable into any Java process, JEE application server or Java-based Enterprise Service Bus.

The Esper engine has been developed to address the requirements of applications that analyze and react to events such as:

- **Business process management and automation (process monitoring, BAM, re-**

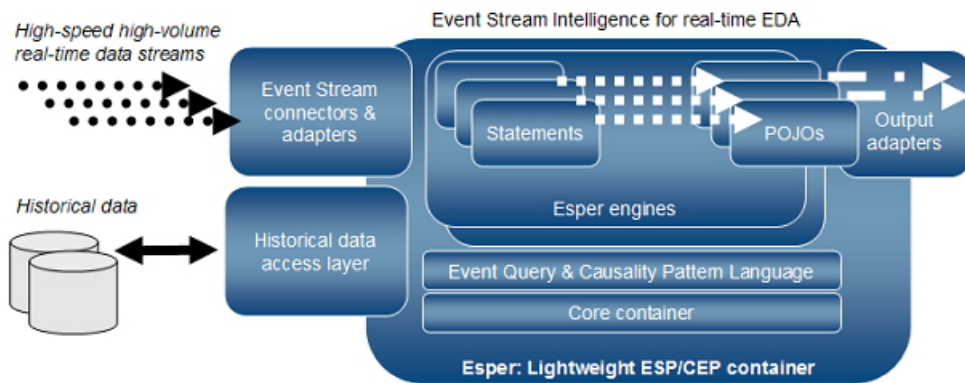


Figure 2.7: Esper's event stream intelligence;
Source: <http://www.espertech.com/products/esper.php>

porting exceptions, operational intelligence);

- Finance (algorithmic trading, fraud detection, risk management);
- Network and application monitoring (intrusion detection, Service Level Agreement monitoring);
- Sensor network applications such as Radio Frequency Identification (RFID) reading, scheduling and control of fabrication lines, air traffic, etc.

Esper Engine

Esper offers an event pattern language to specify expression-based event pattern matching. Underlying the pattern matching engine is a state machine implementation. This method of event processing matches expected sequences of presence or absence of events or combinations of events. It includes time-based correlation of events.

Esper also offers event stream queries that address the event stream analysis requirements of CEP applications. Event stream queries provide the windows, aggregation, joining and analysis functions for use with streams of events. These queries follow the EPL syntax. EPL has been designed for similarity with the SQL query language.

Esper provides a rich EPL which allows expressing rich event conditions, correlation, and joins possibly over sliding time windows as shown in Figure 2.8, thus minimising the development effort required to set up a system that can react to complex situations. It also includes pattern semantics to express complex temporal causality among events.

A listener class, which is basically a POJO, will then be called by the engine when the EPL

condition is matched as events flow in. The EPL enables to express complex matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation and sorting. Events can be represented as JavaBean classes, legacy Java classes,

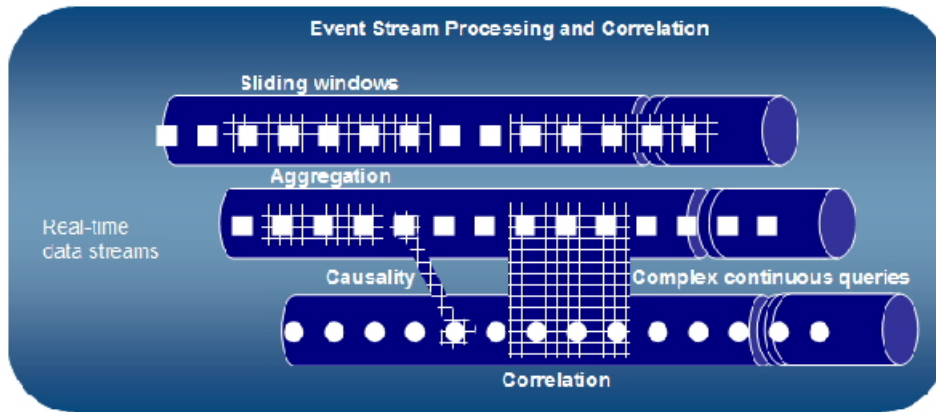


Figure 2.8: Esper's event stream processing and correlation;
Source: <http://www.espertech.com/products/esper.php>

XML document or `java.util.Map`, which promotes reuse of existing systems acting as messages publishers.

Performance

According to Esper's documentation [27], Esper exceeds over 500.000 event/s on a dual CPU 2GHz Intel based hardware, with engine latency below 3 microseconds average (below 10us with more than 99% predictability) on a Volume-Weighted Average Price (VWAP) benchmark with 1000 statements registered in the system - this tops at 70 Mbit/s at 85% CPU usage. Esper also demonstrates linear scalability from 100.000 to 500.000 event/s on this hardware, with consistent results across different statements.

Other tests demonstrate equivalent performance results (straight through processing, match all, match none, no statement registered, VWAP with time based window or length based windows).

Tests on a laptop demonstrated about 5x time less performance - that is between 70 000 event/s and 200 000 event/s.

Main Features

- Support for both the listener (push/subscription) API and the consumer (pull/receive) API for querying results;
- JMS input and output adapter based on Spring JMS templates;
- Time windows - A time window is a moving window extending to the specified time interval into the past based on the system time. Time windows allow to limit the number of events considered by a query;
- Relational database access via SQL-query joins with event streams;
- Multithreaded sends of events into an engine.

Java - Known Limitations

Esper requires a Java Virtual Machine version 5.0 runtime, or above. Esper will not work with JavaVM versions 1.4.2 or below.

Licence

Esper is open-source software available under the GNU General Public License (GPL).

"software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things". [28]

2.2.1.2 PION

Atomic Labs provide an open source Pion Platform community edition. It is a robust and configurable application for data analytics, including a server for data acquisition and transformation. Pion is a real-time platform which enables both simple and complex event processing using pipelines of interconnected plug-ins, called Reactors.

Pion's building-blocks architecture allows to gather data from a wide variety of sources, filter out

the parts you don't need, and then keep the useful information. Additionally, Pion allows you to transform the data according to rules you specify, and you can store the data for compliance purposes or later review. Some features are only available on Pion's paid enterprise edition.

Main Features

Pion might be implemented in C++ but is not available for Java or .Net. It allows integration with Omniture (<http://www.omniture.com/en/>), Webtrends (<http://www.webtrends.com/>), Google Analytics (<http://www.google.com/analytics/>) and Unica (<http://www.unica.com/>). Collects information from relational databases (Oracle, DB2, MS-SQL, MySQL, Informix, Sybase, SQLite, ODBC, etc). Pion pulls in customer information from Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) systems, connects to back end log files and provides a merged stream of data from different sources. Pion is built to be used without requiring a great amount of technical savvy. The application interface is browser-based, and uses drag-and-drop items and menus.

To better understand Pion, it is necessary to define the concept of a Reactor. A Pion Reactor performs a specific task. A Reactor can be thought of as atomic in the sense in that it is the smallest unit of work. Reactors are the basic building blocks which are chained together to accomplish tasks. There are three types of Reactors:

- **Collection Reactors** receive some sort of input (either through monitoring a data stream, an application log, or from a script or another Reactor) and create Events when defined criteria are met.
- **Processing Reactors** process and/or change Events. Processing Reactors may interact with other systems, for example the SQL Reactor interacts with a SQL Database, such as MySQL.
- **Storage Reactors** handle the output of one or more Processing Reactors. Depending on the Storage Reactor, the processed Event may be stored in a database or log file, or it may be sent to another application. Storage Reactors are needed whenever data needs to be persisted for later use for reporting or record keeping. For web analytics to take place, a Pion Workflow must always end with a Storage.

Licence

GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007 *"Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software".* [29]

2.2.1.3 Esper Performance Tests

Esper is a good candidate to support CEP platform. It's performance is obviously an important aspect to consider as well. Although the performance results mentioned by Esper are more than acceptable to satisfy the OMS event flow, Esper's benchmark deals with socket channel and not with a JMS messaging system, which, as mentioned before, is a potential way for communication between the OMS and the BAM solution. Asynchronous communication such as this might drop considerably the performance results. For this reason two performance tests were developed using JMS. The test machine is an Intel Core 2Duo CPU 2.5GHz, 4094MB RAM and 6MB memory cache. The first test developed uses a queue and the second uses a topic. Figure 2.9 represents the performance test architecture. Since the JMS Session has the restriction of having a single thread to handle the invocations to the listeners in the session, it is not recommended to block or delay that thread. This way, when the delivering thread arrives the event process work is delivered in the Esper engine. The Esper engine has an inner thread-pool which has been configured to permit multi-thread usage (by default is a single-threaded), as shown in the Listing 2.1, reducing threads execution delay. The Esper engine threads match the events received against 5 pattern statements in order to know which listener to call.

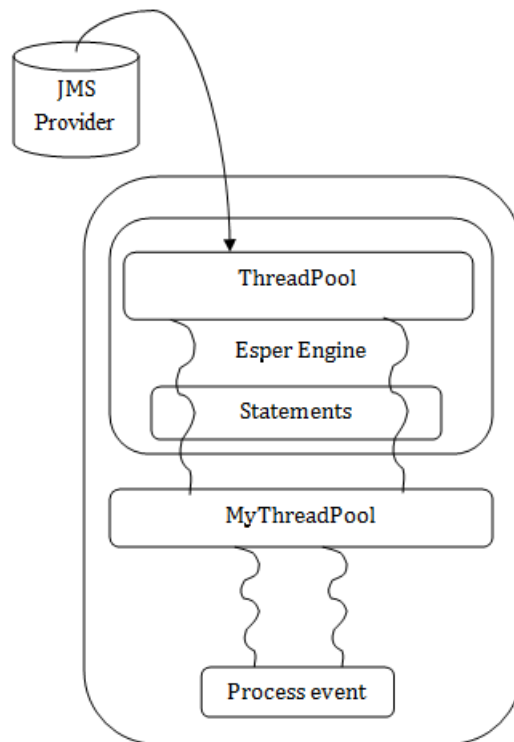


Figure 2.9: Performance test architecture

```

//setthreading
config.getEngineDefaults().getThreading()
    .setThreadPoolInbound(true);
config.getEngineDefaults().getThreading()
    .setThreadPoolInboundNumThreads(2);

```

Listing 2.1: Tuning Esper engine thread-pool.

According to the performance chapter 14 section 14.2.3 in Esper’s documentation [27], it is recommended to process output events asynchronously and not block the Esper engine while an output event is being processed by the listener. For this reason, Esper engine has also been configured to allow multi-threaded output, so every time there is a match and the Esper engine thread is about to process the output event, it delivers the work to a custom thread-pool (which uses the Java ThreadPoolExecutor). By default the engine guarantees that it delivers a statement’s result events to statement listeners in the order in which the result is generated. Esper’s threading, concurrency and other operating characteristics will be deeply explored in the next chapter.

The test results differ by tuning the Esper engine thread-pool and the custom thread-pool.

Since JMS specification recommends using multiple sessions to obtain concurrency, and the first test uses a queue (meaning that each listener remove a message from the queue to be consumed, they all read different messages), creating multiple session to concurrently fetch from the queue has also been taken into consideration. All tests use a single message sender generating event serially. The messages are non-persistent which is the lowest overhead delivery mode because it does not require the message to be logged to a stable storage and guarantees at-most-once, meaning it may lose the message but it must not deliver it twice. The receiver's event processing operation is simply time measuring.

Because, as previously mentioned, the worst case in the OMS event flow is 100.000 events per day, the tests were performed considering that all 100.000 events are serially generated. The following results are the average of a series of test results obtained.

Test 1 - Point-to-Point model

In a real environment the receiver will be processing the senders events in real-time. Nevertheless, since the goal is to measure Esper's engine performance when processing events from a JMS queue, in this test, the 100.000 events are already in the queue when the receiver starts running and processing the events. This way it is possible to ignore the event insertion overhead caused by the sender, and for that reason one should be aware of that overhead in a real case. The following results cover only the JMS event receiving operation and Espers engine processing performance. As seen in Table 2.2 even in the best case it processes $\frac{100.000events}{85seconds} \approx 1176events/s$.

Sessions	Esper engine threads	myThreadPool threads	Time to Process
1	2	2	2m34s
1	2	4	2m43s
1	4	2	2m40s
1	4	4	2m32s
2	2	2	1m29s
2	2	4	1m30s
2	4	2	1m25s
2	4	4	1m29s

Table 2.2: Processing results of 100.000 events using a JMS queue. Each table row shows the results of a test for a specific number of sessions (number of provider's threads), a specific number of threads in Esper engine (frees the provider's threads by receiving events for pattern process) and a specific number of custom threads (to whom the engine delivers the work of processing the events).

So, lets see how 100.000 event processing behaves over time, in terms of process delay, for the best cases in Table 2.2 when using one session and two sessions: As shown in Figure 2.10 the process delay grows almost in a linear way over time. On the other hand, 2.11 shows a small, and nearly constant event process delay over time. This means that, in Figure 2.10, the shown delay is almost entirely due to the fact that there is a single Provider thread delivering messages

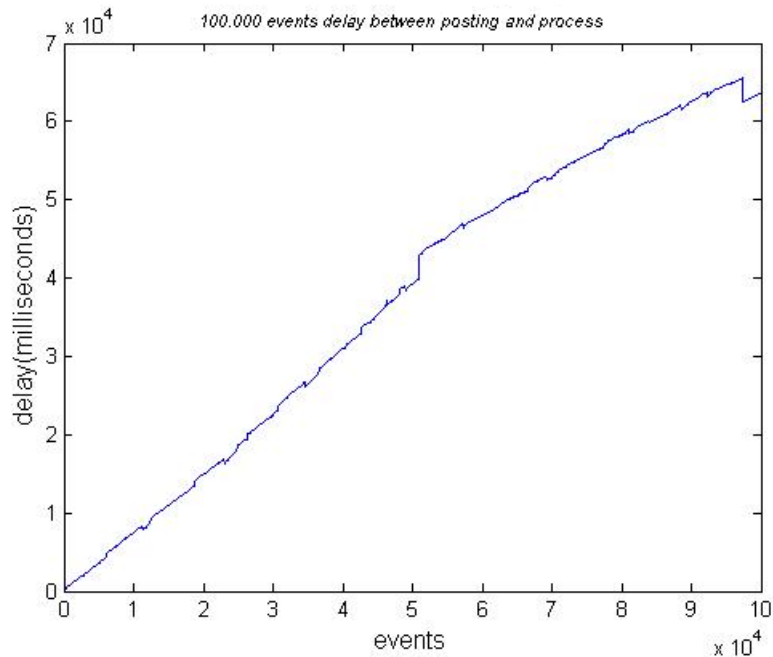


Figure 2.10: Delay growth between posting an event and being processed. Tested for 100.000 events posted on the fly using one session.

to the listener. With two sessions (two threads), in Figure 2.11 the delay is considerably lower, presenting only a few peaks probably due to an OS scheduling to avoid thread starvation.

Even though the performance is much lower than Esper's published results it still clearly fulfils the OMS event flow requirements.

Test 2 - Using a publish/subscribe model

Being a publish/subscribe model test means that messages won't wait in the topic for subscribers to show up and consume them as happens in point-to-point model, unless a durable subscription is used. Even so, durable subscriptions require messages to be stored persistently, like in PTP queues, and when the provider delivers each message to a subscriber he has to remove it from storage. This represents a higher overhead cost, causing higher latency and performance damage. In the previous test the way around was to use multi-sessions (multithreading) to receive messages from the queue, which reduced the latency. For this test that solution does not apply because this model is prepared to deliver messages to multiple receivers which means that all receiver get the same messages from the topic (considering message loss-less scenario). For this reason, in contrast with the PTP test, it is useless to create multiple sessions, for the same receiver, in order to consume messages concurrently and reduce latency (single-threaded).

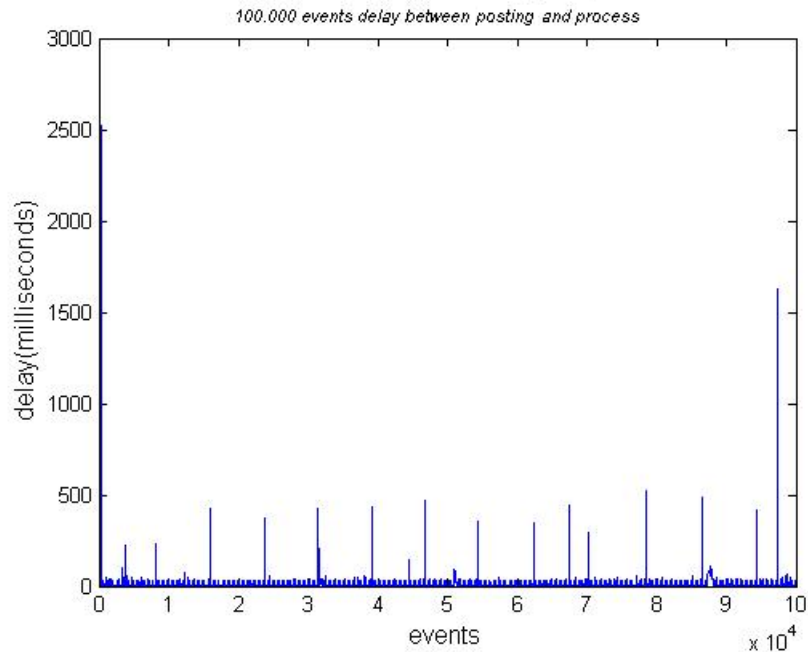


Figure 2.11: Growth delay between posting an event and being processed. Tested for 100.000 events posted on the fly using two sessions.

Nevertheless topics are allowed to work without being durable. This way there is no point in sending 100.000 messages before starting the receivers. Despite the test results contain the delay caused by sending messages to a topic, it is much lower than the one caused by using durable subscriptions in order to avoid latency caused by posting. Table 2.3 presents the test

Esper engine threads	myThreadPool threads	1P 1S	2P 1S	1P 2S	2P 2S	Total
2	2	1m35s	1m38s	2m29s	2m37s	8m20s
4	2	1m38s	1m37s	2m37s	2m29s	8m23s
2	4	1m41s	1m41s	2m32s	2m31s	8m26s
4	4	1m41s	1m42s	2m38s	2m42s	8m44s

Table 2.3: Processing results of 100.000 events using a JMS topic. Each table row shows the results of a set of tests varying the number of publishers(P) and subscribers(S) for a specific number of threads in Esper engine(frees the provider's threads by receiving events for pattern process) and a specific number of custom threads(to whom the engine delivers the work of processing the events). The column 'Total' shows the sum of all the row test results in order to determine the best thread tuning for all possible situations

results. As seen in Table 2.2, the most significant fact is that there is almost a 1 minute difference between tests that use 1 subscriber and 2 subscribers. This means that with subscribers growth the delivery is increasingly slower. Since a subscriber is a session which is a thread consuming from the provider, and there are sender's threads writing at the same time there is a concurrency

in access to the resource. Although this behaviour also happens in a queue by using multiple sessions (in the queue test there was no concurrency since the 100.000 events were previously sent), the PTP model is a one-to-one model and not a many-to-many one, like the publish/subscribe model.

2.2.1.4 Conclusion

Although it allows event processing, Pion CEP Platform lacks the existence of an EPL which is as important to a CEP as data or event processing, since EPL statements are used to derive and aggregate information from one or more streams of events, and to join or merge event streams. Pion's documentation is weaker than Esper's, since Esper's community is much more active. For this reason too, there are a lot more Esper example code available on the Web than Pion's, which makes a huge difference when support is needed for developing a specific solution. According to a 2009 Forrester research, nine CEP platforms were evaluated using 114 criteria and among all of them it is said that,

"EsperTech, the only open source option, is also a Strong Performer... EsperTech is the leading open source CEP provider" [17].

Not even Pion's enterprise edition is mentioned in the research. Again, in a 2008 Senacor Technologies Presentation [21] the only open source BAM approach mentioned used Esper as a CEP solution.

Finally, Esper is able to connect to JMS.

For all reasons above mentioned, including the performance tests, Esper Tech is the chosen CEP platform and will be target of deeper investigation in order to develop the BAM final prototype.

2.2.2 Presentation using Dashboards

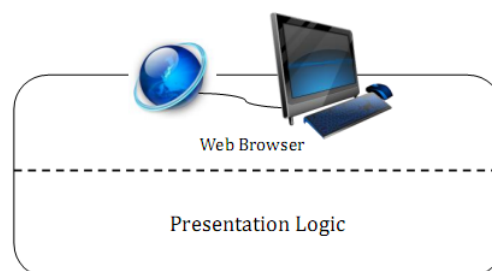


Figure 2.12: Solution's user console architecture.

This section is about to explore the possible solutions for the project's presentation component shown in 2.12.

When dealing with dashboard solutions for a BAM system, charting existence turns out to be essential. Charting provides an intuitive way to assimilate huge amount of data intrinsic meaning, which is a powerful feature when the goal is to control the enterprise business by taking quick decisions based on real-time data. As chart development open source platforms, JFreeChart and Open Flash Chart 2, will be analysed in this section.

2.2.2.1 JFreeChart

JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications.

Features

JFreeChart's extensive feature set includes:

- Interactive features like tooltips etc;
- Supports Swing components, vector graphics file format (like PDF, SVG and EPS) and image files (like JPEG and PNG);
- A consistent and well-documented API, supporting a wide range of chart types;
- A flexible design that is simple to extend. It targets both server-side and client-side applications;
- Support for many output types such as: Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG);

Figure 2.13 shows an example of the final look of a JFreechart pie chart.

Requirements

JFreeChart requires the Java 2 platform (JDK version 1.3 or later). Note that JFreeChart is a class library for use by developers, not an end user application.

Types of charts available

- Pie charts (2D and 3D);
- Bar charts (regular and stacked, with an optional 3D effect);
- Line and area charts;

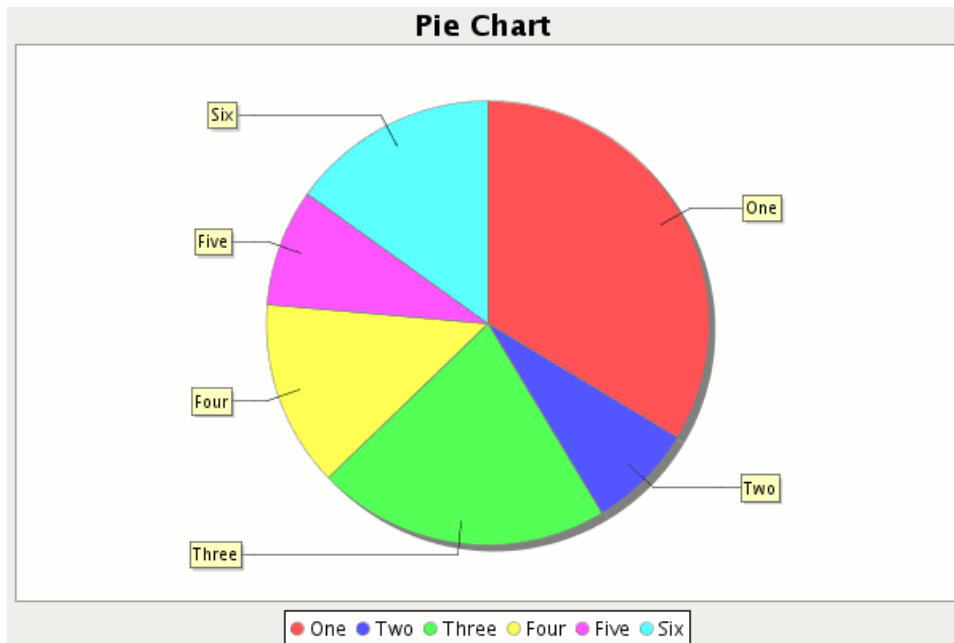


Figure 2.13: JFreechart pie chart example

- Scatter plots and bubble charts;
- Time series, high/low/open/close charts and candle stick charts;
- Combination charts;
- Pareto charts;
- Gantt charts;
- Wind plots, meter charts and symbol charts;
- Wafer map charts.

Refreshment

JFreeChart generates static images to be used over a Web application, such as a web page. JFreeChart also generates maps to be applied over generated static images in order to allow interactivity with the chart information (e.g. drill down). This feature is only available for a few charts. Since the target product is a BAM solution, there will be a permanent necessity to reload the charts for a near real-time monitoring. Considering this need and since were dealing with images, AJAX is a possible way to do it in a web page. Nevertheless, according to JFreechart FAQ,

... the chart is completely repainted for each update, which limits the "frames per second" rate that you can achieve with JFreeChart. Typically, updating once per second is fine, but updating multiple times per second results in high CPU load. If you want to pursue this, do some performance testing with your specific configuration and use cases. [30]

Licence

JFreeChart is free software. It is distributed under the terms of the GNU Lesser [31], which permits use in proprietary applications.

The JFreeChart developer guide is not free.

2.2.2.2 Java API for Open Flash Chart 2

Provides various server side libraries such as PHP, Perl, Python, Ruby, .NET, Google Web Toolkit and JAVA. Open Flash Chart (OFC) provides a poor documentation. OFC code is typically easy to read and generates appealing and interactive flash charts by default. Figure 2.14 shows the life cycle of a chart request with OFC. Types of charts available

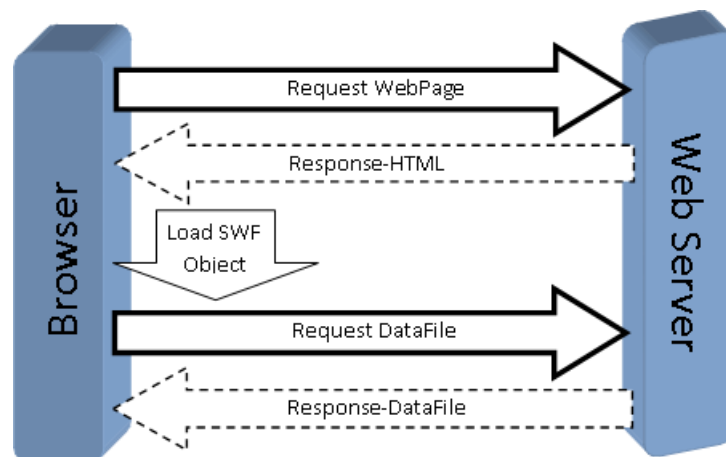


Figure 2.14: OFC chart request life cycle. Adapted from [32]

- Line Charts
- Bar Charts
- Horizontal Bar Chart
- Stacked Bar Chart
- Candle Chart
- Area Charts

- Pie Charts
- Scatter Charts
- Radar Charts

Figure 2.15 shows an example of the final look of an Open Flash Chart bar and line chart.

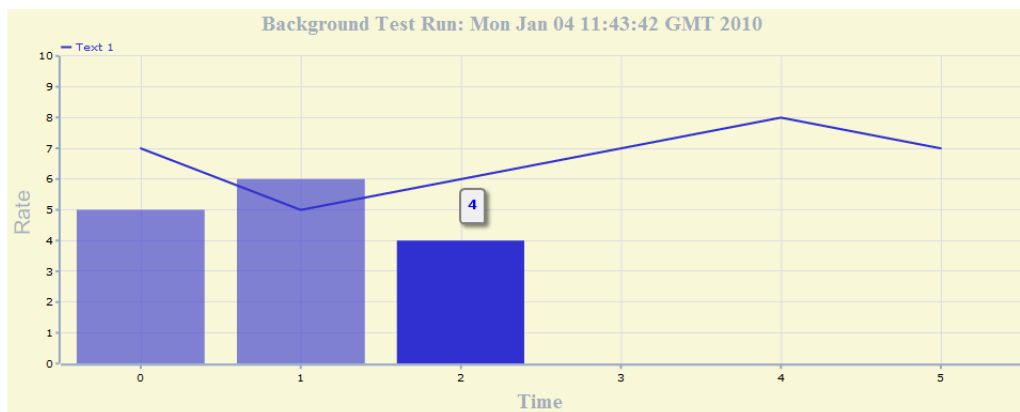


Figure 2.15: Open Flash Chart chart example.

Requirements

Open Flash Chart version 2 requires Adobe Flash Player 9. Nevertheless, according to Adobe, flash content reaches 99% of Internet viewers [33].

Refreshment

Open Flash Chart 2 uses JavaScript Object Notation (JSON) to generate charts. Due to OFC flash characteristic, each chart is bounded to an embedded flash object which is a potential a chart refreshment bottleneck. For that reason OFC provides a way to load a new JSON (Ajax can be used to get the JSON string) into the same client flash object avoiding new flash object creation overhead.

Licence

General Public Licence (LGPL) [28], which permits use in proprietary applications.

Conclusion

Since a goal is to provide attractive dashboards for KPI measuring, the OFC2 platform is a better solution due to its flash component. OFC2 requirements are considered acceptable. Nevertheless,

OFC2 provides a wide range of charts types but doesn't provide all types. For this reason JFreeChart might also come in handy.

2.2.3 Application Business Integration

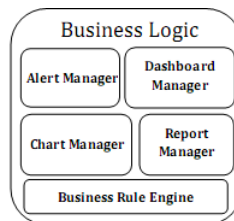


Figure 2.16: Solution's business logic architecture.

This section is about to explore the solution for the project's business logic component shown in 2.16.

JBoss Seam is an open source development platform for building rich Internet applications in Java. As to the Web application framework, JBoss Seam is emerging and is a progressive application framework based on the Java EE platform, that makes writing web-based applications easier by delivering a unified component architecture. Seam builds on the innovative changes in Java EE brought by the Enterprise JavaBeans (EJB) 3 specification. Seam spreads EJB 3's changes across the platform, leveraging more annotations, more configuration by exception and extending the platform, weaving functionality into the JavaServer Faces (JSF) life cycle, and using the unified EL to allow these technologies to communicate. Seam's core mission is to get JSF, Java Persistence API (JPA), and POJO components to work together so that the developer's focus can be placed on building the application, not on integrating unallied technologies.

The Java EE 5 specification incorporates two key component architectures for creating web-based business applications:

- JSF 1.2: Standard presentation framework for the webtier that provides both a user-interface component model and a server-side event model.
- Enterprise JavaBeans (EJB) 3: Standard programming model for creating secure and scalable business components that access transactional resources. Also encompasses the JPA, which defines a standard persistence model for translating data between a relational database and Java entity classes.

2.2.3.1 Why Seam?

Nowadays, there are several Java Web frameworks. Essentially choosing the best one to use has become more difficult. To mention a few other open source choices, there are Struts, Spring, Tapestry, etc. Since Seam is a JBoss framework solution and both this project and the OMS use JBoss AS, it is a natural choice to make. Nevertheless it is imperative to study Seam in order to evaluate if it is useful and brings added value to the project.

EJB components are not intended to be called on directly from JSF. EJB components are scalable, transactional, and secure, but it becomes useless when completely isolated from the web-tier, and from JSF. This isolation makes them of limited use in web-applications because of the complexity involved to integrate them. They are not able to access data stored in any of the web-tier scopes (request, session, and application) and one must be aware of concurrency problems while dealing with EJB components from the web-tier. The Java EE container is not required to serialise access to the same stateful session bean, leaving it up to the developer to take care of this task or catch the exception that can result. The only way the developer can safely use EJB components in the web-tier is by interfacing with an adapter layer.

Because of this, Seam gives EJB 3 components access to web-tier scopes, offers a way to manage the state of these components so that they can be used safely in the web-tier, and even serialises access to stateful components to make concurrency issues a responsibility of the infrastructure and not the developer. Also, there is never a question about thread-safety accessing non-thread-safe resources since Seam handles the scoping properly. Seam also allows JSF UI components to tap into the EJB layer by allowing EJB 3 components to be JSF "backing" beans and action listeners.

Seam reduces the declaration of a component to a single annotation, `@Name`, placed above the class definition. Seam components can take the place of JSF managed beans. Annotations are the central piece of Seam's configuration by exception strategy.

In Seam, configuration by exception goes hand-in-hand with annotations. The annotations give Seam a hint to apply behaviour and Seam tries to assume as much as possible about the declaration by relying on sensible defaults and standard naming conventions.

As shown in Figure 2.17 Seam wires the object with interceptors, wrapping it in a shell known as an object proxy, before handing down the newly created instance. This allows Seam to act as the object's puppeteer during each method call modifying its behaviour. The unified Expression Language (EL) is an expressive syntax used to resolve variables and bind components to properties and methods on JavaBeans. It was introduced to better integrate JSF with JavaServer Pages (JSP). It is also used by JSF to lookup managed beans and other objects stored in web-tier scopes and is the basis for the JSF binding mechanism. EL frees you from having to develop

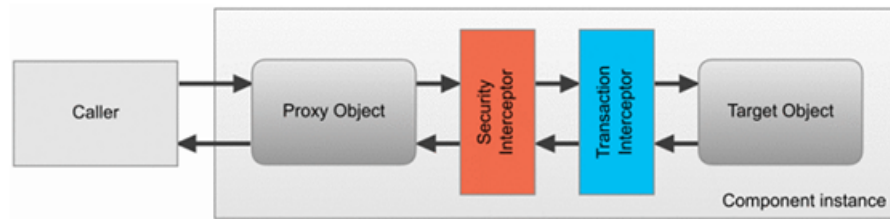


Figure 2.17: Seam Interceptors trap method calls. Source [34]

a custom bridge between the variable contexts of the different technologies in your application. Seam takes advantage of the EL in two ways:

1. Registers a custom EL resolver that is aware of the Seam container, allowing Seam components to be accessed using EL notation from anywhere in the application where the EL is available (which is pretty much everywhere).
2. Allows EL notation to be used in annotations, configuration descriptors, message strings, EJB Query Language (EJBQL) queries, page flow definitions, and even business processes.

Using Facelets as the view handler for JSF in place of JSP is strongly recommended. Facelets is a lightweight view technology that is specifically designed for creating JSF pages. It can accommodate JSF component markup natively and builds the UI component tree from it directly, avoiding the unnecessary JSP tag layer.

All technical characteristics mentioned above makes JBoss Seam one emerging Web framework and an extremely valid option.

Since this project will be working with java, and my experience on coding in java is eclipse IDE based, I'm going to work with Eclipse IDE. JBoss provides JBossTools which is an Eclipse IDE plugin for developing in Seam.

2.2.4 Real-Time Data Storage

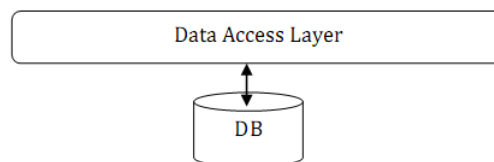


Figure 2.18: Solution's storage architecture.

This section is about to explore the possible solutions for the project's storage component shown in 2.18 in terms of real-time data since it is a critical aspect in BAM solution.

When dealing with a BAM solution, event data storage might be useful for some type of correlations. Since a BAM solution requires support for a high event flow and knowing that for most BAM systems the important event data is the "right now" data (real or near real time data), this means that we need a fast way to store data knowing that data persistence is not a critical aspect. Following the previously mentioned aspects and as mentioned in section 1.3.3, a time series data storage is a good option. Only few time-series data storage products have penetration in BAM solution. Some open source products are: Round Robin Database (RRDtool) and Oracle Berkeley DB Java Edition.

2.2.4.1 Round Robin Database Tool

RRDtool can be remotely controlled through a set of pipes which saves startup time when RRDtool has to do a lot of things quickly. There is also a number of language bindings for RRDtool which allow you to use it directly from Perl, Python, Tcl, PHP, etc. [35]

RRDtool stores your data in Round Robin Databases (RRDs) that maintain a fixed size over time by averaging and compressing data on the fly, based on configured parameters. In RRDtool any data probably fits as long as it is some sort of time-series data.

The data analysis part of RRDtool is based on the ability to quickly generate graphical representations of the data values collected over a definable time period.

Licence

Available under the terms of the GNU General Public License. This means you can do most things you want with this software as long as you do not claim you created the software and don't sell it (or modified version of it) under a license other than the GNU GPL.

2.2.4.2 Oracle Berkeley DB

Oracle Berkeley DB is a family of open source, embeddable databases that allows developers to incorporate within their applications a fast, scalable, transactional database engine. Provides a Direct Persistence Layer (DPL) API for EJB-style POJO persistence. [36]

Berkeley DB is a high performance, scalable and reliable non-relational storage system, designed to be an embeddable database engine, so no separate server is required, and no runtime human administration is needed. [37]

Berkeley DB offers concurrency and transactional storage services, so that multiple threads or processes can operate on the same collection of B-trees and hash tables at the same time without risk of data corruption or loss.

Sometimes an application needs something less than the full power of a relational engine, but more than the low-level services of a file system. In those cases, an Relational Database Management System (RDBMS) offers too many features, is too large and complicated. A file system, by contrast, offers few or none of the integrity, concurrency and performance advantages of a database system. Some applications need transactions and recovery, but not the ability to

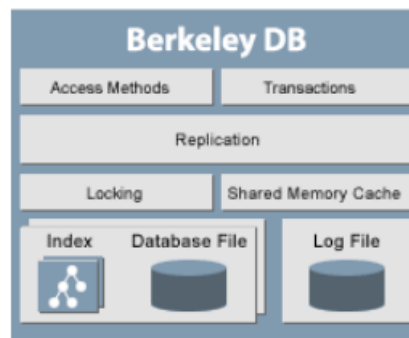


Figure 2.19: Oracle Berkeley DB Architecture. Source [38]

support arbitrary end-user searches against the data they store. For those applications, non relational databases are often a better choice than either the file system or a relational database engine. Berkeley DB combines the file system-style data storage and retrieval with the scalability, reliability and transactional guarantees of a high-end relational system like Oracle Database.

Berkeley DB supports a programming interface to insert, update, retrieve or delete information. Rather than passing SQL strings to the database system for interpretation and execution, developers using Berkeley DB make function or method calls. There is no standalone server and no SQL query tool for working with Berkeley DB databases [38]. It stores any kind of data in any format just like the file system, but provides no easy tools for ad hoc queries. Developers must write code to search a Berkeley DB database.

Relational database engines are heavyweight servers and, by contrast, Oracle Berkeley is a library. Berkeley DB links into the same address space as the application that uses it. It is possible for multiple applications to share a single database. Berkeley DB uses shared memory and operating system primitives for mutual exclusion to be sure that each thread of control cooperates with the others.

Performance

By avoiding context switches and minimising copies, Berkeley DB gets outstanding performance on commodity hardware. For example, performance tests of Berkeley DB published in a recent white paper documented 90,774 sequential inserts per second, using transactions, in a single thread of control. Read throughput, reading a single record at a time, was 466,623 records per second. Using a high performance bulk retrieval interface for large sequential scans of the database, the same system was able to read 13,501,800 records per second.

Licence

According to Free Software Foundation, Berkeley Database License (aka the Sleepycat Software Product License) is a free software license, compatible with the GNU GPL.[39]

Conclusion

Oracle DB is far better documented than RRDTool, and allows programming in Java. Oracle DB presents a good performance without RDBMS complexity.

2.3 Summary

In this chapter the target OMS characteristics were analysed. In order to accomplish the requisites, the BAM solution have to easily integrate with OMS. Due to one or another restriction, from the BAM solution or the OMS, the number of technological options was reduced. This way, Queues turned to be the chosen channel of communication between both systems because of its coupling and easy extension characteristics. JBoss AS is the chosen application server since the OMS is already built on top of that technology.

Further more, some open source platforms and frameworks were studied in order to be able to build a BAM solution like the one discussed in Chapter 1. From the analysis presented in this chapter, the technologies chosen to build the solution were: Open Flash Chart 2, for dashboard display, Esper Tech, for event engine development, and Oracle Berkeley DB, for event data storage. JBoss Seam will be used to build the entire web application, due to its integration through JBoss tools.

Chapter 3

Prototype Development

This chapter describes the adopted solution in order to accomplish an application that matches the explored BAM concept.

Before starting the implementation it is necessary to understand the modularity of this prototype solution. There are main features with different purposes which have to be implemented in separate like the event engine and the website. The event engine is a desktop application which will be running in the application server, fetching all incoming events, processing them and depositing in the right destination, which, in this case is a database. It is a J2EE application developed as an EJB project. The user console is a web application and represents the front end with the user. The event data storage, as discussed in the previous chapter, is a library and a non relational database. It is implemented as an J2EE application and was also developed as an EJB project. EJB is the server-side component architecture and enables the development of distributed, transactional, secure and portable applications. At last, the common business objects are in a separate solution as a class library to be used in the event engine, website and data storage.

It was also necessary to divide the prototype development into different phases in order to ease and reduce its building effort. This way, requirement analysis is necessary to understand where to go from here and the decisions to make.

Since this is a quite wide process, some analysis were necessary to realise the complexity of the problem and to reduce it drastically, identifying the main objectives and the solutions for them. Some analysis like: sequence diagrams, robustness diagrams and use cases brought an entirely new perspective over the project and lead to its solution approach. All analysis can be found in appendix.

3.1 Development

3.1.1 The Architecture

According to information gathered, a BAM solution distinguishes quite well what is the event engine and the user front-end application. The architecture divides both components. Inside the BAM application it is necessary to identify what main modules support the application features mentioned as important to a BAM solution. Among this features we can identify the report generation, alert notification and KPI analysis through charts and dashboard management.

After requirement analysis and platform studies, the previous architecture draft matches with the chosen technologies and turns into Figure 3.1. When comparing with the previous architec-

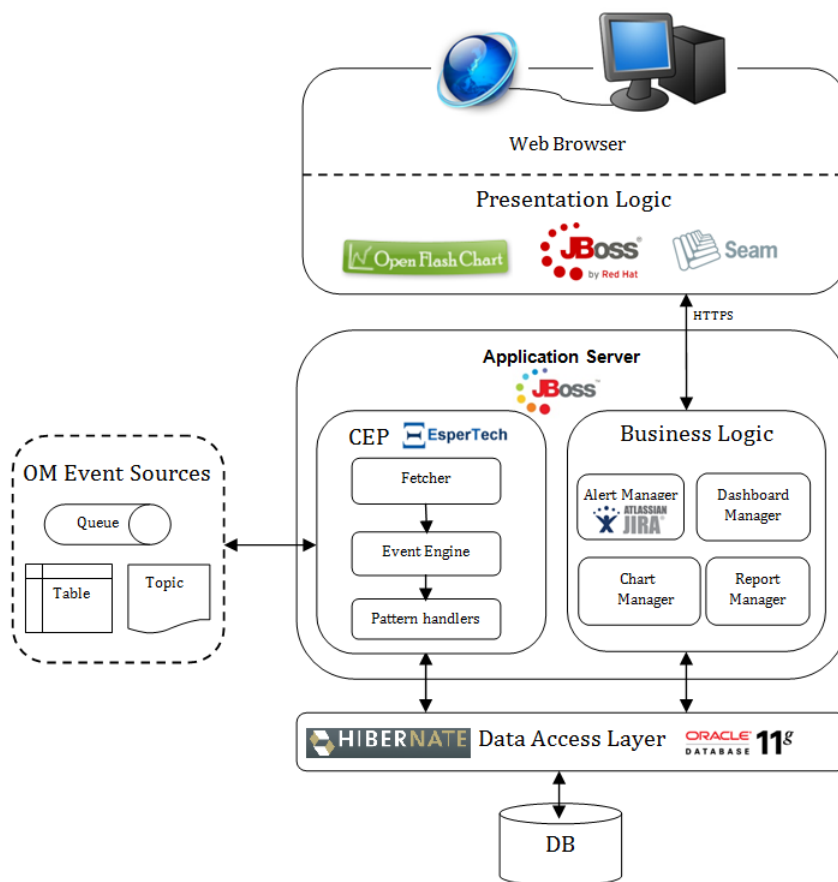


Figure 3.1: Final architecture after platform analysis and prototype requisites.

ture draft, the rule based engine was replaced by a more specialised tool like Esper Tech since it already deals with rules and pattern matching which gives flexibility.

3.1.1.1 Rules Engine versus Complex Event Processing

Although if-then rules might well be the simplest way to understand, rules are only a subset of CEP techniques since there are other ways to do event filtering, correlation and analysis.

Rule language is also distinct from CEP language. Most rule languages are designed to deal with information technology data models, not real-time or time-based event models. For this reason some extensions are necessary before a rule language can be used for CEP.

3.1.1.2 Versions

The used JBoss AS was the 2009-05 release. The used JBoss tools plugin for Eclipse was the 2010-02 release. The used Eclipse IDE is Galileo, 2010-06 release. The used JBoss Seam framework was the 2009-07 release. The used Oracle Berkeley DB was the 4.8.24 release.

3.1.2 The Foundations

Before developing any feature there was the need to prepare the basis for the project to be built on top. After analysing every feature to be developed it became clear that, without the application server installed and prepared there was no way to test what was being developed, like queues being used, since there wasn't either an event gathering destination for the event engine nor event reading sources for the website. For that reason before anything else it is necessary to launch and prepare the application server.

As explained in the previous chapter the chosen application server is JBoss AS. Since the project is to be developed in Java, the Eclipse IDE will be used. JBoss provides a plugin JBoss tools for Eclipse. This plugin allows to work with JBoss technologies like Seam and the application server in a widely used IDE.

3.1.3 The Event Engine

Figure 3.2 represents the event engine diagram class. Before jumping into the event engine implementation it is necessary to understand a little more what happens inside the esper engine.

3.1.3.1 Esper Engine

As mentioned in section 2.2.1.3 there are some suggestions in the Esper specification which should be considered. Esper's specification indicates that by default the engine guarantees that it delivers a statement's result events to statement listeners in the order in which the result is generated.

In the Esper's default configuration the same application thread that invokes any of the sendEvent methods into the engine, will process the event fully [27] and also deliver output events to

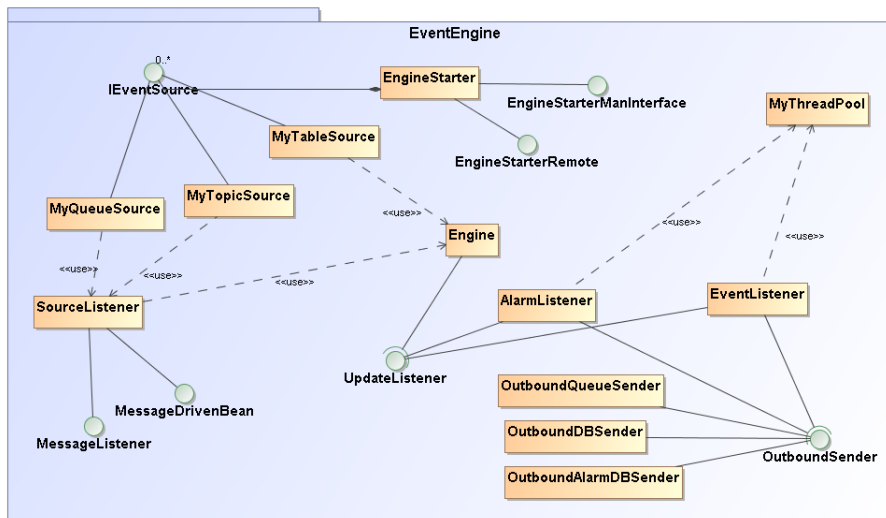


Figure 3.2: Event engine class diagram.

listeners and subscribers.

The Esper engine should not be blocked with output process on the listeners, so the engine provides a threading mechanism to avoid this. With inbound threading an engine places inbound events in a queue for processing by one or more engine managed threads other than the delivering application threads. The engine receives the event and places the event into a queue, allowing the delivering thread to continue and not block while the event is being processed and results are delivered.

As explained previously, in section 2.2.1.3 and shown in Figure 2.9, there is a custom thread-pool to deal with the delivering thread, avoiding this way engine manage threads to block. As expected the engine threading was configured according to the results in section 2.2.1.3.

The event engine is a J2EE project. Every time it is started or restarted it initialises all data sources and alerts defined by the user.

3.1.3.2 Extension Points

The Event Engine has two extension points: data sources and outbound senders.

As shown in Figure 3.2 the core class **Engine** implements **UpdateListener** allowing it to process new events from the known data sources. In this project there are data source classes like, **MyQueueSource** and **MyTopicSource** that fulfill the idea of using JMS, and also **MyTableSource**

for table pooling. All classes implement the interface `EventSource` and its corresponding methods, `start` and `stop`. The `start` method is responsible for creating a connection between the real source and the destination. To be able to add a new data source to the event engine, it is only necessary to create a new class which implements `EventSource`. The new data source implementation connects to the source and sends all information to the destination which ends up on the engine processing threads. Finally a new instance of that class has to be added to the `EngineStater` class so it knows which classes to initialise.

For `MyQueueSource` and `MyTopicSource`, the destination is a listener named `JMSSourceListener`. The `JMSSourceListener` receives every event from the source and delivers it to the event engine to be processed. When dealing with a table source, for example `MyTableSource`, there is no need to indicate a listener since there is already a dedicated thread pooling the table and warning the destination which is the engine.

After processing the event, the engine may deliver the results to an outbound sender. The outbound senders can also be extended by creating a new class and implementing the `OutboundSender` interface. The current senders are the `OutboundDBSender` which sends the events to the database and `OutboundAlarmDBSender` which sends the alarms to the database.

3.1.3.3 Correlation

Esper engine provides, as previously mentioned, an event pattern language to specify expression-based event pattern matching. It includes time-based correlation of events. Lets look at the correlation in Listing 3.1.

```
stmt = "select * from BaseTerminalEvent where type = 'LowPaper' or type = '
      OutOfOrder' ";
      statement = epService.getEPAdministrator().createEPL(stmt);
      statement.addListener(new CheckinProblemListener(outboundSender));
```

Listing 3.1: Esper's event engine EPL example for event correlation .

This statement notifies the listener with all available information for every time it detected that the terminal has low paper or is out of order. Listing 3.2 shows how to count all events that happened, grouped by type for the last 10 minutes. This operation is done every minute and the output is delivered to the listener.

```
stmt = "insert into CountPerType " +
      "select type, count(*) as countPerType " +
      "from BaseTerminalEvent.win:time(10 min) " +
      "group by type " +
      "output all every 1 minutes";
      statement = epService.getEPAdministrator().createEPL(stmt);
      statement.addListener(new CountPerTypeListener(outboundSender));
```

Listing 3.2: Another Esper's event engine EPL example for event correlation .

Every listener knows what to do with the information that is given to him. In this project the listener mainly outputs the events to a persistent storage (Database) so later they can be consumed by the user's application (Website). This way all correlation can be configured in the event engine, removing that responsibility to the user's application and allowing the application to fetch from storage what is expected and already treated. This way it is possible to configure the engine to detect events that match the alerts defined by the user. When a user defines an alert (to be explained further in this chapter) the engine is told to add a new statement at run time that matches with events that should be detected as an alert. Whenever an alert pattern matches, the event engine builds a new event of Alarm type, which is dumped into the events queue, so the target user can be notified according to the notification process chosen (for now are defined email sending and Jira [40] issue). The alarm event is then read by the respective user. These events are also used to generate system behaviour reports. This way the user is not forced to be online to be able to monitor the system.

3.1.4 The User Console

The user console (which is a web application) development was the longest process in all project. In order to support the main features there are core classes for business management shown in Figure 3.3. The JBoss Seam framework already supplies some user management logic. The

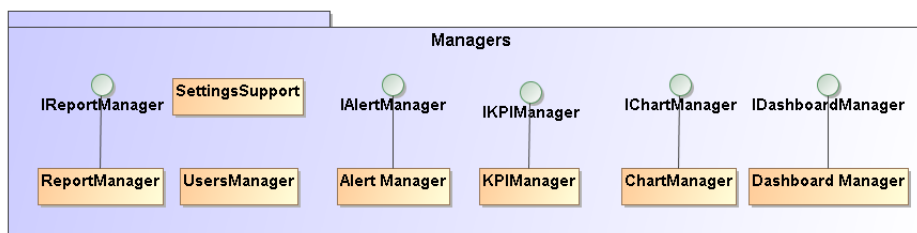


Figure 3.3: Business management classes diagram.

business logic requires some user personal information to be persisted like user configurations in charts and alerts. For this purpose the support classes in Figure 3.4 were created.

3.1.4.1 Event Types

There is a superclass, named `BaseEvent`, for every event in the project. If a new event type is needed its class should extend from `BaseEvent`. Every new event type also need to implement `Serializable` for a matter of storage (to be detailed later in this chapter).

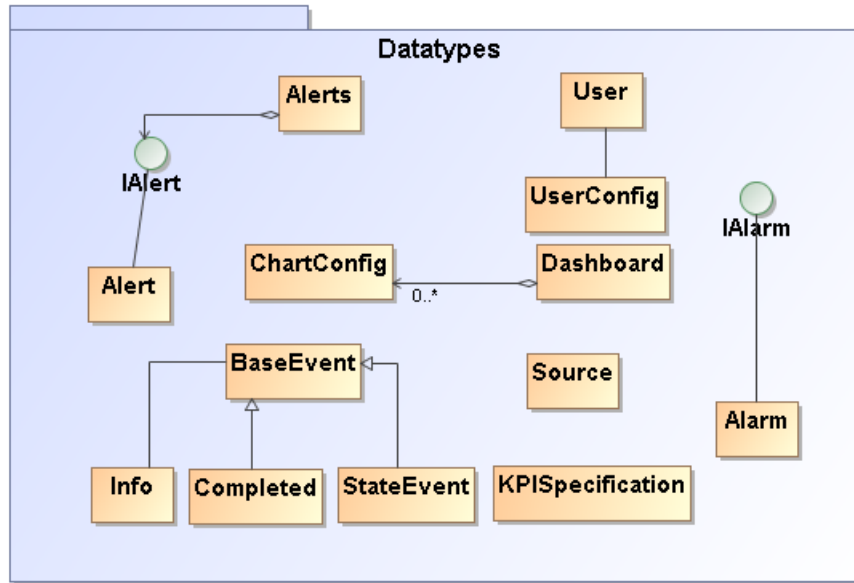


Figure 3.4: Datatype class diagram.

3.1.4.2 Dashboards

Dashboards can be understood as containers of charts monitoring each one its own KPI. This way an user is able to add a chart to an existing dashboard or a new one, allowing to organise the monitoring view. For each chart the user can change its type (line, bar or pie chart are implemented), refresh time, monitored time range and the KPI being monitored, as shown in Figure 3.5. Every KPI implementation, like the implemented KPI Simple in Figure 3.6, for test purpose, has to implement the interface KPI. It is responsible to retrieve the event data it needs from the data storage. Every time a KPI event is retrieved is wrapped into a ChartTimeUnit element. The ChartTimeUnit (shown in Figure 3.7.) class represents a time unit in the chart. This element is then inserted into a ChartBuildingInfo instance that represents all relevant information in the current chart beside the values themselves. All important information that a chart should have should be added to the ChartBuildingInfo. This way the ChartManager is able to work with a "standard" container of values and pass it through as parameter to each chart implementation to create the chart. This allows the user to dynamically change the chart he sees. The user can also change the KPI, time range and refresh time dynamically. All changes are stored in the users configurations.

All charts are displayed using the Java API for Open Flash Chart 2. Every time there is a chart refresh, the client side issues a request and the ChartManager on the server side generates the necessary code. Every chart class uses the OFC2 Java API to build the chart and finally gen-

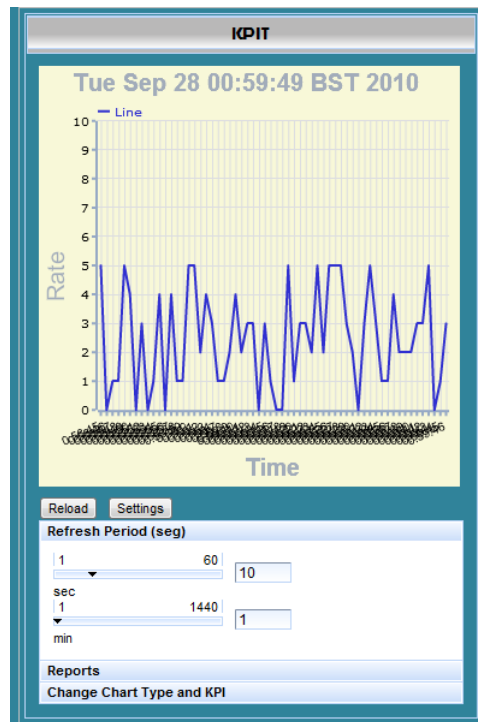


Figure 3.5: Dashboard chart and options

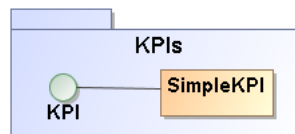


Figure 3.6: KPI support class diagram

erates a JSON string, which is then sent to the client side, to be loaded into the flash embedded object. Figure 3.8 shows the BAM solution dashboard layout.

3.1.4.3 Key Performance Indicators and Data Sources

The user has the opportunity to customise the BAM tool by configuring a KPI and a data source. Starting with the KPI customisation, the user is able to define a name to its new KPI, the name of the events stored that he wants to monitor and the type of KPI (like the implemented class Simple, every new KPI type has to be implemented by the developer) as shown in Figure 3.9. This way the user can monitor different events with the same KPI type. When the user selects a KPI type, the application shows the additional information that specific type of KPI requires and that

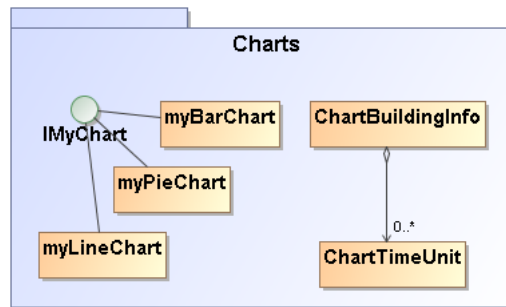


Figure 3.7: Charts support class diagram.

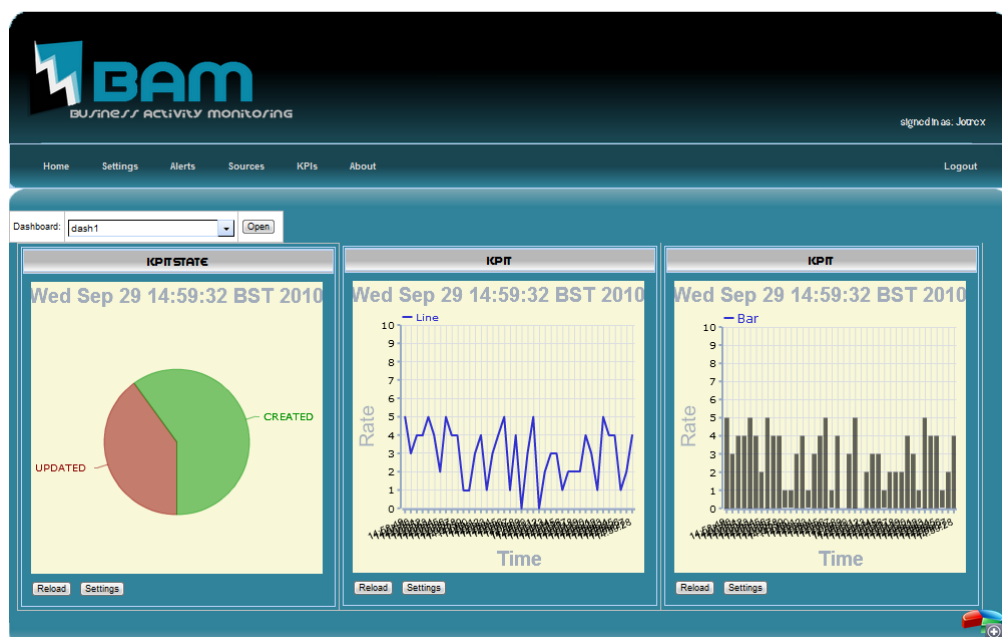
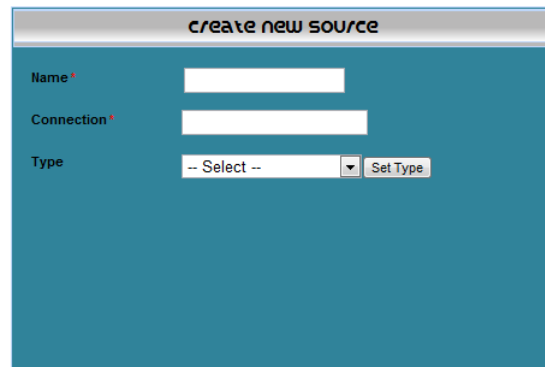


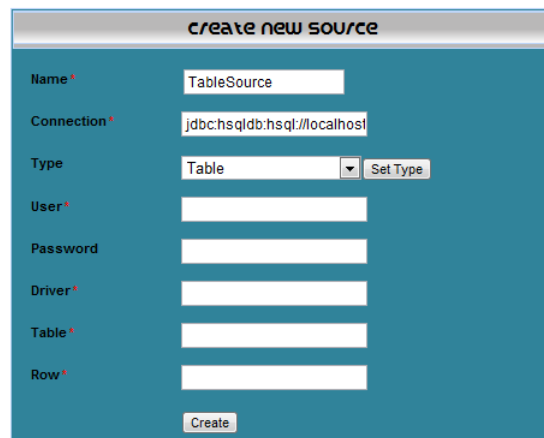
Figure 3.8: BAM solution Dashboard.

are not common to all KPI types, if any. Each KPI, and its additional information, is presented to the user within a form as shown in Figure 3.10. The additional information form is created by the developer that needs to create the XHTML corresponding source. The data sources creation is very similar to KPI. Even so it is relevant to say that the BAM solution already provides JMS queue, JMS topic and table connection as data source types. When the user hits the create button, the event engine is told to connect to a new data source. JMS data source connects to its queue or topic and defines the listener to process every income message. The table data source needs a little more concern. For it to work, it expects that the target database administrator



The screenshot shows a dialog box titled "create new source" with a teal background. It contains three input fields: "Name" with an empty text box, "Connection" with an empty text box, and "Type" with a dropdown menu showing "-- Select --" and a "Set Type" button to its right.

Figure 3.9: Common information in the source creation menu.



The screenshot shows the same "create new source" dialog box, but with additional fields filled in. The "Name" field contains "TableSource", the "Connection" field contains "jdbc:hsqldb:hsq://localhost", and the "Type" dropdown menu is set to "Table". Below these are five more empty text boxes for "User", "Password", "Driver", "Table", and "Row". A "Create" button is located at the bottom of the dialog.

Figure 3.10: Source type additional information in the source creation menu.

creates a trigger to populate a BAM Table called BAM_EVENTS_QUEUE. Every time an event enters the target table in the target database BAM_EVENTS_QUEUE is populated. This way the data source creates a thread to be pooling on the BAM_EVENTS_QUEUE, reading each entry, transform it into a known event, and deliver it to the engine.

3.1.4.4 Alerts

An alert is the concept of notifying someone responsible about a specific problem that requires urgent attention. From this concept we can extract some important information an alert should contain, like a person who should be notified and a way to notify that person. In addition, an alert needs a condition to be fired also known as threshold. As explained in the previous section 3.1.3.3, when the event engine matches the threshold pattern over the incoming events

it treats the happening also as an event, generating a new type called Alarm, shown in Figure 3.4. The Alarm events are stored for later consult by the user console. To support alerts and its configuration, the user has three operations available such as, equal, greater than and lower than as shown in Figure 3.11. All operations implement the IOperation interface which requires a method execute for the threshold detection. A user needs to define a way to notify the target

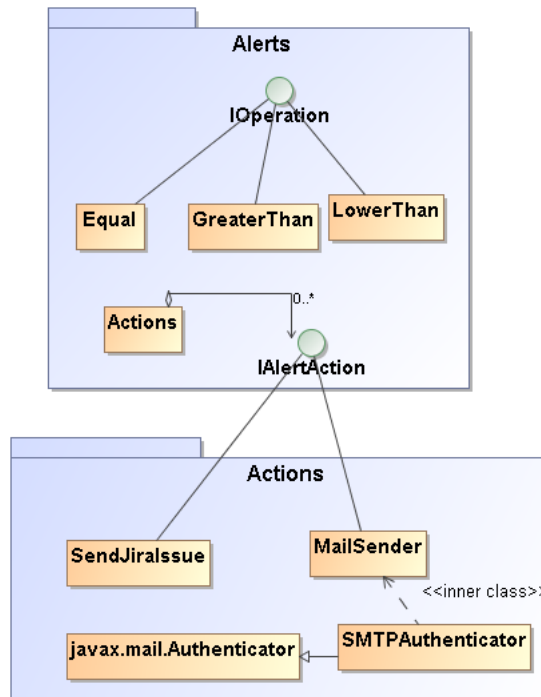


Figure 3.11: Alerts class diagram.

user, and for that the class should implement IAlertAction like EmailSender and SendJiraIssue classes. It requires a way to notify the target user, a description of the action, for combo box filling purposes, and finally a contact validation that checks if the contact given for the specific notification process is valid.

An alert is after all an object that encapsulates not only the action to be executed and the operation to detect the threshold but also the target user, the threshold value, the KPI being measured and the time window in which the value appears a specific number of times, as shown in Figure 3.12. The implemented solution does not grant a non administrator user the right to define alerts whose notification target is not himself, for a matter of security.

After the alert configuration and storage, the event engine is notified so it can define at run-time

Figure 3.12: Alerts menu.

a pattern to be listening for, that matches the defined alert. Every time an alert pattern is detected, the engine informs the target user through the defined action (Jira issue or e-mail). It also issues an Alarm event (previously described) into the queue in order to be read by the EventListener and consequently stored for further consulting. So, whenever an alarmist situation is detected it is treated like an event.

3.1.4.5 Reports

Report generation is an important feature since it allows the user to keep the system behaviour records or even for auditing purposes. Reports usually present detailed information about what happened in a specific system in an historical way. The ReportManger in Figure 3.13 generation

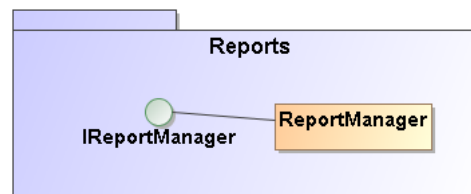


Figure 3.13: Reports support class diagram

is a feature available for each KPI being measured. The user may choose to generate a specific KPI and specify the time period in hours for which he wants to generate the report. A report consists on a pdf document with an initial chart that shows the KPI evolution for the time specified followed by a section that shows the alarms that occurred in the same period of time.

By the end of the generation process, the pdf document is sent to the user's email box. As a future enhancement the report could also be available for download.

The initial chart on the report is generated with the JFreeChart library mentioned in the previous chapter. This solution is necessary because the charts shown in the user's console are generated using a library for flash (Open Flash Chart) and, in this case, a chart image was necessary to post on the report's pdf document.

3.1.5 Data Storage

In Java terms persistence means that we would like the state of our objects to live beyond the scope of the JVM so that the same state is available later.

The data storage is a very important aspect in a BAM System. As explained in chapter 1, a BAM system is a near real-time solution which means that a conventional relational database has too much overhead and complexity to allow it to provide that kind of characteristic. For this reason Oracle Berkeley Database was chosen for the event data access layer solution's support. All data access layer (DAL) classes are shown in Figure 3.14. There are different DALs: the

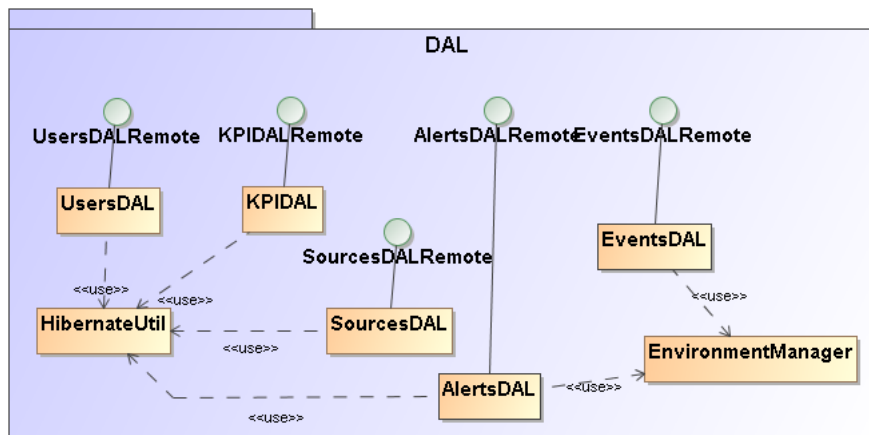


Figure 3.14: Data Access Layer class diagram.

users, the sources, the KPI, the alerts and the events. The last two DAL use the Environment-Manager for a matter of data storing configurations over Berkeley like transactional aspects, locking, duplicates, etc.

All DALs except the events use the relational hypersonic hibernate database to store the data, since there is no advantage in using the Berkeley database because there is no real-time process-

ing required.

3.1.5.1 Oracle Berkeley DB

Even if Oracle Berkeley DB is not a relational Database it supports indexes in order to easily retrieve data from storage. Berkeley DB also provides a way to retrieve data in a date oriented manner, since it is allowed to retrieve data fetched between a begin-date and an end-date. This makes it very useful when dealing with date oriented solutions like BAM.

For this to be possible it is necessary to implement data access object compatible with the Berkeley DB storage mechanism. The Direct Persistence Layer is intended for applications that represent persistent domain objects using Java classes. An entity class is an ordinary Java class that has a primary key and is stored and accessed using a primary index. It may also have any number of secondary keys. Its entities may be accessed by secondary key using a secondary index.

An entity class may be defined with the Entity annotation as shown in Listing 3.3. For each entity class, its primary key may be defined using the PrimaryKey annotation and any number of secondary keys may be defined using the SecondaryKey annotation.

```
@Entity
public class Completed extends BaseEvent
{
    private int _num;

    public Completed(int id, int num)
    {
        super(id);
        _num = num;
    }

    private Completed(){super();}

    public void setNum(int n){_num = n;}
    public int getNum(){return _num;}
}
```

Listing 3.3: The Completed event type defined for integration with Oracle Berkeley DB data storage.

Every field or base class must be annotated as persistent and defined as serializable in order to be compatible with the storing process as shown in Listing 3.4.

```
@Persistent
public class BaseEvent implements Serializable
{
```



```
@PrimaryKey
    private Date time = new Date();

@SecondaryKey( relate=Relationship.MANY_TO_ONE)
    private String type;

    private int id;

    protected BaseEvent() {}

    public BaseEvent(int id)
    {
        this.id = id;
        this.type = this.getClass().getSimpleName();
    }
}
```

Listing 3.4: The BaseEvent class annotated in order to be stored in Berkeley DB.

The SecondaryKey annotation support the many-to-one, one-to-many, many-to-many and one-to-one relationships. It supports foreign key constraints as well.

It is also important to understand that when the developer creates a new event type he needs to modify the existing KPI types classes so they can retrieve data from Berkeley referring to the new type of events. This happens due to the fact that Oracle Berkeley DB does not support SQL queries which forces the developer to programmatically call the right DAL method to retrieve the data instead of simply constructing a SQL query string with a different event name and executing it.

3.1.5.2 Hibernate

Hibernate [41] allows to develop persistent classes following natural Object-oriented idioms including inheritance, polymorphism, association, composition, and the Java collections framework. It requires no interfaces or base classes for persistent classes and enables any class or data structure to be persistent. Hibernate requires no special database tables or fields and generates much of the SQL at system initialization time instead of runtime. It also supports Hibernate Query Language (HQL), Java Persistence Query Language (JPQL), Criteria queries, and "native SQL" queries.

Hibernate uses annotations to ease the configuration of a new object that maps a persistent table just like the example in Listing 3.5. In the Listing we can see an object Source that is mapped to a table named SOURCES, which has a column representing the automatically generated ID, and a column NAME with the name of the source. Annotation referring to columns have to be placed over get methods.

```

@Entity
@Table(name = "SOURCES")
public class Source implements Serializable {

    private static final long serialVersionUID = 6498883703637089447L;
    private int _id;
    private String _name;

    public Source() {}

    public Source(String name) {
        _name = name;
    }

    @Id
    @GeneratedValue
    public int getID() {
        return _id;
    }

    public void setID(int i) {
        _id = i;
    }

    @Column(name = "NAME")
    public String getName() {
        return _name;
    }

    public void setName(String name) {
        _name = name;
    }
}

```

Listing 3.5: Example class annotated in order to be persisted in Hibernate.

After defining every class to be persistent, a file has to be defined for Hibernate configuration like the one shown in Listing 3.6. This file defines the connection to the Hibernate storage and the mapping entities.

```

<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="connection.url">jdbc:hsqldb:hsqldb://localhost:1701</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>

    <property name="show_sql">true</property>
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

```

```

<property name="hibernate.transaction.factory_class">org.hibernate.transaction
    .JTATransactionFactory</property>
<property name="hibernate.transaction.manager_lookup_class">org.hibernate.
    transaction.JBossTransactionManagerLookup</property>

<property name="hibernate.session_factory_name">java:hibernate/
    HibernateFactory</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property name="hibernate.connection.autocommit">false</property>

<!-- Mapping entities -->
<mapping class="com.xpand.User"/>
<mapping class="com.xpand.Alerts"/>
<mapping class="com.xpand.Source"/>
<mapping class="com.xpand.KPISpecification"/>

</session-factory>
</hibernate-configuration>

```

Listing 3.6: Hibernate configuration.

Finally, every time a connection to the Hibernate storage is necessary a new session is created. To ease this task a class named `HibernateUtil` (Listing 3.7) was created. This class configures the Hibernate with the annotation configuration provided in the XML file just described and opens the session.

```

public class HibernateUtil {

    private Logger logger = Logger.getLogger(this.getClass());

    private static final SessionFactory bamSessionFactory;

    static {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            AnnotationConfiguration ac = new AnnotationConfiguration();
            ac.configure("/bam.hibernate.cfg.xml");

            bamSessionFactory = ac.buildSessionFactory();
        } catch (Exception ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    /**
     * @return omSessionFactory
     */
    public static SessionFactory getBAMSessionFactory() {
        return bamSessionFactory;
    }
}

```

```
    }  
  
    public static Session getSession() throws HibernateException {  
  
        return bamSessionFactory.openSession();  
    }  
}
```

Listing 3.7: HibernateUtil Class

For object persistence, a simple **persist(Object)** method is necessary. For object retrieval there are many ways, including SQL queries. Even so, the simplest way might be the example in Listing 3.8.

```
session.beginTransaction();  
    Object src = session.get(Source.class, srcID);  
  
    session.delete(src);  
    session.getTransaction().commit();
```

Listing 3.8: Hibernate object retrieval

3.2 Summary

To summarise it is important to understand the importance of requisites analysis in order to decrease the initial complexity of an unknown domain that a solution like this represents. By the end of this project the importance of these analysis role became notorious and without it the solution would have some inconvenient rollbacks that would certainly delay the project deadline.

Developing the prototype revealed to be an integration challenge due to the different technologies in use. While developing there was also concerns about preparing the solution for later extension and improvement as well as performance aspects since near real time, in a BAM solution, is a main objective.

Even if unable to fully compare with other existing BAM solutions, this solution provides the same main features other solutions present.

Conclusions

The initial investigation became crucial so it turned possible to drill down into the problem's root and to understand what is necessary to fulfil a BAM solution in its essence. That same investigation allowed to discover the state of the art as well as the existing frameworks and technologies available in order to deploy a full open source BAM solution. This way the study of existing material, to build a BAM, was essential and its analysis was necessary to understand where to grab each problem and start giving a solution that fits.

This solution provides all three identified main features that a BAM solution should have. They are, a near real-time business key performance indicators monitoring component, alerts to notify target users about important previously defined business behaviours and finally a detailed report generation with system behaviour important information. The feature set provided by this BAM solution is equivalent to others available in the market. Nevertheless, the solution developed has a value added: it is open source. Since other BAM solutions are not free nor open source it was not possible to compare the solution developed with others. This BAM solution is completely built using a set of carefully chosen open source technologies and provides integration with the target OMS and fulfils its requirements such as the event flow aspect.

Even if the developed solutions can be seen as a fully operational BAM solution, it does not provide drill down features. Nevertheless, the solution is prepared to support drill down characteristics.

The solution developed only deals with past activities. The next generation of BAM systems should be able to infer predictions for future business.

This last two topics could be addressed in future work.

Acronyms

- API - Application Programming Interface;
- BAM - Business Activity Monitoring;
- BPEL - Business Process Execution Language;
- CEP - Complex Event Processing;
- CPU - Central Process Unit;
- CRM - Customer Relationship Management;
- DW - Data Warehouse;
- DAL - Data Access Layer;
- DBMS - Database Management System;
- EDA - Event Driven Architectures;
- EJB - Enterprise JavaBeans;
- EJBQL - EJB Query Language;
- EL - Expression Language;
- EPL - Event Processing Language;
- ERP - Enterprise Resource Planning;
- ISEL - Instituto Superior de Engenharia de Lisboa;
- JBoss AS - JBoss Application Server;
- JBoss ESB - JBoss Enterprise Service Bus;
- jBPM - JBoss Business Process Management;
- JEE - Java Enterprise Edition;
- jPDL - JBoss Process Definition Language;
- JMS - Java Message Service;
- JPA - Java Persistence API;
- JSF - JavaServer Faces;
- JSP - JavaServer Pages;
- KPI - Key Performance Indicator;
- OFC - Open Flash Chart;
- OLAP - Online Analytical Processing;

-
- OMS - Order Management System;
 - POJO - Plain Old Java Object;
 - PTP - Point to Point.
 - RAC - Real Application Clusters;
 - RDBMS - Relational Database Management System;
 - RFID - Radio Frequency Identification;
 - RRD - Round Robin Database;
 - SLA - Service Level Agreement;
 - SOA - Service Oriented Architecture;
 - SOI - Service Oriented Infrastructure;
 - SQL - Structured Query Language;
 - UI - User Interface;
 - UML - Unified Modelling Language;
 - VWAP - Volume-Weighted Average Price;
 - XML - Extensible Markup Language;

References

- [1] D. McCoy, R. Schulte, F. Buytendijk, N. Rayner, and A. Tiedrich, “Business activity monitoring: The promise and reality,” *Gartner*, July 2001.
- [2] C. White, “Is bam alive and well?,” *Information Management Magazine*, September 2005. Retrieved October 22, 2009, available at <http://www.information-management.com/issues/20050901/1035618-1.html>.
- [3] W. W. Eckerson, *Performance Dashboards, Measuring, Monitoring and Managing your Business*. New Jersey, USA: John Wiley & Sons, Inc, 2006.
- [4] D. Nesamoney, “Bam: Event-driven business intelligence for real-time enterprise,” *Information Management Magazine*, March 2004. Retrieved October 29, 2009, available at <http://www.information-management.com/issues/20040301/8177-1.html>.
- [5] W. C., “Operational analytics: Yesterday, today and tomorrow,” *BeyeNETWORK*, November 2007. Retrieved October 23, 2009, available at <http://www.b-eye-network.com/view/6536>.
- [6] D. Chait, “Moving beyond messages per second: Evaluating cep products the right way,” *aleri*. Retrieved November 3, 2009, available at http://www.aleri.com/email/issue2/messages_second.html.
- [7] F. S. Rainer von Ammon, Christoph Emmersberger and C. Wolff, “Event-driven business process management and its practical application taking the example of dhl,” October 2008. Retrieved October 28, 2009, available at http://icep-fis08.fzi.de/papers/iCEP08_8.pdf.
- [8] S. Oberoi, “Introduction to complex event processing & data streams,” *RTI*, August 2007. Retrieved November 3, 2009, available at www.rti.com/docs/SOAWM7_8_oberoi.pdf.
- [9] N. S. group, “Current value table (cvt) reference library,” *National Instruments*, October 2009. Retrieved October 27, 2009, available at <http://zone.ni.com/devzone/cda/epd/p/id/5326#toc0>.

-
- [10] T. Lubinski, "Business activity monitoring: Process control for the enterprise," *ebizq*, February 2008. Retrieved October 26, 2009, available at textttwww.sl.com/pdfs/BusinessActivityMonitoring-tom-lubinski.pdf.
- [11] D. Luckham, "Bam and cep: A marriage of necessity or: Why bam must use cep," *ebizq*, June 2004. Retrieved October 26, 2009, available at <http://www.ebizq.net/topics/cep/features/5102.html>.
- [12] S. Malik, (*Enterprise Dashboards - design and best practices for IT*. New Jersey, USA: John Wiley & Sons, Inc, 2005.
- [13] P. Software, "Apama." Retrieved November 25, 2009, available at <http://web.progress.com/en/apama/index.html>, 2009.
- [14] Aleri, "Coral8 engine." Retrieved November 25, 2009, available at <http://www.aleri.com/products/aleri-cep/coral8-engine>, 2009.
- [15] Oracle, "Oracle business activity monitoring." Retrieved November 25, 2009, available at <http://www.oracle.com/appserver/business-activity-monitoring.html>, 2009.
- [16] S. R. T. Visibility, "Business activity monitoring (bam) solution." Retrieved November 25, 2009, available at <http://www.sl.com/solutions/busactivity.shtml>, 2009.
- [17] M. Gualtieri and J. R. Rymer, "The forrester wave: Complex event processing (cep) platforms, q3 2009," *Forrester*, August 2009. Retrieved December 1, 2009, available at http://www.waterstechnology.com/digital_assets/1261/progress_apama_forrester_report.pdf.
- [18] J. Community, "Jboss as." Retrieved December 21, 2009, available at <http://www.jboss.org/jbossas/>, May 2009.
- [19] O. Feodorov and D. Bevenius, "Jboss esb." Retrieved December 21, 2009, available at <http://community.jboss.org/wiki/JBossESB>, May 2009.
- [20] R. Hat, "Jboss messaging." Retrieved December 21, 2009, available at http://www.redhat.com/docs/manuals/jboss/jboss-eap-4.3/doc/messaging/JBoss_Messaging_User_Guide/html/index.html, 2009.
- [21] C. E. Rainer V. Ammon, Florian Springer and C. Wolff, "Event-driven business process management taking the example of dhl," *Senacor Technologies*, pp. Retrieved December 1, 2009, available at http://icep-fis08.fzi.de/presentations/iCEP08_9_1.ppsx, September 2008.
- [22] JBoss, "Jboss jbpn." Retrieved December 22, 2009, available at <http://www.jboss.com/products/jbpm/>, 2009.

-
- [23] O. Corporation, “Oracle database 11g editions.” Retrieved December 21, 2009, available at http://www.oracle.com/database/product_editions.html, 2009.
- [24] S. Microsystems, “Publish/subscribe messaging with jms topics and using the document object model.” Retrieved November 26, 2009, available at <http://java.sun.com/developer/EJTechTips/2003/tt0415.html>, 2009.
- [25] M. Hapner *et al.*, *Java Message Service Specification - version 1.1*. Sun Microsystems, 2002. available at <http://java.sun.com/products/jms/docs.html>.
- [26] S. Microsystems, “Interface queuebrowser.” Retrieved March 5, 2010, available at <http://java.sun.com/j2ee/1.4/docs/api/javax/jms/QueueBrowser.html>, 2002.
- [27] Esper, *Reference Documentation Version: 3.2.0*. Esper Tech, 2009. available at <http://esper.codehaus.org/nesper/documentation/documentation.html>.
- [28] GNU, “Gnu general public license.” Retrieved December 1, 2009, available at <http://www.gnu.org/licenses/gpl.html>, 2007.
- [29] GNU, “Gnu affero general public licence.” Retrieved December 1, 2009, available at <http://www.gnu.org/licenses/agpl.html>, 2007.
- [30] JFreeChart, “Frequently asked questions (faq).” Retrieved January 05, 2010, available at <http://www.jfree.org/jfreechart/faq.html#FAQ5>, 2009.
- [31] GNU, “Gnu lesser general public license.” Retrieved September 23, 2010, available at <http://www.gnu.org/licenses/lgpl.html>, 2007.
- [32] J. McAdams, “Open flash chart and perl,” YAPC::EU, 2008. Retrieved January 05, 2010, available at <http://www.slideshare.net/joshua.mcadams/open-flash-chart-and-perl-presentation>.
- [33] Adobe, “Flash player penetration.” Retrieved January 05, 2010, available at http://www.adobe.com/products/player_census/flashplayer/, December 2009.
- [34] D. Allen, *Seam in Action*. Manning, 2008.
- [35] T. Oetiker, “Rrdtool overview.” Retrieved March 5, 2010, available at <http://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html>, 2010.
- [36] Oracle, “Oracle berkeley db.” Retrieved March 5, 2010, available at <http://www.oracle.com/us/products/database/berkeley-db/index.html>, 2010.
- [37] Oracle, “Oracle berkeley db java edition vs. apache derby: A performance comparison,” November 2006. Retrieved March 5, 2010, available at <http://www.oracle.com/technology/products/berkeley-db/pdf/je-derby-performance.pdf>.

- [38] Oracle, “A comparison of oracle berkeley db and relational database management systems,” March 2009. Retrieved March 5, 2010, available at <http://www.oracle.com/database/docs/Berkeley-DB-v-Relational.pdf>.
- [39] F. S. Foundation, “Licenses.” Retrieved March 5, 2010, available at <http://www.fsf.org/licensing/licenses/>, 2010.
- [40] Atlassian, “Jira.” Retrieved September 27, 2010, available at <http://www.atlassian.com/software/jira/?gclid=CIjIprntp6QCFY0sDgod7ggS6g>, 2010.
- [41] J. Community, “Hibernate.” Retrieved September 27, 2010, available at <http://www.hibernate.org/>, 2010.
- [42] H. Kochar, “Business activity monitoring and business intelligence,” *ebizq*, December 2005. Retrieved October 23, 2009, available at <http://www.ebizq.net/topics/bam/features/6596.html?&pp=1>.
- [43] D. Luckham, “The beginnings of it insights: Business activity monitoring,” *complexevents*, November 2004. Retrieved October 24, 2009, available at <http://complexevents.com/2004/11/06/the-beginnings-of-it-insight-business-activity-monitoring/>.
- [44] S. Sen, “Business activity monitoring based on action-ready dashboards and response loop,” October 2008. Retrieved October 27, 2009, available at http://icep-fis08.fzi.de/papers/iCEP08_7.pdf.
- [45] W. C., “The extremes of web analytics: From google to bam and business intelligence,” *BeyeNETWORK*, September 2009. Retrieved October 23, 2009, available at <http://www.b-eye-network.com/view/11599>.
- [46] M. Corporation, “What is bam?.” Retrieved October 18, 2009, available at [http://msdn.microsoft.com/en-us/library/aa560139\(BTS.10\).aspx](http://msdn.microsoft.com/en-us/library/aa560139(BTS.10).aspx), 2009.
- [47] S. R. T. Visibility, “Overview.” Retrieved November 16, 2009, available at <http://www.sl.com/products/rtview.shtml>, 2009.
- [48] S. Microsystems, “J2ee - java message service api overview.” Retrieved November 26, 2009, available at <http://java.sun.com/products/jms/overview.html>, 2009.
- [49] E. Tech, “Event stream intelligence: Esper & nesper.” Retrieved November 24, 2009, available at <http://esper.codehaus.org/about/esper/esper.html>, 2009.
- [50] JFreeChart, “Welcome to jfreechart.” Retrieved November 24, 2009, available at <http://www.jfree.org/jfreechart/>, 2009.

- [51] Pion, *Enterprise Edition - User Guide and Reference*. Atomic Labs, 2009.
- [52] F. Marchioni, “Esb service orchestration with jbpn,” 2007. Retrieved November 17, 2009, available at <http://www.mastertheboss.com/en/soa-a-esb/120-esb-service-orchestration-with-jbpm.html>.
- [53] B. Lublinsky, “Using jboss esb and jbpn for implementing vms solutions,” May 2009. Retrieved November 17, 2009, available at <http://www.mastertheboss.com/en/soa-a-esb/120-esb-service-orchestration-with-jbpm.html>.
- [54] JBoss, “jbpn userguide.” Retrieved November 17, 2009, available at http://docs.jboss.com/jbpn/v4/devguide/html_single/, 2007.
- [55] F. S. R. v. Ammon, C. Emmersberger and C. W. Vienna, “Event-driven busines process management taking the example of dhl.” Retrieved November 17, 2009, available at <http://www.citt-online.com/downloads/EDBPM-IP-proposal.ppt>, September 2008.
- [56] S. Microsystems, “Event-driven busines process management taking the example of dhl.” Retrieved December 11, 2009, available at <http://java.sun.com/products/jms/docs.html>, March 2002.
- [57] S. Microsystems, “Java messaging service 1.0.2 api specification.” Retrieved December 11, 2009, available at <http://java.sun.com/products/jms/javadoc-102a/index.html>, March 2002.
- [58] R. Hat, “Seam framework.” Retrieved January 11, 2010, available at <http://seamframework.org/>, 2009.
- [59] Oracle, “Berkeley db.” Retrieved March 5, 2010, available at <http://www.oracle.com/database/berkeley-db/index.html>, 2010.
- [60] T. P. Vincent, “Cep vs business rules.” Retrieved September 19, 2010, available at <http://tibcoblogs.com/cep/2007/10/22/cep-vs-business-rules/>, 2007.

List of Figures

1.1	The basic BAM solution	3
1.2	Integration of BI and business transactions	4
1.3	BAM as a layer over CEP engine	7
1.4	Placing the CVT in context of the application functional blocks	10
1.5	Real-time data flow with relational database as persistent storage	10
1.6	Baseline trend	12
1.7	A museum Dashboard example	15
1.8	Performance dashboard example	16
2.1	JBoss ESB rchitecture	23
2.2	High level perspective of an Order Management System	25
2.3	OMS architecture	26
2.4	JMS message publish/subscribe model	29
2.5	The first BAM solution architecture	32
2.6	Solution's Complex Event Processing component architecture.	33
2.7	Esper's event stream intelligence	34
2.8	Esper's event stream processing and correlation	35
2.9	Performance test architecture	39
2.10	Growth delay between. Tested for 100.000 events using one session.	41
2.11	Growth delay between. Tested for 100.000 events using two sessions.	42
2.12	Solution's user console component architecture.	43
2.13	JFreechart pie chart example.	45
2.14	OFC chart request life cycle.	46
2.15	Open Flash Chart chart example.	47
2.16	Solution's business logic architecture.	48
2.17	Seam Interceptors trap method calls.	50
2.18	Solution's storage architecture.	50
2.19	Oracle Berkeley DB Architecture	52
3.1	Final architecture	55

3.2	Event engine class diagram	57
3.3	Business management classes diagram	59
3.4	Datatype class diagram	60
3.5	Dashboard chart and options	61
3.6	KPI support class diagram	61
3.7	Charts support class diagram	62
3.8	BAM solution Dashboard	62
3.9	Common information in the source creation menu.	63
3.10	Additional information in the source creation menu.	63
3.11	Alerts class diagram	64
3.12	Alerts menu.	65
3.13	Reports support class diagram	65
3.14	Data Access Layer class diagram	66
A.1	Use cases and roles	88
B.1	Sequence for changing the KPI	94
B.2	Sequence for alert creation	95
B.3	Sequence for chart report generation	96
C.1	Project planning	99
C.2	Project planning Gantt map	100

List of Tables

- 1.1 Set of Structured Data 9
- 2.1 Corresponding PTP interfaces with Publisher/Subscriber interfaces. 28
- 2.2 Processing results of 100.000 events using a JMS queue 40
- 2.3 Set of Structured Data 42
- A.1 Additional Specification 93

List of Listings

2.1	Tuning Esper engine thread-pool.	39
3.1	Esper's event engine EPL example for event correlation	58
3.2	Another Esper's event engine EPL example for event correlation	58
3.3	The Completed event type defined for integration with Oracle Berkeley DB data storage.	67
3.4	The BaseEvent class annotated in order to be stored in Berkeley DB.	67
3.5	Example class annotated in order to be persisted in Hibernate.	69
3.6	Hibernate configuration.	69
3.7	HibernateUtil Class	70
3.8	Hibernate object retrieval	71

Appendix A

Use Cases

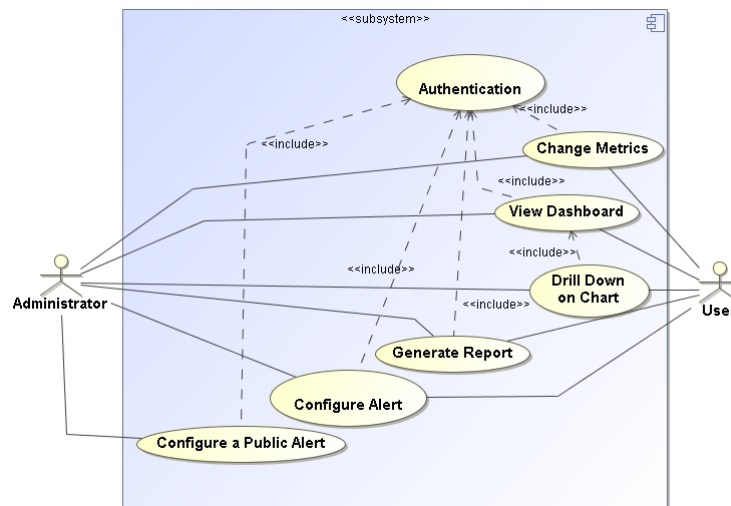


Figure A.1: Use cases and roles

Authentication

Name: Authentication

Description: User Authentication

Actors: User, Administrator

Preconditions:

Postconditions:

Basic flow:

1. The user wants to be authenticated;

2. The user inserts the username (see rule R1);
3. The user inserts the password (see rule R2);
4. The user confirms data inserted;
5. The system validates the username;
6. The system validates the password;
7. The use case ends.

Alternative flow:

1. Triggering conditions: On the main flow step 4 or 5, the system won't validate the inserted data;
 - a. The system informs the user that inserted data is invalid;
 - b. The system returns to its initial state;
 - c. The use case ends.

View Dashboard

Name: View Dashboard

Description: Shows the dashboard with charts and graphically monitored information

Actors: User, Administrator

Preconditions:

Postconditions:

Basic flow:

1. The user wants to view a dashboard;
2. Include use case **Authentication**;
3. The systems shows an initial dashboard;
4. The user chooses from a drop down list a new dashboard;
5. The user clicks on load button;
6. The systems shows the real-time monitored information in a dashboard;
7. The user consults the detailed information;
8. The use case ends.

Drill Down on Chart

Name: Drill Down on Chart

Description: Shows detailed info when drilling down on a chart after clicking it

Actors: User, Administrator

Preconditions:

Postconditions:

Basic flow:

1. The user wants to drill down on a chart;
2. Include use case **View Dashboard**;
3. The user clicks on a clickable chart;
4. The systems shows the detailed information in a new window;
5. The user consults the detailed information;
6. The use case ends.

Generate Report

Name: Generate Report

Description: Generate a detailed report

Actors: User, Administrator

Preconditions:

Postconditions:

Basic flow:

1. The user wants to generate a kpi report;
2. Include use case **Authentication**;
3. The user chooses one kpi chart;
4. The user chooses the time window he wants the report to include (see rule R7);
5. The user confirms on the chosen chart the intention to its report;
6. The use case ends.

Configure Alert

Name: Configure Alert

Description: Configure an alert

Actors: User, Administrator

Preconditions:

Postconditions:

Basic flow:

1. The user wants to configure an alert;
2. Include use case **Authentication**;
3. The user confirms the intention to configure an alert;
4. The system shows an alert form;
5. The user fill in the fields, alert name (see rule R3), alert action (see rule R4) and finally the threshold parameter that triggers the alert (see rule R5);
6. The user is an administrator so he chooses the target user.
7. The user confirms the form;
8. The system validates the form;
9. The system stores the new personal alert;
10. The use case ends.

Alternative flow:

1. Triggering conditions: On the main flow step 6, the user is not an administrator;
 - a. The system automatically knows the current user is the target user;
 - b. This flow returns to the basic flow step 7.
2. Triggering conditions: On the main flow step 8, the system won't validate the form;
 - a. The system informs the user that inserted data is invalid;
 - b. The system returns to the personal alert form;
 - c. The use case ends.

Configure a public Alert

Name: Configure a Public Alerts

Description: Configure a public alert

Actors: Administrator

Preconditions:

Postconditions:

Basic flow:

1. The user wants to configure a public alert;
2. Include use case Authentication;
3. The user confirms the intention to configure a public alert;
4. The system shows a public alert form;
5. The user fill in the fields, alert name, alert action, the group of targeted (see rule R6) users and finally the threshold parameter that triggers the alert;
6. The user confirms the form;
7. The system validates the form;
8. The system stores the new personal alert;
9. The use case ends.

Alternative flow:

1. Triggering conditions: On the main flow step 7, the system won't validate the form;
 - a. The system informs the user that inserted data is invalid;
 - b. The system returns to the public alert form;
 - c. The use case ends.

Change Metrics

Name: Change Metrics

Description: Change a chart metric to be monitored

Actors: Administrator

Preconditions:

Postconditions:

Basic flow:

1. The user wants to change the metrics of a given chart;

2. Include use case **Authentication**;
3. The user clicks on the chosen chart metrics drop down list;
4. The user chooses on metric to be monitored on that specific business area;
5. The system refreshes and shows the new graph monitoring the chosen metric;
6. The user confirms that the metric changed;
7. The use case ends.

Additional Specification

Rule number	Description
R1	The Username must be alphanumeric and with 4 characters at least and 12 maximum.
R2	The Password must be alphanumeric and with 6 characters at least and 25 maximum.
R3	The alert's name must have a maximum length of 15 characters.
R4	The alert action is either an email or jira.
R5	The threshold parameter is either numeric or textual (for example representing a state).
R6	The group of targeted users is the user's role or a group of specific individuals. A group can, therefore, be one person as a minimum boundary.
R7	The time window allowed is between 1 and 24 hours.

Table A.1: Additional Specification

Appendix B

Sequence Diagrams

Sequence for Changing the KPI

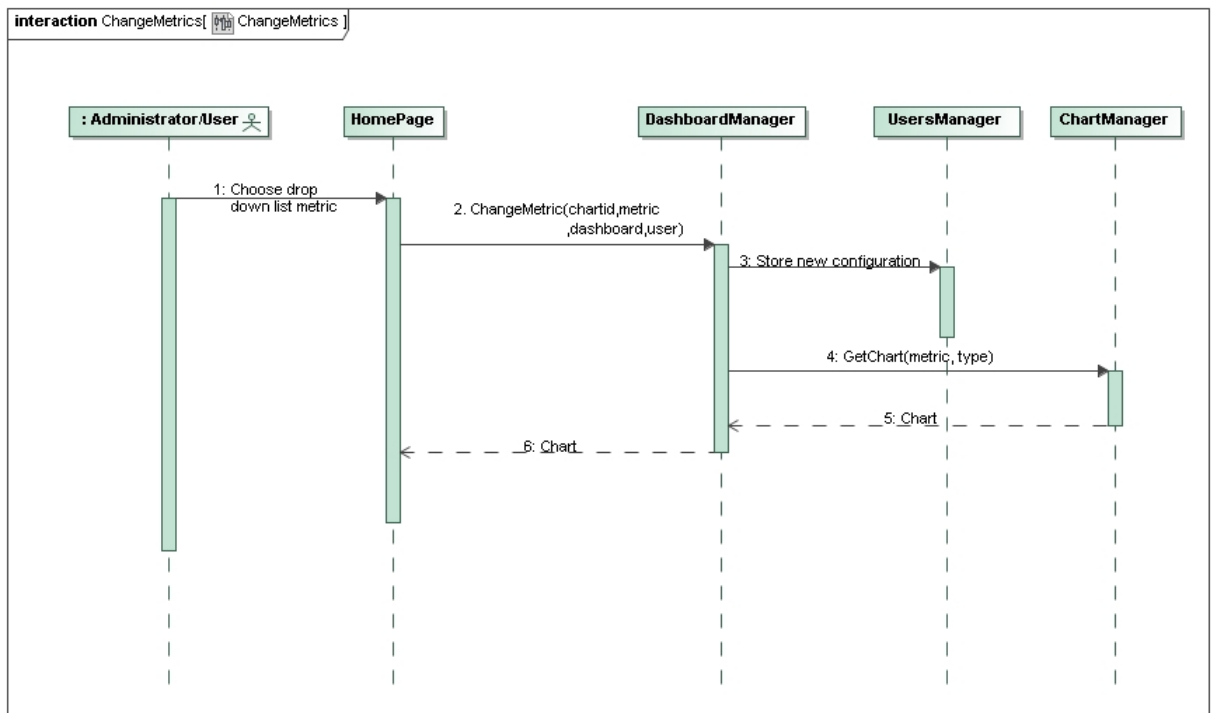


Figure B.1: Sequence for changing the KPI

Sequence for Alert Creation

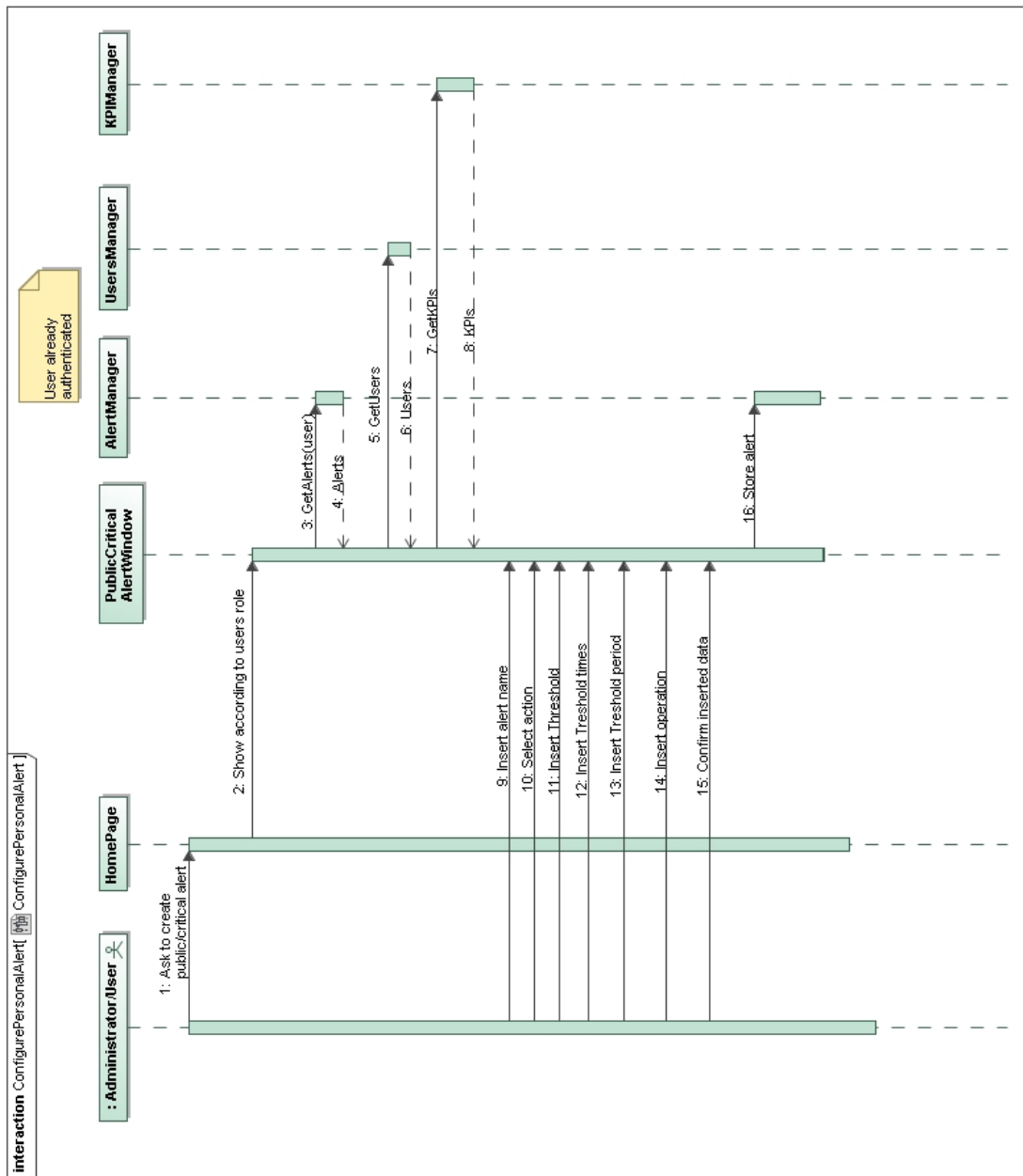


Figure B.2: Sequence for alert creation

Sequence for Chart Report Generation

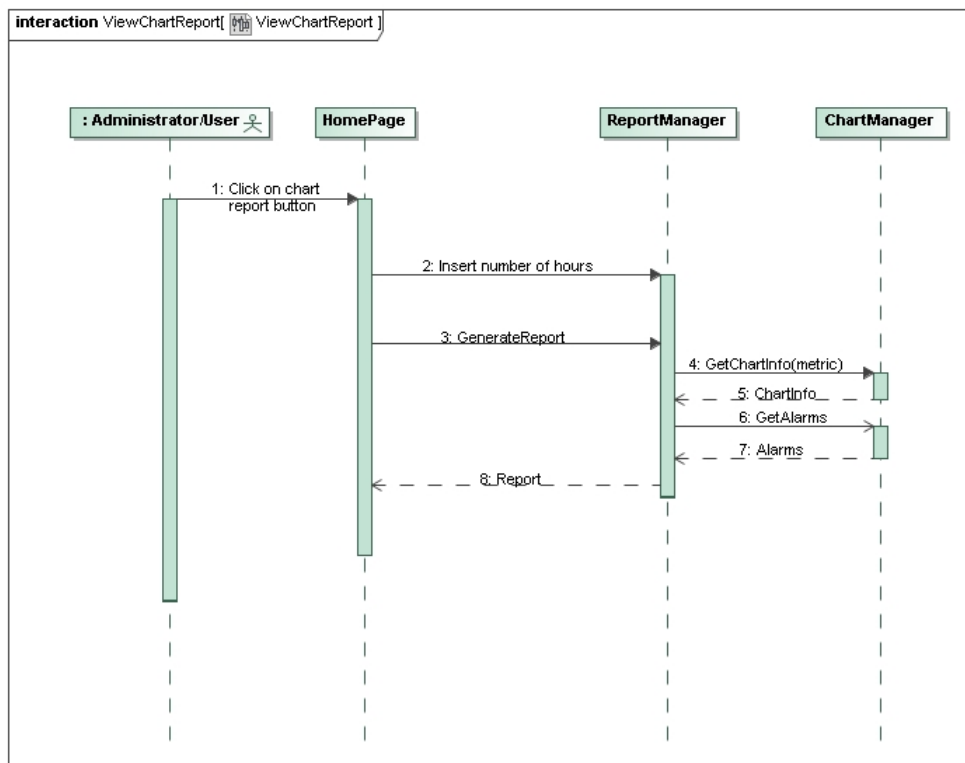


Figure B.3: Sequence for chart report generation

Appendix C

Project Planning Maps

ID	Task Name	Duration	Start	Finish	Predecessors
1	Study, Evaluate & Compare BAM solutions/Frameworks	64 days?	Mon 02-11-09	Thu 28-01-10	
2	Event data gathering	20 days	Mon 02-11-09	Fri 27-11-09	
3	Event data integration	13 days	Fri 20-11-09	Tue 08-12-09	
4	Reporting	10 days	Wed 25-11-09	Tue 08-12-09	
5	Monitoring	12 days	Wed 09-12-09	Thu 24-12-09 4	
6	Alerting	5 days	Thu 17-12-09	Wed 23-12-09 3	
7	Performance analysis	18 days	Wed 09-12-09	Fri 01-01-10 2;3	
8	Licensing Models & Pricing	2 days	Mon 04-01-10	Tue 05-01-10 6;7	
9	Write State-of-the Art documentation	5 days	Wed 06-01-10	Tue 12-01-10 8	
10	Study Jboss Seam	25 days	Fri 25-12-09	Thu 28-01-10	
11	Documentation	64 days?	Mon 02-11-09	Thu 28-01-10	
12					
13	Analyze BAM Scenario	16 days?	Mon 01-02-10	Mon 22-02-10 1	
14	Write Business Case for applying BAM on existing OMS Framework	5 days	Mon 01-02-10	Fri 05-02-10	
15	Document monitoring, reporting and alerting use cases	5 days	Mon 08-02-10	Fri 12-02-10 14	
16	Define entities and events scope	3 days	Mon 15-02-10	Wed 17-02-10 15	
17	Inspect OMS framework for event data collection points	3 days	Thu 18-02-10	Mon 22-02-10 16	
18	Documentation	16 days?	Mon 01-02-10	Mon 22-02-10	
19					
20	Prototype	92 days	Tue 23-02-10	Wed 30-06-10 13	
21	Select Open Source technologies	15 days	Tue 23-02-10	Mon 15-03-10	
22	Implement event data collection	15 days	Tue 16-03-10	Mon 05-04-10 21	
23	Build event data generator for OMS framework	15 days	Mon 22-03-10	Fri 09-04-10 21	
24	Design an interactive real-time monitoring dashboard for selected KPIs	25 days	Tue 06-04-10	Mon 10-05-10 22	
25	Implement reporting over correlated events	20 days	Tue 11-05-10	Mon 07-06-10 24	
26	Integrate with Atlassian JIRA issue tracking platform for alerting	10 days	Mon 19-04-10	Fri 30-04-10 21	
27	Unitary & Component Tests	15 days	Tue 08-06-10	Mon 28-06-10 25	
28	Performance analysis	15 days	Tue 08-06-10	Mon 28-06-10 25	
29	Documentation	92 days	Tue 23-02-10	Wed 30-06-10	

Figure C.1: Project planning

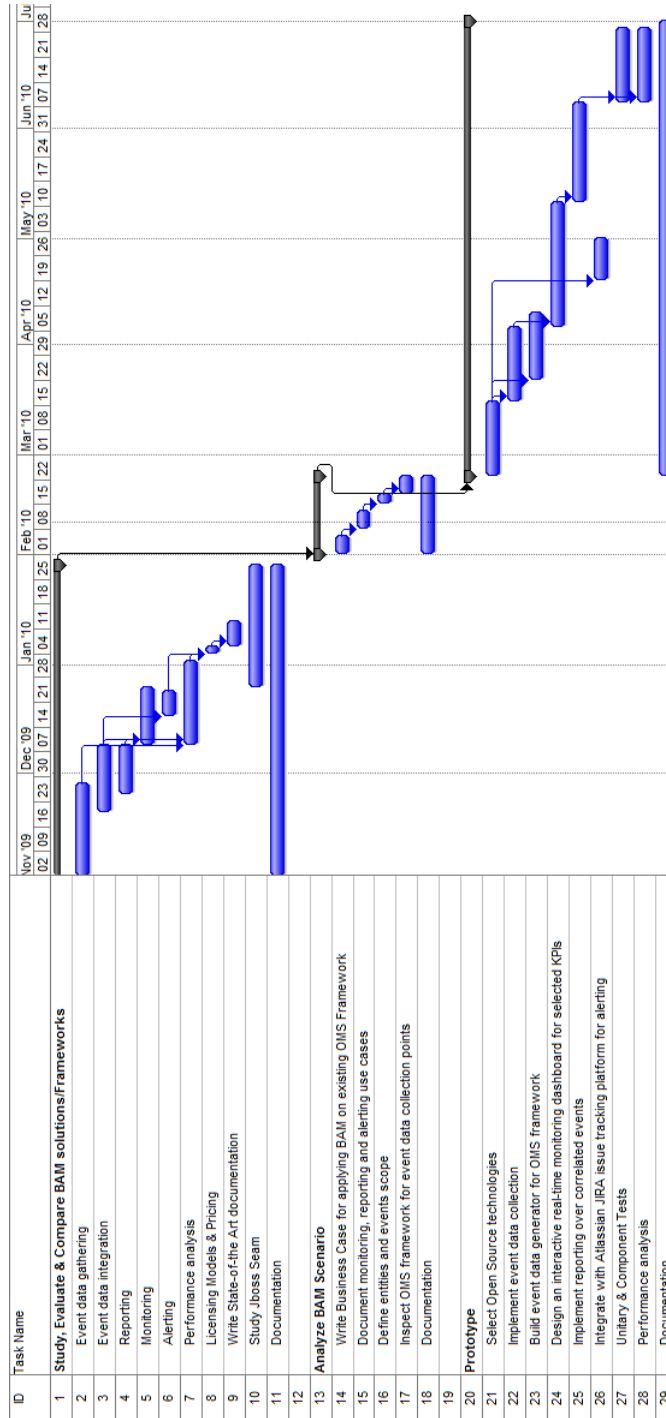


Figure C.2: Project planning Gantt map