University of Applied Sciences Regensburg WS 2007/08

Faculty Information Technology/Mathematics

Event Driven Business Process Management taking the Example of Deutsche Post AG

An evaluation of the Approach of Oracle and the SOPERA Open Source SOA Framework

Diploma Thesis according to

§ 12 APO and § 31 of RaPO

of the University of Applied Sciences Regensburg

Presented by:

Florian Springer

Am Sonnenberg 3

92360 Mühlhausen

AND

Christoph Emmersberger

Immanuel-Kant-Str. 52

84489 Burghausen

Advisor:

Dr. Rainer v. Ammon

Referees:

Prof. Dr. Fritz Jobst

Dr. Rainer v. Ammon

Handed in on:

29.02.2008

Erklärung

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

Ich erkläre hiermit, dass ich die Kapitel 1, 2, 3.1, 3.2, 4, 5, 8.2.2 dieser Diplomarbeit selbständig verfasst, noch nicht anderweitig für andere Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benützt sowie wörtlich und sinngemäße Zitate als solche gekennzeichnet habe.

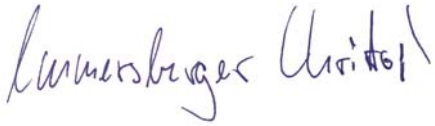
Regensburg, 29. Februar 2008

Florian Springer

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

Ich erkläre hiermit, dass 3.3, 6, 7, 8.1, 8.2.1, 9 dieser Diplomarbeit selbständig verfasst, noch nicht anderweitig für andere Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benützt sowie wörtlich und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, 29. Februar 2008



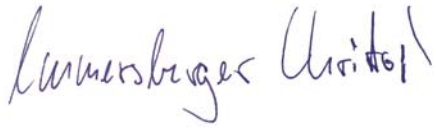
Christoph Emmersberger

Assertion

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

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

Regensburg, 29. February 2008



Florian Springer Christoph Emmersberger

Partition

The particular chapters and paragraphs are written by the authors as follows:

Florian Springer: 1, 2, 3.1, 3.2, 4, 5, 8.2.2

Christoph Emmersberger: 3.3, 6, 7, 8.1, 8.2.1, 9

Contents

1 Introduction and Purpose

This chapter describes the reasons and the motivation of the thesis as well as the objectives of the work. The construction and the outline of the work are presented and justified.

1.1 Motivation and Relevance

Enterprises have to deal with an increasingly fast changing market situation. That is the reason why business processes in companies need to be changed more frequently than a few years ago. Fig. 1 shows the development of cycle times for business processes and delivery times of IT solutions since 1980. The average cycle time for necessary changes of business processes have decreased since the year 1980 by 84 months to six months. The delivery of IT solutions decreased from 30 month to three month. [GREV 04; p. 4]

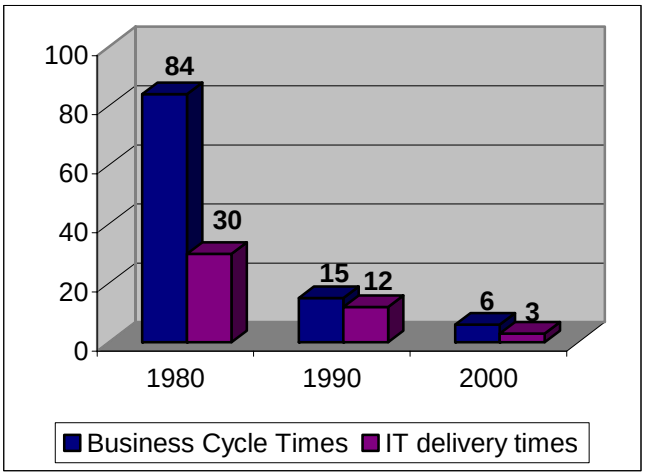


Figure 1: Business cycle time and IT delivery time [GREV 04; p. 4]

Service Oriented Architecture (SOA) should support enterprises to deal with this situation. Deutsche Post was one of the first companies worldwide, which have started to realize a SOA. Deutsche Post decided in 1999 to develop their own Enterprise Service Bus (ESB) because at that time, there was still not an appropriate product on the market. Since 2007, Sopera GmbH was founded to carry on the development and offer that product under the name “SOPERA” on the open source-market. SOPERA includes the ESB and tools for developing services.

Depending on the understanding of the term ESB (para. 2.1.3.6), a standalone ESB does not provide all the components which are necessary to deal with the requirements in today's businesses. For creating and changing business processes quickly (that means daily or weekly, e.g. because of a decision of the marketing department), a component for service orchestration and choreography (para. 5.1.3) is needed, based on a common standard like the Business Process Execution Language (BPEL, para. 5.1.1). During the execution of business processes, the processes and their services also have to be monitored, measured and continuously optimized as shown in fig. 2 from Gartner.

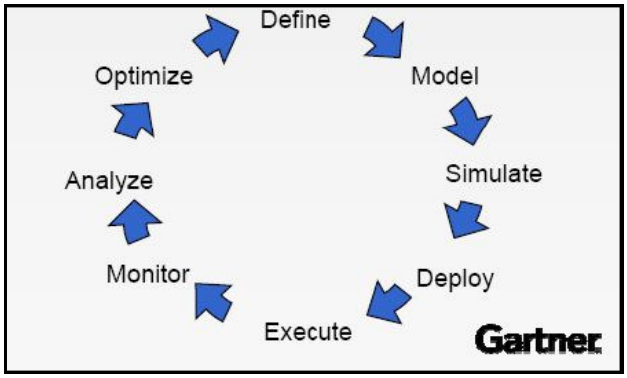


Figure 2: Development lifecycle [REGU 06]

To do this, the concepts of Complex Event Processing (CEP) and Business Activity Monitoring (BAM) have to be used to extend the functionality of the ESB which does not offer that. That is the reason why it was decided to use third party products for such tasks. Oracle and Deutsche Post already had in the past a business relationship and Oracle offers with its big array of products all components, which are complemental and easy to integrate into SOPERA. So, it was decided to see if it is possible to integrate Oracle components into the SOA Framework of Sopera/Deutsche Post and to test the functionality and quality by means of a typical business process of Deutsche Post.

Combining the technologies of Business Process Management (BPM), Business Activity Monitoring (BAM), Complex Event Processing (CEP), Service Oriented Architecture (SOA and Event Driven Architecture (EDA) a new term is just created and will be defined in the upcoming Springer Encyclopedia of Database Systems: Event Driven Business Process Management System [LiTa 08].

The topic of this diploma thesis is the prototypical integration of the Oracle products

- Oracle BPEL (Business Process Management),
- Oracle BAM (Business Activity Monitoring),
- Oracle CEP (Complex Event Processing)

within the SOPERA system environment, with the focus on CEP, which will become a future product of Oracle. For evaluating the capabilities of the components, a business process regarding to shipment, investigation and claim has to be modeled and implemented. Some different approaches are discussed, evaluated and implemented prototypically. The focus of the implementation is to provide events for the purpose of monitoring the business process.

1.2 Content

The following chapters describe the steps of integrating the Oracle products BPEL, CEP and BAM into the SOPERA service environment, as well as a business process of the Deutsche Post, which is used for an exemplified implementation for this thesis. Chap. 2 gives an introduction into SOA, explains the basic architecture of SOA and presents in a next step the SOA approach of SOPERA and the components and the architecture used in this thesis.

The example business process, the events within it, and use cases for simulation purpose are described in chap. 3. Approaches for event generation as well as event stream processing are also discussed. Chap. 4 shows an exemplified implementation of the SOPERA service “Shipment” with the focus of the integration into Oracle BPEL. Extending SOPERA services with the functionality of generating events in an intrusive way as well as in a non-intrusive example is also shown. The integration and the possibility of orchestration SOPERA services with Oracle BPEL, generation of events with BPEL Sensors, as well as a basic introduction into BPEL is in chap. 5.

Chap. 6 introduces the NR component which is responsible for sending events to a defined endpoint, in this case to Oracle CEP and BAM. An introduction in to JMS

which is used for transportation of events, as well how to setup JMS is done in chap. 7.

An introduction into the components of Oracle BAM, setting up data objects, transforming incoming messages as well as creation of a BAM report for analyzing is done in chap. 8. Finally chap. 9 introduces CEP and shows bases on the use cases for simulation and the basic events a complex event pattern for detecting possible errors within the shipment process.

2 Introduction into SOA and the Basic Architecture of the Prototype

This chapter describes the basic concepts of SOA and explains the SOPERA SOA approach.

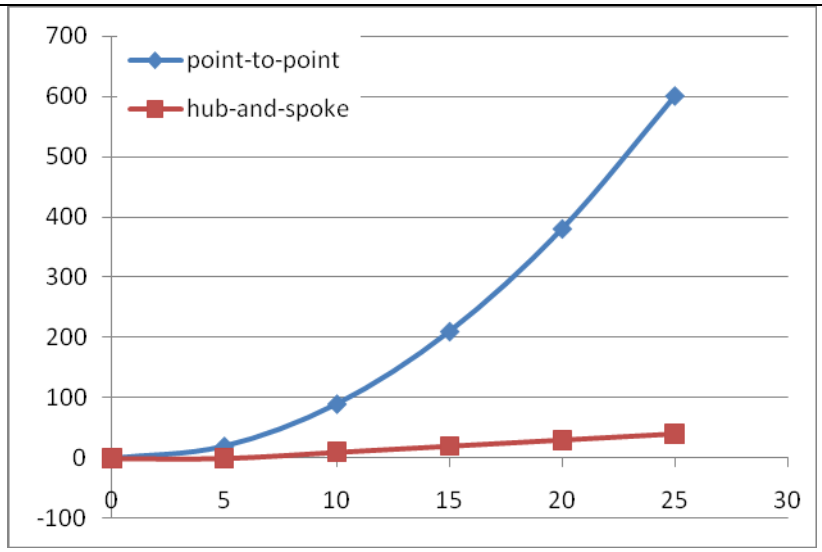
2.1 Introduction into SOA

This paragraph is not a holistic explanation about the Service Oriented Architecture (SOA) it is rather a short explanation about the concept behind SOA, to allow the reader a better understanding of the components and the technology used in this diploma thesis. The open source SOA framework SOPERA provides the base technology, which has to be extended with the features of Oracle BPEL/CEP/BAM.

2.1.1 Necessity of SOA

The necessity of SOA is based on IT systems currently existing requirements and problems within the enterprises. First of all, IT has to handle the fast changing demands of the business, which causes decreasing innovation cycles as explained in para. 1.1.

Other problems in enterprises are heterogeneous information systems, containing a range of different systems, applications and technologies. Integration of these technologies is important, because only integrated information systems are able to deliver business values which allow efficient decision-making. Instant information access and data integrity also requires integrated systems [MATJ 06; p. 12]. The integration of these heterogeneous systems needs the development of interfaces for communication, which has to be done by experts for every needed function. The traditional point-to-point mechanism to do that causes higher costs compared to the hub-and-spoke mechanism which is realized in a SOA by using the ESB as message broker. That is caused by the larger number of interfaces which have to be developed. Fig. 3 compares the two mechanisms and illustrates the increasing number of the developed interfaces. Consequently that issue causes rising costs.




```
point-to-point:  m = n * (n-1);  n = number of functions
hub-and-spoke:   m = n * 2;      m = number of interfaces
```

Figure 3: Point-to-point compared to hub-and-spoke

Because a detailed explanation of all reasons would be beyond the scope, it will be waived at this point.

2.1.2 Definition of SOA

Regarding to [NATI 03], the concept of SOA was described first by Gartner in 1996. Since then, Gartner is referred as the inventor of the term SOA. In that paper Gartner describes not the future software architecture, rather that the technology has to be pushed in the background and the business processes have to be moved in the foreground. So SOA can be seen as an approach of a software concept, with the focus on the business processes. This approach has to handle the increasing demand in changing business requirements of software applications (para. 1.2) to enable enterprises to react quickly on new situations of the market.

One of the research results is that there exists no unique definition of a SOA and every software vendor or company has its own definition. The most often quoted definition comes from OASIS published in the SOA-RM. “(…) Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. (…)” [OASI 06; p. 8, line 128]. Yefim V. Natis -Vice President and Research Director of Gartner Research - defines SOA as “(…) a software architecture that builds a

topology of interfaces, interface implementations and interface calls. SOA is a relationship of services and service consumers, both software modules large enough to represent a complete business function. Service-oriented architecture is about modularity, reuse and agility on behalf of the real-time enterprise (…)" [NATI 03].

In the authors view the most significant definition comes from Bernhard Borges – technical executive and solutions architect at IBM – and his colleagues who combine these two definition. "(…) SOA is the architectural style that supports loosely coupled services to enable business flexibility in an interoperable, technology-agnostic manner. SOA consists of a composite set of business-aligned services that support a flexible and dynamically re-configurable end-to-end business processes realization using interface based service descriptions. (…)" [MATJ 06; p. 12].

As a consequence of these definitions applying SOA to an IT infrastructure means the creation of a system that is able to react quickly and efficiently in fast changing business demands. IT systems have to be justified on the demand of the business processes in a real-time enterprise environment. Achieving this aim, modular and reusable services are recommended to be used. Because these services are provided by different applications, the approach of SOA is also related to the integration, development and maintenance of complex enterprise information systems. All definitions have in common, that none describes exactly how to implement the approach, and which technical standards or concepts have to be used.

2.1.3 Basic Concepts and Architecture in a SOA

This paragraph gives a rough overview about the architectural view of a SOA and describes the basic concepts in a SOA.

2.1.3.1 Service and Interface

As the name already says, the main component within a SOA is a service. Regarding to OASIS a service is "(…) a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by an entity – the service provider – for use by

others (…). A service is accessed by means of a service interface, where the interface comprises the specifics of how to access the underlying capabilities. (…)" [OASI 06; p. 12]. In a SOA, services are able to provide business functionality as well as technology-oriented functionalities. But the focus is to provide business value, be autonomous and hide the details of the implementation. The functionality of a service can be implemented with several programming languages, e.g. C++, Java. Services are called by software entities – the service consumer – which uses its functionality through their public interface. The interface is a contract between the service provider and a service consumer. It is self-describing, platform independent and separate from the implementation. Each interface defines a set of operations. The interface description provides a basis for the implementation, for the service consumer, as well as for the service provider [MATJ 06; p. 12]. Fig. 4 pictures services communicating through its public interface. The internal implementation of the service is oblique.

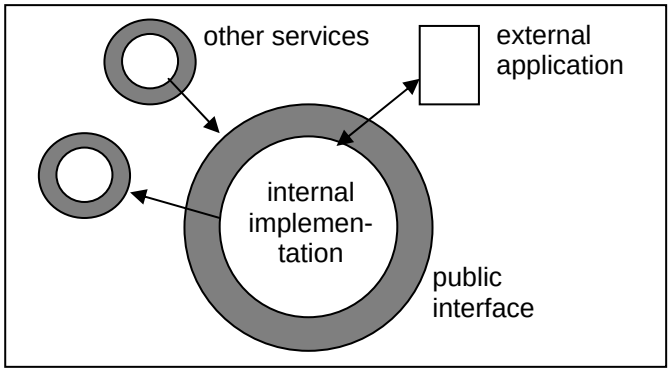


Figure 4: Principles of a service

A concrete implementation in a SOA is called Web service. Regarding to the definition of W3C a Web service is a "(…) software system designed to support interoperable Machine to Machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). (…)". A service consumer interacts "(…) by its description using SOAP messages (…)" with the Web service [W3C 04; p. 4].

2.1.3.2 Service Description

Regarding to W3C, a Web service and its interface is described with the Web Services Description Language (WSDL). It is a XML based language that

provides a model for describing Web services completely that means including its types, bindings, messages and port types.

- <portType> describes the operations which can be performed and the involved messages.
- <messages> define the data elements of an operation
- <types> define the types used by the service by using the XML schema
- <binding> defines the message format and protocol details for each port.

WSDL is a W3C recommendation. The current version 1.2 was renamed into WSDL 2.0.

2.1.3.3 Service Protocol (SOAP)

Web services within a SOA use SOAP for message exchange. It is a lightweight protocol for exchanging structured information in a decentralized, distributed environment based on XML. The messages are independent of any operation system. SOAP does not provide its own protocols or standards, but is based on existing standards, like XML for encoding of invocation requests as well as responses. It uses several internet protocols, e.g. HTTP/HTTPS or SMTP, for communication and thereby provides just a framework for message exchange. It is also not a standard, just a W3C recommendation. Formerly SOAP meant Simple Object Access Protocol, but is not used since version 1.2 as an acrony, because of the confusing meaning [W3C 07].

2.1.3.4 Loose Coupling

Loose coupling of services is achieved through the self-describing interfaces, the exchange of data structures using SOAP, and coarse granulation. Loose coupling is basically defined by Karl Weick [WIKI 07] as “(…) a resilient relationship between two or more systems (…). Reflected to services, loosely coupled services are services that expose only the necessary dependencies and reduce all kinds of artificial dependencies.

This is particularly important when services are changed frequently, as is common in today’s businesses (para. 1.1). Minimal dependencies assure that there is a minimal amount of changes to other services required when a service has to be

modified. This approach improves robustness, makes the system more resilient to changes and allows reuse of services [MATJ 06].

2.1.3.5 Service Registry

A service registry is the central element in a SOA to simplify and automate searching for an appropriate service. On this point, a basic explanation of a service registry is sufficient, to enable a better understanding of the SOA concepts. The services within a SOA are maintained in the service registry. Service providers publish their metadata (WSDL) into the registry, service consumer are able to lookup or search in the registry (para. 2.1.4). The lookup can be realized by several criterions, e.g. name, service functionality or business process properties. Within a SOA UDDI (Universal Description, Discovery and Integration) was established as an implementation of a service registry and is regarded by W3C as the de facto standard for web services management [W3CU 01; chap. 7]. It is based on a common set of industry standards, including HTTP, XML, XML-Schema and SOAP. So it is able to provide an interoperable, foundational infrastructure for a Web service-based software environment for public available services, as well as for internal (within an enterprise) available services [OASI 04; chap. 1.1]. The vision for UDDI is to provide a huge service registry where everyone who needs a service has access to and can use the service. Currently enterprises set up their own UDDI system to maintain services inside a company for their business processes and to hide the services from others.

2.1.3.6 Enterprise Service Bus

There is no common standard or definition of an Enterprise Service Bus (ESB), because every software vendor has its own definition of an ESB and implements its own features. Possible implementations of an ESB are, e.g. Oracle ESB, IBM WebSphere, BEA AquaLogic Service Bus or SOPERA ESB.

In the authors view, the most felicitous definition comes from Stefan Ried (Forrester Group), who describes an ESB as an “(…) infrastructure software that makes reusable business services widely available to users, applications, business processes, and other services (…)” [STRI 07; p5]. Regarding that, the main task of an ESB is to provide data exchange between applications, done via service calls. The ESB does that using the hub-and-spoke mechanism to reduce the

number of interfaces, as described in para. 2.1.1. There are other implementations of a service bus too, e.g. the distributed service bus (DSB) which SOPERA uses. The implementation of an ESB is based on the concept of a Message Oriented Middleware (MOM) using SOAP. It adds flexibility to the communication between services, simplifies the integration and reuse of services and makes it possible to connect services implemented in different technologies (Java, CORBA, etc.) in an easy way. It acts as a mediator between different and often incompatible protocols and applications [MATJ 06; p. 10]. Because of this main reason an ESB is used in a SOA. Fig. 5 shows a possible implementation of an ESB, and possible tasks executed with it.

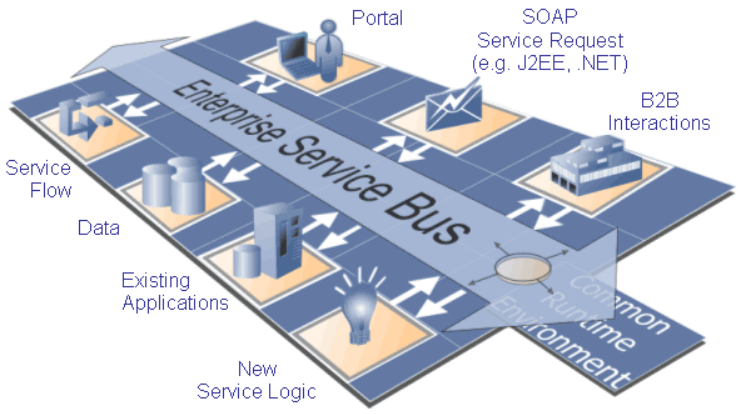


Figure 5: Implementation of an ESB [REGU 07]

Depending on the software vendor of an ESB, additional features like a registry, quality-of-service, single-sign-on, are implemented, but detailed explanation of that is waived at this point.

2.1.3.7 Service Composition

The most important concept of a SOA is the composition of services into business processes. The definition of a business process by Matjaz B. Juric (University of Maribor) helps to understand that. “(...) A business process is a collection of coordinated service invocations and related activities that produce a business result, either within a single organization or across several (…)” [MATJ 06, p. 16]. A consequence of this is that services have to be composed in a particular order and follow a set of rules to provide support for business processes. The composition of services has to provide creation, as well as modification of business processes in an easy way to enable companies to react on changes faster

and with less effort. Only if this aim is achieved, the requirements of the business can be realized with the benefits of SOA. For achieving that a language for service orchestration and an engine able to execute the processes is necessary. Doing that the Business Process Execution Language (BPEL) has been established which is explained in detail in para. 5.

2.1.3.8 Basic Architecture

Fig. 16 explains the basic architecture of a possible SOA implementation, to enable a better understanding of the concept mentioned before. A detailed explanation of Quality of Service concepts are waived at this point, because it would not contribute to a better understanding. The concepts are described with black letters. The technology used in the thesis for realization is highlighted with grey letters. The service registry uses the concepts of the UDDI implementation to store the metadata of the services. As an implementation of a service bus, the SOPERA DSB can be used. For message exchanges, SOAP is used. The programming language for the service implementation is Java and for composition and orchestration of the services, BPEL can be implemented. The architecture of the components used in the prototype is described in para. 2.3.

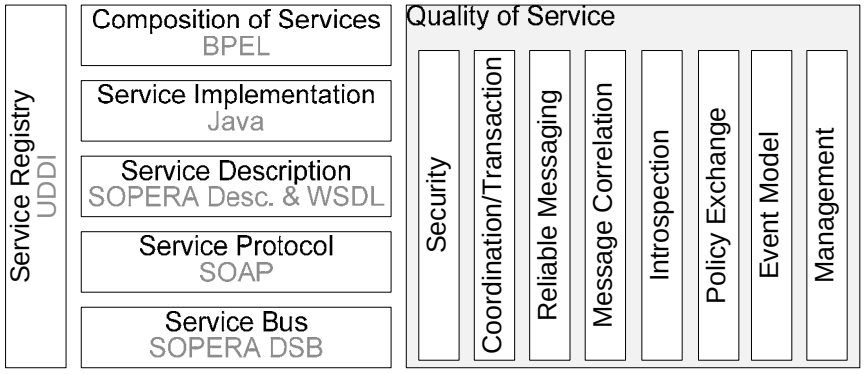


Figure 6: Placement of used technologies on the general SOA architecture adapted from [MATJ 06, p. 16]

2.1.4 Roles and Actions within a SOA

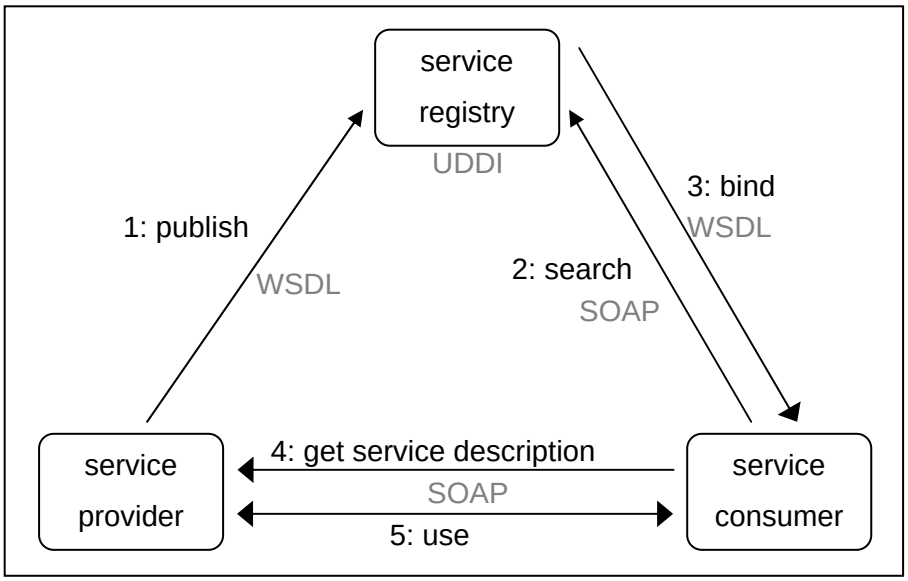


Figure 7: SOA triangle [DOST 06; p28]

Fig. 7 describes the roles and actions done in a SOA and how the service participants are interacting.

1. The service provider publishes its service description (WSDL) to the service registry (UDDI).

2. A potential service consumer requests for an appropriate service at the service registry.

3. If the requirements of the request match, the service registry sends the location (URI) to the service consumer.

4. Having the location of the appropriate service provider, the service consumer starts a request (SOAP) to the service provider, to get the exact service description (WSDL).

5. By means of the service description service consumer can use the service. The communication is based on SOAP message exchange.

2.2 SOA Concept of SOPERA

The aim of this paragraph is to introduce the SOA product of SOPERA, which is based on the concepts described in para. 2.1.

2.2.1 Overview

As mentioned in para. 1.1 the team of Sopera GmbH has experience for more than seven years in developing the SOA framework SOPERA. Regarding to Ricco Deutscher (CIO Sopera), "(…) SOA and open source are the most important trends in IT. Combining the two brings companies increased flexibility while reducing costs (…)" [SOPE 08]. That is the reason why the product is based on open source technology using the SOA standards. In May 2007, the source code of the SOPERA ESB was transferred to the Eclipse project. Doing that, the SOA runtime framework project "Swordfish" was established. The goal of that project is to extend the framework with additional open source components, e.g. a BPEL Engine or a Messaging System.

Regarding to Stefan Ried "(…) open source ESBs are not leaders today (…)", but "(...) Open Source ESBs matured significantly in 2007 (…)" [RIED 08]. That statement is based on the "Forrester Wave: Enterprise Bus, Q2 2006" (fig. 8) which shows the market position of ESB vendors in 2006. Currently there is no open source ESB within the big player. A new Version of this Forrester Wave will be available the next month. Oracle was not evaluated in 2006, because of this they are not mentioned in this picture. But currently Oracle is also a big player on the ESB market.

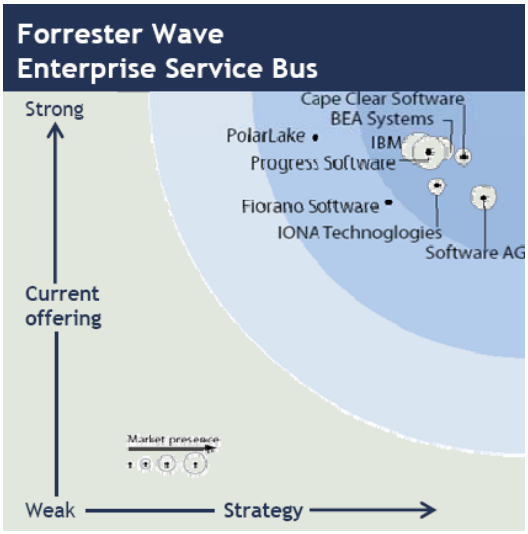


Figure 8: Forrester wave [RIED 08]

2.2.2 Sopera Framework

SOPERA supports the whole life cycle of a SOA by using its own products as well as existing open source products. Fig. 9 [SOPE 08] gives an overview about the life cycle and the provided features.

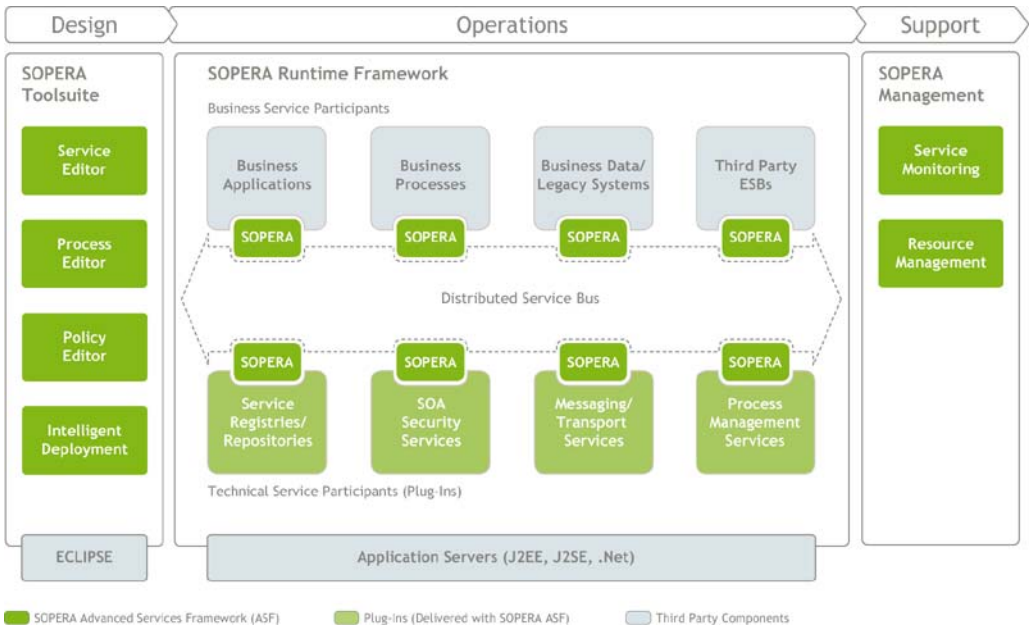


Figure 9: SOPERA Framework [SOPE 08]

Within the design phase, the SOPERA Toolsuite offers a complete Integrated Development Environment (IDE), using Eclipse. It is extended with SOPERA specific plug-ins, like the SOPERA Service Studio, for developing Web services and maintaining the SOPERA environment.

The SOPERA Service Studio is part of the ToolSuite and consists of several Eclipse based editors for developing SOPERA services.

- *SOPERA Service Editor* is used to create and edit service descriptions as well as service provider descriptions. They are used for dynamic service lookup when published into the service registry and for validation during message exchange.

- *SOPERA Policy Editor* is an Eclipse based XML editor providing a graphical user interface to create and edit SOPERA policies, which describe the complete set of Quality of Service (QoS) aspects for their service participants.

- *SOPERA Code Generator* allows the code generation of the stubs and skeletons based on the service descriptions and policies.

The SOPERA ESB, in fig. 9 named as *Distributed Service Bus (DSB),* manages the communication between service participants as well as other applications during operations. It comes with several standard plug-ins like an UDDI Service Registry, Security Services, Messaging Services and Process Management Services. It is also possible to integrate 3rd party applications into the SOA environment, like other business processes or legacy systems. This modular concept based on open standards allows rapid implementation of business requirements on the platform. Examples for that are J2EE, J2SE or .Net Application Server (AS). Usually the Tomcat application server is distributed within the development framework.

SOPERA also provides management tools, to support the pattern of a SOA. It includes monitoring, notification, controlling as well as message tracking. It is also possible to integrate this in an existing service management environment, which is mandatory for realizing that prototype using Oracle CEP and BAM.

2.2.3 Particularities of SOPERA to the Standard Concepts

As mentioned, SOPERA is based on “standard concepts” of SOA, but there are also some particularities, regarding policies and service descriptions.

2.2.3.1 WSDL Compared to SOPERA Proprietary Descriptions

Compared to a plain WSDL file, describing a Web service, SOPERA splits that information into two different files, the service description and the service provider description. The *service description* contains all the information that defines the interface of a service logically which is the `portType` and the `message`. The *service provider description* contains the concrete binding of a service to a location using the `types` and the `binding`.

2.2.3.2 Web Service Compared to SOPERA Service

The main difference between Web services and SOPERA services is that SOPERA services are able to use JMS as a transportation layer in addition to HTTP. SOPERA services allow using policies to control the message exchange during run-time. Compared to Web services using WSDL, there is only one `portType` per service description allowed. Because SOPERA uses the WS-I basic profile 1.1 with document-literal binding style, each message definition must contain exactly one part, which is able to reference only an XML `Element` not an XML `Type`. Therefore messages with only a simple type as payload are not possible. If SOPERA services are created by using the Java Code generation (what is done in this project), only a single fault message for each operation is possible, from their Application Programming Interface (API).

2.2.3.3 Policies

SOPERA uses policies to ensure that a given service consumer interacts with only matching service providers, which means that all service requirements specified by the service consumer are fulfilled by capabilities specified by the service provider and vice versa. So a policy has to describe the requirements as well as the capabilities of the participants. The requirements describe attributes that a service participant requires for communication; the capabilities describe the attributes a participant can offer. SOPERA policies are also used to determine QoS attributes which are non-functional properties, like security aspects or performance, etc. Those attributes are defined with the policy assertions.

There are Several Types of Policies:

Participant policy describes QoS capabilities and requirements for one service. A service is able to have several participant policies.

Agreed policy is used during a conversation between an instance of the service provider and the service consumer. The agreed policy is generated by the service registry during a service consumer (consumer policy) lookup (fig. 7; 2: search) for a potential service provider (provider policy) using the policies. If there is a match between the capabilities and the requirements of both sides, the agreed policy is created (fig. 7; 3: bind) and used to determine the attributes of the message exchange. The agreed policy contains operation policies for each operation which are available during the conversation.

Default policy is a policy generated by the service registry if the service registry was not able to create an agreed policy. There are several default policies available, the default provider policy, the default consumer policy and the default agreed policy.

Operation Policy is the subset of a policy that applies to a single operation and describes as part of an agreed policy the QoS properties, which have to be applied during each step of processing and contains only mandatory elements.

Policy assertions describe the particular attributes of the message exchange, and define as mentioned the capabilities and requirements for non-functional aspects of service calls. They control e.g. security aspects, as well as message tracking for *SBB events* (para. 3.2.2.1) and correlation (para. 4.3.3). Based on the attributes of the policy assertion the service registry ensures during the service lookup, that only matching policies interact with each other, which ensures the correct processing of service invocations.

2.3 Basic Architecture of the Prototype

This paragraph provides a short overview about the components and applications used for the prototype in the diploma thesis. The short explanation refers to the paragraph, where the detailed explanation can be found. Fig. 10 shows the simplified system environment.

2.3.1 Architectural Overview

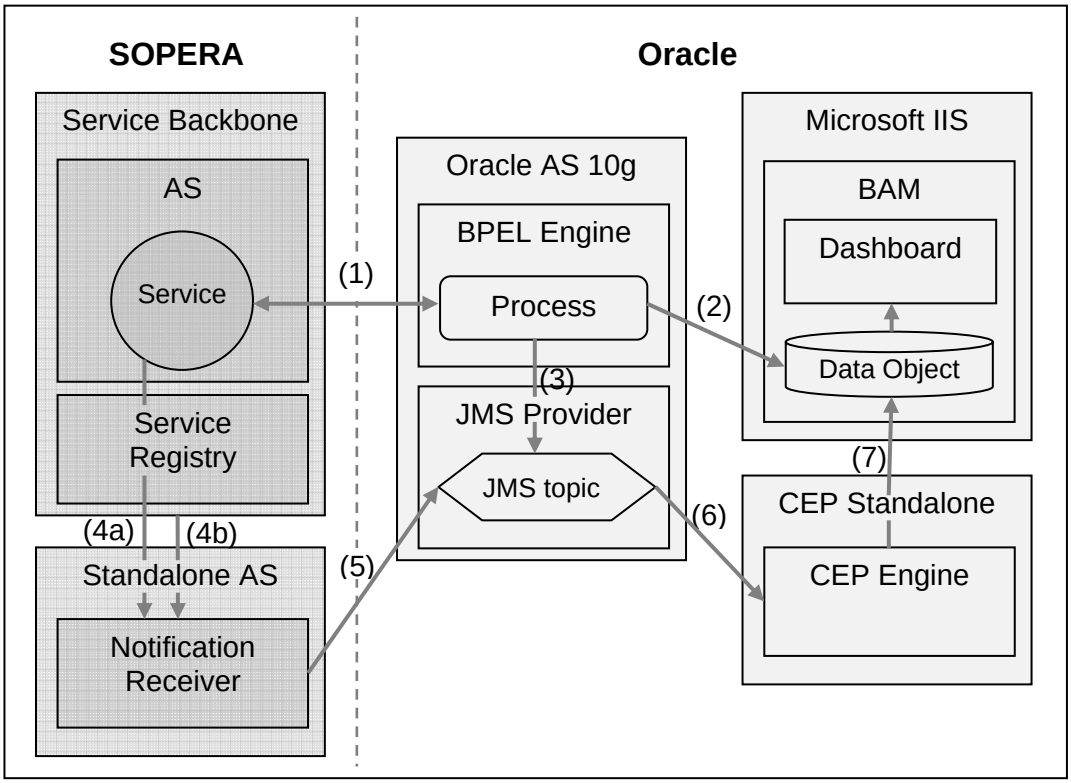


Figure 10: System environment

2.3.2 SOPERA Components

Service Backbone: The SOPERA Service Backbone (SBB) includes all basic SOA components, e.g.

- **Application Server (AS):** Within an AS all SOPERA services are running.
- **Service Registry:** The Service Registry is responsible for service lookup as described in para. 2.1.3.5.

Standalone AS: It is independent from the SBB and includes the **Notification Receiver (NR)**. The NR component (para. 6) is responsible for publishing events to CEP and BAM, after a SOPERA service was called (1). On the one hand, *SBB events* (para. 3.2.2.1), including more technical events are fired (4b). On the other hand *business events* (para. 3.2.2.2), including business content are fired (4a). The NR publishes these events to a *JMS topic* (para. 7) (5) for further processing.

2.3.3 Oracle Components

Oracle AS 10g: The Oracle AS, also known as OC4J includes and manages all Oracle SOA components. On this point the focus is just on the BPEL engine and the JMS provider, which are directly involved during the implementation of this prototype.

- **BPEL Engine:** The BPEL engine holds all business processes and is responsible for running them. After a business process is created and deployed, it is possible to run it within the BPEL engine. An instance of the process calls a SOPERA service (1). During running, events are published to a *JMS topic* (3) as well as to a *Data Object* (2).

- **JMS Provider:** This component is responsible for providing *JMS topics*. A *JMS topic* is an endpoint which allows publishing (3, 5) and accessing (6) XML data.

Microsoft IIS: **Oracle BAM** runs in version 10g within the Internet Information Server (IIS) from Microsoft. It includes **Data Objects** for storing events and monitoring information as well as the **BAM Dashboard** for presenting the monitoring reports.

CEP Standalone: Because CEP is not part of Oracle 10g there is a standalone CEP version available. CEP is embedded into Oracles AS. CEP standalone includes the **CEP engine** for event processing. It accesses (6) events, provided by several JMS topics which allows event correlation. After processing, the correlated events are published (7) to BAM for reporting.

2.4 Future Development

This paragraph gives a short forecast about current developments and new technology approaches.

2.4.1 SOA Combined with CEP

In the last month, during which this thesis was written, several buzzwords, like “SOA2.0” or “advanced SOA” came up. Gartner created a formula saying “SOA2.0 = SOA + EDA” what is propagated by them as the new SOA approach as shown in fig. 11. This statement has become a hot topic by experts, also on the OOP 2008 conference in Munich. Experts rejected this statement, because Event

Driven Architecture (EDA) is seen as an independently existing architecture concept. Therefore the approach has to be to enable a proper working of the architecture concepts of SOA and EDA and not to create a new buzzword for marketing purpose. Experts claim CEP vendors to provide better tracking mechanism to get the control of the events [UEBE 08].

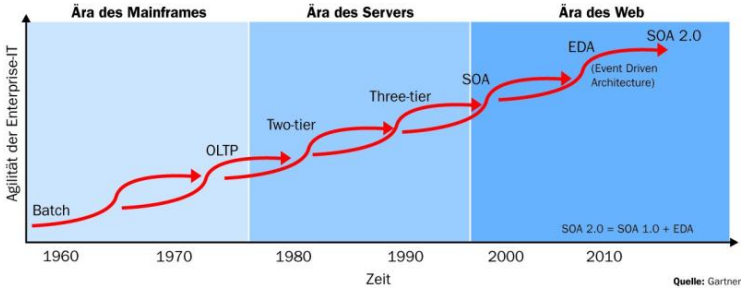


Figure 11: Architecture development [HERR 07]

Regarding to Clemens Utschig [UEBE 08] the CEP approach is mainly used in the finance sector, where the event approach is mostly hard coded. In this thesis, it is attempted to combine the SOPERA SOA framework with Oracle CEP, which is currently still in development. Therefore different approaches have been done to integrate CEP, as described in the thesis.

3 Business Process with Focus on BPEL, CEP and BAM

This paragraph describes the example business process for the prototype. First the process is described in EPC notation to give a rough overview. After that the different event types, occurring in that process are described, followed by several test cases for simulation purposes, as well as approaches for event generation and event stream processing.

3.1 Business Process

This chapter describes the business process used in this thesis, the event occurring in it, use cases for simulation as well as several approaches for event generation and event stream processing.

3.1.1 Specification of a Business Process

The background of the prototype is the analysis of the opportunities of CEP and BAM on a business process of Deutsche Post AG that should be provided by the Sopera GmbH. Because of the confidentiality of the real business processes and a Non Disclosure Agreement (NDA) Sopera respectively Deutsche Post was not able to provide a real business case. Therefore, a sample process related to logistics was specified. This process contains no real business functionality, but the possibility to provide information and data for using BAM and CEP use cases. The typical, practical use case related to a logistics company is a shipment combined with a claim process.

The following specification gives an overview about the basic functions of the business process to provide a better understanding of the whole topic, especially of the provided events in para. 3.2.6 and the test cases in para. 3.2.8.

3.1.2 Description of the Business Process in EPC Notation

Fig. 12 and 13 describe the flow within in the example process, which is explained in the next paragraphs. The syntax is not really Event Driven Process Chain (EPC)-conform to reduce the complexity for the reader but describes the issue in an ordinary way. For execution purpose, Web services were developed and orchestrated with Oracle BPEL, as exemplified described in chap. 4 and 5.

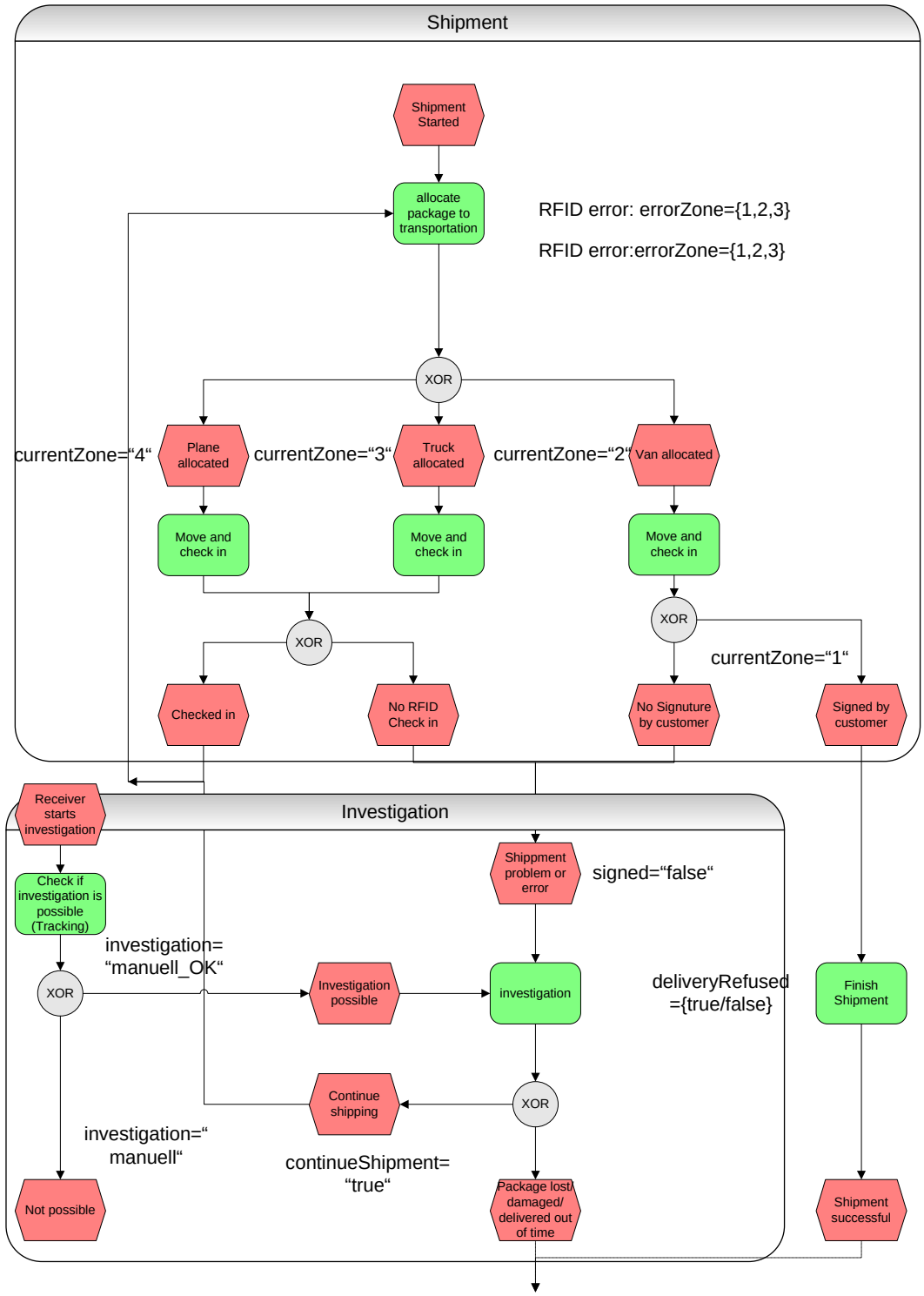


Figure 12: Business process 1/2

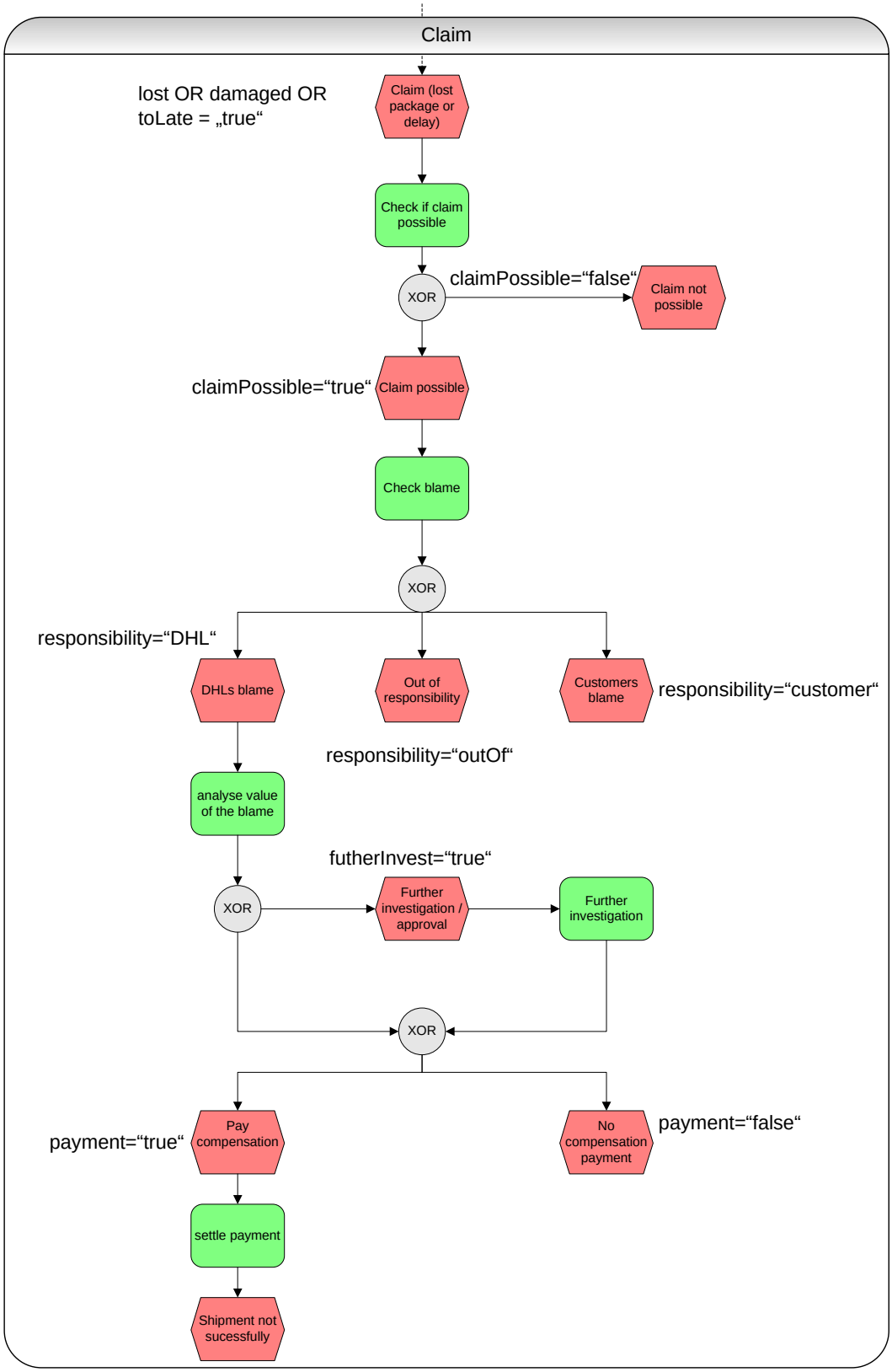


Figure 13: Business process 2/2

3.1.3 Shipment

The process is initiated when a customer ships a package to a specific address. A RFID tag allows the identification of the parcel during the whole shipment process.

After shipment starts, the package has to be allocated for transportation. This step calculates the route of each package and is divided in separate zones. For reducing complexity, just 3 zones are modeled - plane, truck and van - one for each types of transportation. It is possible that a package starts in any zone, but the shipment ends always in the last zone with delivery by van. E.g. a package which starts shipment by plane has to pass all three zones, plane, truck and van. If it starts with van, only this zone has to be passed and the package is directly delivered to the customer.

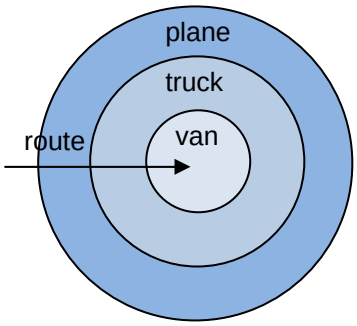


Figure 14: Transport route

In every zone miscellaneous problems are thinkable and different kinds of events take influence on the parcel and the shipping process. E.g. on one hand a plane is not able to start because of bad weather or strike of pilots. On the other hand it is possible that a van has an accident during the delivery process. All these events are responsible for losses, damages or delay during the shipment process. During the whole process of shipment RFID events are sent throughout the system. If in a determined period of time no RFID message happens and the IT-system is still running, a shipment error is detected.

In the last zone the package is shipped by van and assigned to the customer, who commits the receipt by signature on the hand held unit of the courier. After arrival in the office, the handheld is synchronized with the IT-system and provides the information about the delivered goods by signature events. Both, RFID and

signature events, correlated with other events give information about possible shipment errors in future.

If during shipment no errors are detected, it seems to be the package was successfully delivered to the recipient. But it is also possible that the freight was damaged during shipment, so the customer is entitled to claim within a defined period of time as described in para. 3.1.5.

3.1.4 Investigation

Usually the investigation is automatically started, after a shipment error or problem is detected. This is done either, if no RFID-event was received within a specified time or the information about the customer signature is missing. But also other, more complex error events are able to initiate that process. E.g. if it is detected that a truck has not changed its GPS position in a period of time and there is no traffic information about a traffic congestion.

A customer is also able to start the investigation if he wants to know, what happened with his parcel, because it has not reached the addressee yet. Before detailed investigation is started, the customer receives basic tracking information. Meanwhile it is checked, if investigation is possible for the package at the moment. This depends on the delivery agreements, e.g. whether a package has to be delivered as express mail with a delivery time of two days or as standard mail within ten working days. If it is too early for investigation, the customer has to wait until it is possible, otherwise the workflow for the investigation is started.

During the investigation, staffers try to find out where the package is located and check if shipping can be continued or the freight has been lost. In case that a package is out of the agreed delivery time investigation is also started. If during the workflow of investigation a RFID event is received, the investigation stops and shipment continues. If an error was detected, that means the parcel has been lost, damaged or was not delivered in agreed time the claim process is started.

3.1.5 Claim

A claim is started if a parcel was not delivered in time, lost, damaged or shipped successfully but the customer rejects it. First it is checked whether a claim is possible in this case or not. E.g. if a parcel was sent without any insurance or

claim and it is more than seven days after the package was shipped, the claim is not possible anymore.

In case of a possible claim, it has to be analyzed by a staffer who is responsible for damage. There are a lot of influences which have to be considered to assign the responsibility of the damage. E. g. the supplier contracts out of liability, if the freight has to be delivered in an area of conflict or an earthquake or storms has taken influence on the freight. The customer is also responsible if the box for shipping package was wrong or the freight was in bad packing. In this case the supplier also excludes liability.

In case of DHLs blame, the value of compensation has to be analyzed by staffer in the next step. The decision about payment of compensation is either done immediately or further investigation or approval has to be done. This is necessary when facts are not clear or the amount of damage is above a defined threshold. In this step the account of the damaged freight is necessary or the damaged value has to be analyzed by an independent expert who clarifies the blame and value of damage.

Depending on whether compensation payment is necessary, the process ends or the next step is to settle the payment. The payment is be done by check. After the amount of damage is established the customer receives a check which has to be paid.

3.2 Events Generated by the Business Process

According to the definition of the term “event” in the literature it has to be explained which types of events are used in this use case and how they are classified. Furthermore an approach for extending the generation of events has to be discussed and developed.

3.2.1 Event Definition

”(…) An event is an object that is a record of an activity in a system. The event signifies the activity. An event may be related to other events. (…)” [LUCK 07; p. 88]. According to this definition three aspects are given to an event, the form, significance and relativity.

”(…) The Form of an event is an object. It may have particular attributes or data components. A form can be something as a simple as a string or more often a tupel of data components. (…)” [LUCK 07; p. 88]. An example for this is a timestamp, a unified identifier or a simple attribute for further event correlation.

“(…) An event signifies an activity. We call this activity the significance of the event. An event’s form usually contains data describing the activity it signifies. (…)” [LUCK 07; p. 88]. The RFID event in para. 3.2.6.2 is described by its attributes, e.g. the RFID number or the expected shipping date.

The third aspect is the relativity. “(…) An activity is related to other activities by time, causality and aggregation. Events have the same relationships to one another as the activities they signify. The relationships between an event and other events are together called its relativity. (…)” [LUCK 07; p. 88]. The relationship of the events in our case is realized either by the business process, event type or time interval. The relationship by business process means, that all events occurring within a specific instance of the process have a relation and can be correlated. The detailed mechanism is described in para. 3.2.3. The relation of event type is directly by the name of the event type possible, e.g. all RFID events. The relation by time is implemented with the attribute “`TimeStamp`” or with other time or date related attributes, which are also part of the basic events, defined in para. 3.2.6.

3.2.2 Types of Events

The events used for the prototype are divided into three categories: first the more technical *SBB events*, which give information about what the service does at the moment. The second types are *business events*, describing what is happening during the business process or during the workflow. And the last type is the *BPEL Sensor event*, realized with BPEL sensors, which provide information about the process instance. This mechanism works independently from the other events, coming out of SOPERA.

3.2.2.1 SBB Events

The content of the *SBB events* is, for example, the time when a service was started or error events occurring during startup or during an operation call. Simple summary information, about how often an operation was called during the last period of time is also part of *SBB events*. They are enabled with the SOPERA

policy "`tracking assertion`", described in para. 4.3.2. *SBB events* are be fired automatically without any effort for the developer. He only has to define the level of detail in which he is interested. The amount of *SBB events* can be configured from only summary events to very detailed debug or trace events. If the level of detail is configured very high, a lot of *SBB events* are fired during a service call. Depending on the hardware, influences on the performance of the system are possible.

3.2.2.2 Business Events

The main disadvantage of *SBB events* is, they do not provide any business information, which is much more interesting and mandatory for analyzing and monitoring processes with CEP and BAM. Business events provide content about the business case. They describe for example when a package was delivered to the customer and who has signed it. Another event is the receipt of an RFID message to locate exactly where the package currently is. These events provide content for analyzing and acting during the business process. They are not fired automatically by configuring the SOPERA policy side. They must be exactly defined. Each event provides different content about the current status within the business process. The event has to be defined exactly with the information needed for further analyzing and correlation. A description of the events, necessary for the prototype is discussed in para. 3.2.6.

3.2.2.3 BPEL Sensor Events

Sensors within the BPEL process provide events for CEP and BAM and are the third event type. A sensor is implemented directly in BPEL and provides information about the running process instance. This is information about activities, faults or variable data of the BPEL process.

An advantage of a sensor is that it can be added to every process component directly by using the BPEL process designer within JDeveloper and sensors are running independently form the business process. So sensors have no influence on the business logic of the process, on the development side, as well as during runtime. This mechanism is also called non-intrusive and is described in para. 3.2.9. A sensor is able to provide not only information about the implemented SOPERA services but also about other BPEL components like an *Assign activity*

or *Switch*. A disadvantage is that only data which is directly accessible in BPEL can be provided through a sensor. That means the sensor does not know what is happening within a service during the process execution. Just data of the message payload is accessible by the sensor as well as service faults. That is the reason why besides sensor events other event types are necessary.

Regarding to a Tech Note of Oracle about sensors, "(…) the BPEL process should publish all its sensor events to a JMS topic bus. In this version, only JMS topics are supported (…)" [ORAC 07; p. 1]. Para. 5.2.2 contains more information about sensors as well as an implementation example.

3.2.3 Basic Concept of Extending a SOPERA Service with Events

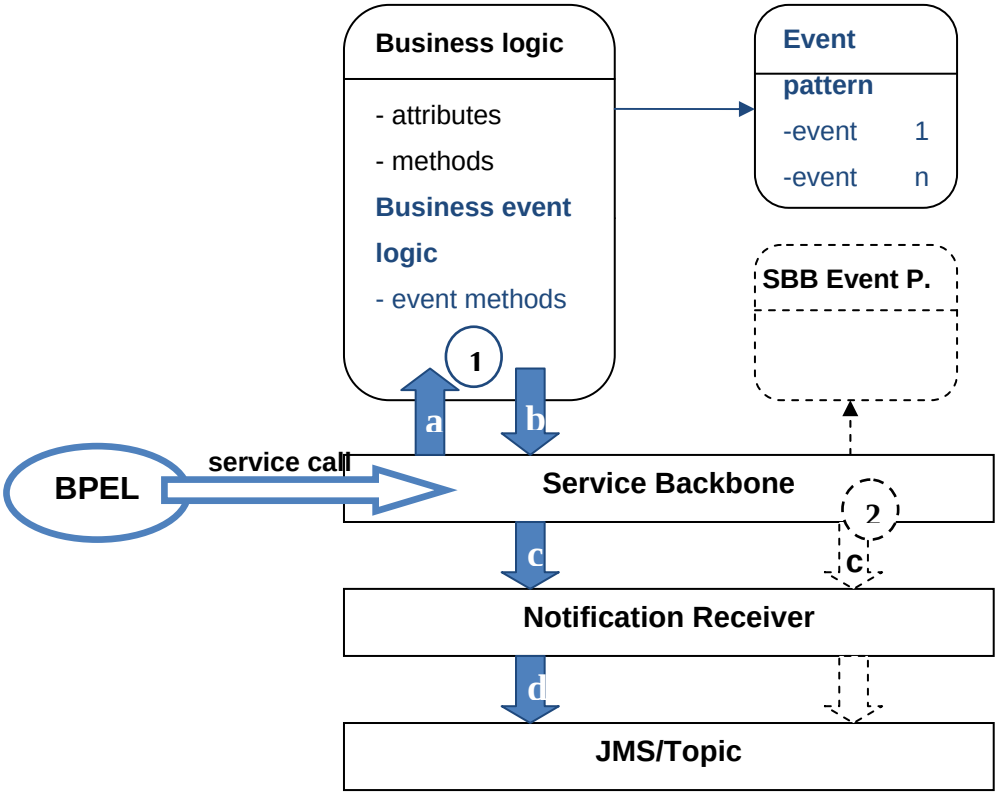


Figure 15: Basic concept of publishing events

As mentioned in the paragraph before, there are different types of events within the SOPERA environment. On the one hand there are *SBB events* which provide technical information. They are fired automatically via the Service Backbone (c') if a service is invoked from the BPEL process engine. The programmer is able to configure only the level of detail, but not the content of these events. So the events include only technical information, because there is no communication between

the service and SBB. These events are directly fired when a service call is received on the SBB. That concept is expressed by dashed lines in the figures (no. 2).

On the other hand, in the case of the prototype, a service has to provide information about business content and process information out of BPEL. This is not realized automatically via the SBB, because the SBB has no information about the encapsulated attributes of the service. If a BPEL service invocation is executed, the method will be called via the SBB. The SBB handles the communication and calls the service method (a). During the service processes, events are generated (b) and sent via SBB to the NR (c) which is able to allocate the fired events to several destinations, like a *JMS topic* (d).

3.2.4 Approaches of Basic Event Patterns and the Component which Generates the Event

For the technical implementation in case of the prototype, it is necessary to extend each service to provide events. Event patterns define the structure and the content of a basic event. One solution of defining events and their content is a definition in a *property file*. A property file describes all events occurring in a service. Every entry in this file is used as a framework for a specific event. If an event is fired, the service enhances the pattern, defined in the property file with data of the service. In a next step the event is forwarded to a specific destination which has to marshal the information and sends it via *JMS topic* to Oracle CEP or BAM where the information has to be unmarshaled for further processing. One solution to handle and transform the events on the Oracle side uses XML documents. So it is necessary to provide the events as XML document.

For defining the basic event patterns and generating an event for the prototype several approaches are possible. The following paragraph points out the differences between the concepts, the advantages and disadvantages, followed by a decision. For better reading, the structure is adopted from the JEE pattern description of Adam Bien [BIEN 07] and each section is structured in the following way:

- Essence:

  The essence provides an overview about the components used and their tasks during generating and sending events. This is clarified with an overview picture at the beginning.

- Context:

  The context describes the influences on the technical side and points out the effort needed to use this solution.

- Application Recommendation:

  Describes the chosen design environment and explains the constraints on the business side as well as on the technical side.

- Example:

  This part rounds up the illustration with a simple example.

3.2.4.1 XML Event Pattern

Essence

The XML structure of an event is defined in the property file and is forwarded without any further transformation to the destination. On the Oracle side, the event is unmarshaled and the extracted information is used for further processing on the BAM and CEP side.

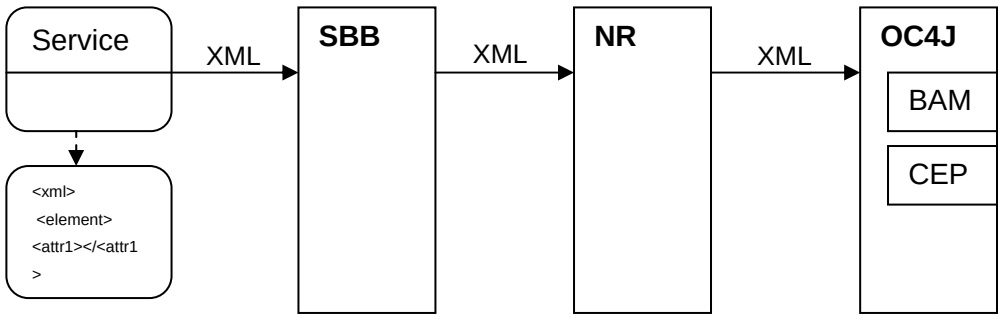


Figure 16: Event pattern as XML

Context

- The structure of an event is flexible and is defined completely by the developer
- No influence on SBB and NR, no changes have to be done
- Defining and implementing an event is very complex, error-prone and time consuming
- Very high effort, if the layout of the XML has to be changed

Application Recommendation

This draft is only recommended if the developer does not have any influence on the SBB and NR components. Designing an event pattern is time consuming, laborious and error-prone, because the whole event structure must be defined for every single event as an XML file. The effort to describe an event with its whole XML structure is higher compared to the other solutions, where only the attributes must be defined. The advantage is that the developer defines the whole XML structure which is sent to an endpoint. In the other solutions that is done automatically and the developer does not have any influence on it.

But the biggest expenses emerge if the layout of the XML document needs to be changed. That is necessary if the system requirements of the consuming components are changing and, for example, the structure used in the XML must be changed. In this case every event description has to be modified

Example

Fig. 17 shows an XML event description which defines the content to be transferred. Even for a small amount of variables, the pattern is very complex and hard to handle. So, it is not recommended to use this solution.

```
<msg host="'{x}'" app="{x}'"          cat="{x}'"   id="{x}'"    lvl="{x}'"
          text="{x}'"
          xmlns="http://schemas.sbb.org/management/Notification/1.0">
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
  <para   xmlns="http://schemas.sbb.org/management/Notification/1.0">{x}</para>
</msg>
```

Figure 17: XML event pattern

3.2.4.2 Event Pattern with Simple Attributes Transformed into XML by SBB

Essence

The event pattern is defined using simple attributes. Marshaling is done by the SBB which generates a XML document. The further processing is based on the same way as mentioned before.

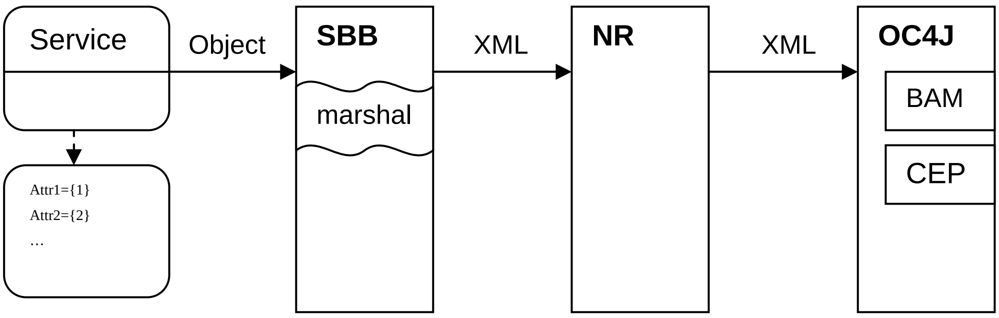


Figure 18: Event pattern transformed into XML by SBB

Context

- The content structure of an event is flexible and is defined by the developer
- Influence on SBB, changes regarding transformation have to be done
- Defining and implementing an event is much easier than on the solution before
- No changes necessary when XML structure changes

Application Recommendation

This draft is recommended if the developer has influence on the SBB because some changes regarding marshaling must be done. But instead of having big efforts during the implementation of every event, those changes must only be done once. Also, if the XML structure changes, there is no influence on existing event definitions. An event pattern can be defined very easily, because only the attributes of the event have to be defined in the property file.

Example

Fig. 19 shows a property file including an event definition. Based on the pattern, an event object is generated by the service. That object is forwarded to the SBB where marshaling to an XML event is executed. The created XML can be forwarded to a defined endpoint.

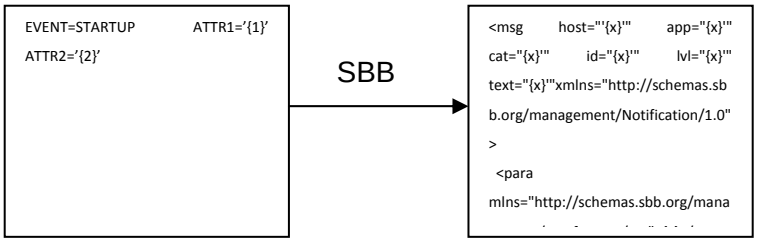


Figure 19: Transformation of an event pattern

3.2.4.3 Event Pattern with Attributes Transformed into XML by NR

Essence

The event pattern is defined using simple attributes as in the example before. Marshaling is done by the NR instead of the SBB. The NR sends the generated XML document for further processing in the same way as in the other examples.

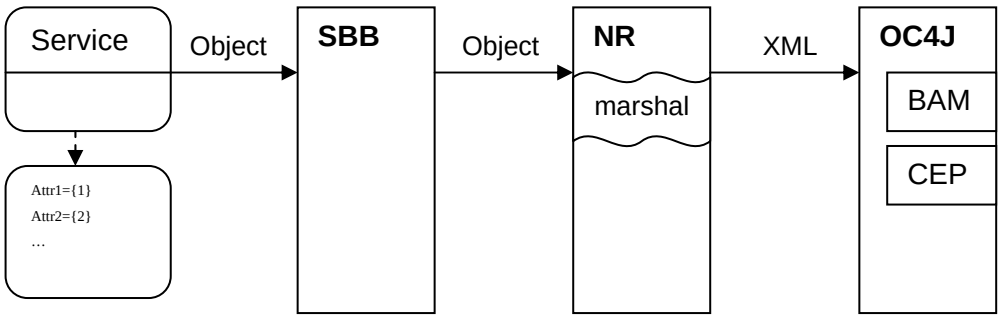


Figure 20: Event pattern transformed into XML by NR

Context

- The content structure of an event is flexible and is defined by the developer
- Influence on NR, changes regarding transformation must be done
- Defining and implementing an event has less effort than the first solution
- No changes necessary when XML structure changes

Application Recommendation

The explained solution comes up with nearly the same advantages and disadvantages as the recommendation mentioned before. Instead of extending the SBB, the NR must be changed to provide the XML transformation.

Example

The example is equivalent to the SBB example in para. 3.2.4.2, as mentioned before.

3.2.4.4 Event Pattern with Attributes Transformed by a Stand-Alone Event Transformer

Essence

The event pattern will be defined using simple attributes as well as described in the examples before. To transform the event object a new component is necessary. This event to XML transformer is responsible for marshaling the received event into the needed XML structure. After transformation, the event is forwarded by the other components to the endpoint, the OC4J.

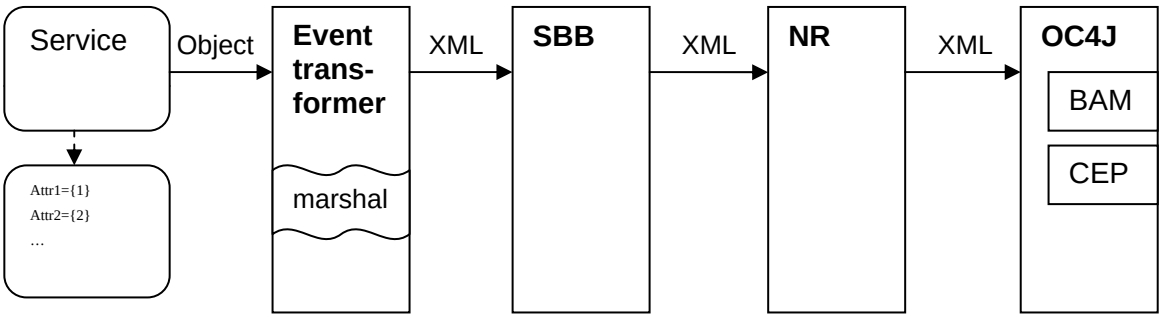


Figure 21: Even pattern transformed by a stand-alone transformer

Context

- The content structure of an event is flexible and is defined by the developer
- No influence on SBB and NR, no changes must be done
- Defining and implementing an event is much easier than on the first solution
- No changes necessary when XML structure changes

Application Recommendation

The existing components, the NR and SBB are used without any changes, because a new service component provides all functions necessary for transforming the incoming event objects. This solution can be used in every system environment. It is also possible to develop the event transformer as stand-alone web service, which can be replaced easily with an equivalent one, providing the same functionality. So this pattern belongs to the independent solutions.

Example

The example is equivalent to the SBB example mentioned before. Instead of the XML transformation with the SBB it is done by a stand-alone event to XML transformer.

3.2.4.5 Implementation Decision

For the implementation of the prototype, the “event pattern with attributes transformed by the notification receiver” (para. 3.2.4.3) is chosen. The reason for this decision is based on different influences. The worst decision in our opinion was using the solution “event pattern as xml”. As mentioned, the effort for designing events is very high, error-prone and should only be used if there is no

possibility to extend the components SBB or NR. The solution, based on extending the SBB is not possible in our case, because this component is not controlled by our project. So the decision came up to use the possibility to extend the NR with the function of marshaling the event objects. This is in this context the best solution because only the existing components have to be extended. Another advantage is that it is very simple to create event patterns and send events out of the service. This solution also offers an advantage if the structure of the XML document has to be changed. In this case only the marshaling function has to be modified instead of every event pattern defined in the system. For using this functionality in a real system environment, creating a stand-alone event transformer will be the best and independent solution. For prototype purposes, it would be too much effort and very time-consuming to develop a stand-alone event transformer. This mechanism is also based on an intrusive event generation, as described and comes up with some disadvantages as described in para. 3.2.9.2.

3.2.5 Identifying Events with BPELContent

Before events are ready to be used for further processing they have to be identified. This means that every event needs an identifier which can be used for further processing and correlation in Oracle CEP und BAM. If events do not provide an identifier, it is not possible to identify them and to see to which origin they belong. In this case events do not provide any benefit. They are only becoming bugs and useless data heaps.

The only identifier providing this information is available within the BPEL process and is called "`Instance ID`". For every started business process, Oracle BPEL creates this individual identifier. But this field is not directly available within BPEL and has to be transferred via XPath statement to the service. The service must extend the event with this identifier whereby events are assigned to a business process instance. This identifier will be provided with an element called "`BPELContent`". This element is part of the payload. The implementation is explained in para. 4.3.3.

Fig. 22 describes the message flow and explains the basic concept behind that idea. The color blue highlights the Oracle components, orange stands for the

SOPERA components and grey is for a service call from Oracle BPEL to SOPERA.

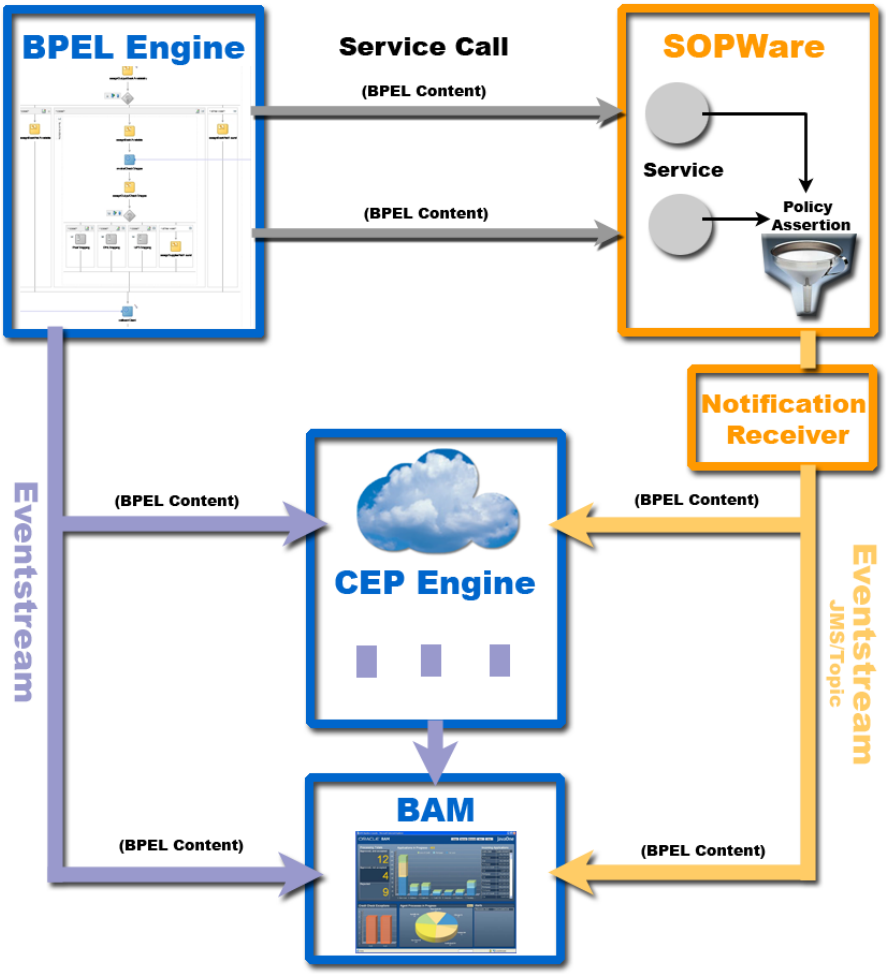


Figure 22: Event flow

3.2.6 Basic Events within the Business Process

This paragraph describes the basic events occurring within the business process of the prototype and is divided into common events, shipment events, investigation events, claim events and external events.

3.2.6.1 Common Events for Every Service

3.2.6.1.1 Startup Event

During the development, the first intension was to use a *SBB event* to receive information about invoked operations out of BPEL. Unfortunately *SBB events* do not provide all the information needed for monitoring and analyzing. The biggest drawback is that they do not provide BPEL content, e.g. the "`ProcessID`" out of

a BPEL process for being able to correlate events in a further step. Another disadvantage is that *SBB events* are very complicated to handle on the correlation side. For extracting all needed content from the message, very complicated and error prone mechanisms are necessary. That is the reason why the format of those events is very rigid and cannot be changed.

This is done to avoid errors during this step and for more flexibility as well as an easier implementation of the prototype and to define and design its own startup events. This event is fired every time when an operation is triggered and provides all necessary information about the service call.

STARTService	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
Operation	name of operation called
StartupTime	time when the service was started
ProcessID	process ID, extracted from BPELContent
ProcessURL	BPEL process URL, extracted from BPELContent
ProcessOwner	process owner, extracted from BPELContent
ProcessVersion	process version, extracted from BPELContent
TimeStamp	time, extracted from BPELContent

Table 1: STARTService event

3.2.6.2 Shipment Events

3.2.6.2.1 RFID Received Event

During the shipment process the package passes a couple of ways of transport. In every section the package is scanned and package data will be sent to the system as “RFID” event. These events allow future steps to get information about the delivery status of the package. It also can be analyzed if a package was lost or is still on shipment.

RFID	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
StartDate	date when shipping process was started
CurrentDate	current date in shipping process
ExpectedShippingDate	expected date, when package will arrive at recipient
CurrentZone	transport zone, where the package is right now
DeliveryTimeAgreement	maximum delivery time agreed with the customer
TransportID	vehicle which transports the package currently
TimeStamp	time when the event occurs

Table 2: RFID event

Toll Exception Event

Packages shipped into other countries will be randomly tested by customs inspection. During that process, errors can occur which can prevent further shipping. Reasons for that are e.g. drugs, weapons or food was sent, or just no duty was paid.

TollException	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
Reason	reason for duty error
CurrentZone	transport zone, where the package is right now
CurrentDate	current date in shipping process
ExpectedShippingDate	expected date, when package will arrive at
Country	destination where the package should be shipped
TimeStamp	time when the event occurs

Table 3: TollException event

3.2.6.2.2 Package Damaged Event

A package which is delivered over a long distance is not always treated carefully and damages during the shipment can occur. Sometimes only the box is damaged but it is also possible that the contents inside the package can be affected. The following event occurs if damage is noticed by a DHL employee, who reports the

error and an event is sent. This event starts the investigation process automatically.

Damage	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
CurrentZone	transport zone, where the package is right now
CurrentDate	current date in shipping process
ExpectedShippingDate	expected date, when package will arrive at
Country	destination where the package should be shipped
AreaOfConflict	true, if the destination is in an crises area
Amount	value of the package
Insurance	true, if the package is covered
TimeStamp	time when the event occurs

Table 4: Damage event

3.2.6.2.3 Package Delivered

The process “shipment” is supposed to end with the event “FinishShipment”. The package is delivered to the recipient, who confirms with signature. But this event does not exclude a future claim. It is also possible that the package was too late or content inside the package was damaged. In that case, there is a claim despite successfully delivering possible. This event only informs that the recipient has received the package.

FinishShipment	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
StartDate	date when shipping process was started
CurrentDate	current date in shipping process
ExpectedShippingDate	expected date, when package will arrive at
DeliveryTimeAgreement	maximum delivery time, agreed with the customer
Signed	recipients signature
Country	destination where the package was shipped to
TimeStamp	time when the event occurs

Table 5: FinishShipment event

3.2.6.2.4 Delivery Refused

If the courier wants to deliver a package but the recipient does not accept the package, this event is fired. Reasons are that the package was delivered too late, it is damaged or simply the recipient does not need the ordered items anymore. In this case, the package will be shipped back to the sender.

DeliveryRefused	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
StartDate	date when shipping process was started
CurrentDate	current date in shipping process
ExpectedShippingDate	expected date, when package will arrive at
DeliveryTimeAgreement	maximum delivery time agreed with the customer
Amount	value of the package
Country	destination where the package was shipped to
Reason	reason for refused delivery
TimeStamp	time when the event occurs

Table 6: DeliveryRefused event

3.2.6.3 Investigation Events

3.2.6.3.1 Possibility of Investigation

If a customer wants to start an investigation process, the system has to check, if it is acceptable or not. The customer can also intend to start an investigation process without having any claim for investigation. In this case, shipment is still within the agreed time or the package was already shipped.

InvestPossible	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
StartDate	date when shipping process was started
CurrentDate	current date in shipping process
ExpectedShippingDate	expected date, when package will arrive at
CurrentZone	supposed transport zone of the package right now
DeliveryTimeAgreement	maximum delivery time agreed with the customer
Initiated	who has initiated investigation
Possibility	true, if investigation is valid
TimeStamp	time when the event occurs

Table 7: InvestPossible event

3.2.6.3.2 Start Investigation

If the former check resulted in a possible investigation or the investigation was started automatically, an event is fired. This event gives information about the shipment process. Correlated with other events, e.g. the RFID event, it is possible to take action on the running shipment process. It is also possible to inform responsible persons about a possible loss.

InvestStart	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
StartDateInvest	day when investigation was started
StartDateShip	day when shipping process was started
CurrentDate	current date in shipping/investigation process
ExpectedShippingDate	expected date, when package should arrive at
CurrentZone	supposed transport zone of the package right now
TimeStamp	time when the event occurs

Table 8: InvestStart event

3.2.6.3.3 Continue Shipment

When the staffer was able to solve the problem during investigation shipment continues. This simple event is fired if the package was located and shipment

continues. If an investigation is running because the system received no RFID message for a certain period of time and suspects that the package is lost, an incoming RFID event causes the “ContinueShipment” event.

ContinueShipment	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
currentZone	transport zone, where the package is right now
TimeStamp	time when the event occurs

Table 9: ContinueShipment event

3.2.6.3.4 Result of Investigation

After an investigation is started and it is not possible to continue shipment within a certain period of time, the investigation quits with the finishing information in the “InvestResult” event. The “InvestResult” event also provides status information about the investigation.

InvestResult	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
StartDateInvest	day when investigation was started
EndDateInvest	day when investigation was closed
StartDateShip	day when shipping process was started
CurrentDate	current date in shipping/investigation process
expectedShippingDate	expected date, when package should arrive at
CurrentZone	currently supposed transport zone of the package
Result	result or current status of investigation
TimeStamp	time when the event occurs

Table 10: InvestResult event

3.2.6.4 Claim Events

3.2.6.4.1 Claim Possible

Before a claim can be started, it is checked if a claim is basically possible. It is only possible to complain if there was an error during shipment. This is only possible in case of damage, lost or too late delivery, as already mentioned. Every

claim is analyzed by a staffer, who decides about the further processing. If a claim is not possible for a case the process ends with this step. The “ClaimPoss” event gives information if a claim is possible.

ClaimPoss	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
StartClaimDate	day when claim was started
CurrentDate	current date in claim process
ExpectedShippingDate	expected date, when package should arrive at
Country	destination where the package was shipped to
AreaOfConflict	true, if the destination is in an crises area
Amount	value of the package
Insurance	true, if is package is covered
Possibility	true, if is claim is possible
TimeStamp	time when the event occurs

Table 11: ClaimPoss event

3.2.6.4.2 Check Responsibility

In case of a claim based on a delivery issue it is possible that the next scope tries to figure out, who is responsible for it. If DHL is responsible for, e.g. too late delivery they have to pay compensation. That is covered by DHL’s insurance. The insurance does not pay for a delivery to an area of conflict which means the claim is out of responsibility. A customer, who has not boxed the content well, can be also responsible for the damage. In this case he does not receive any compensation. The “ClaimRespons” event shows the responsibility for the error.

ClaimRespons	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
Responsibility	who has to compensate the claim
ShipmentFinished	true, if shipment was finished
AreaOfConflict	true, if the destination is in an crises area
Country	destination where the package was shipped to
BadPackaging	true, if stuff was bad boxed
Amount	value of the package
Insurance	true, if is package is covered
Damaged	true, if damaged could be detected
ToLate	true, if delivery was to late
Lost	true, if package was lost
TimeStamp	time when the event occurs

Table 12: ClaimResponse event

3.2.6.4.3 Value Analyses

After the question of **responsibility** is clarified, the claim has to be compensated. For low values the staffer is allowed to approve at first instance the payment. In all other cases further investigation is necessary. If the staffer decides, no payment is required the process finishes at this step.

ClaimValAnalyse	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
Result	staffers result
BadPackaging	true, if stuff was bad boxed
Amount	value of the package
Approval	true for paying compensation
Damaged	true, if damaged could be detected
ToLate	true, if delivery was to late
Lost	true, if package was lost
TimeStamp	time when the event occurs

Table 13: ClaimValAnalyse event

3.2.6.4.4 Further Investigation

If the value of a package is over a defined threshold, further investigation is necessary to detect the exact value. An independent investigator defines the exact value of the damage. For the investigation the package itself or the account for the transported items are necessary to appoint the value of the claim.

ClaimFurtherInvest	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
Result	independent investigators result
Amount	value of the package
SupposedAmount	investigated value of claim
Approval	true for paying compensation
TimeStamp	time when the event occurs

Table 14: ClaimFurtherInvest event

3.2.6.4.5 Settle Payment

If the staffer decides DHL is responsible for the error and the payment is approved, compensation payment has to be fulfilled. The “SettlePayment” event signals the payment.

SettlePayment	
BPEL_InstanceID	identifies every instance of the business process
ID	identifies the content of the packages
RFID	RFID identification number
Name	recipient name
Amount	value of the package
SupposedAmount	investigated value of claim
CheckPaid	true, if check was cashed
TimeStamp	time when the event occurs

Table 15: SettlePayment event

3.2.6.5 External Events

Beside the business process events, there are also external events that are part of the prototype. These events are provided by different event sources, e.g. weather information from metrological office or traffic data. In real business applications, developers do not have any influence on the content or the format of the

information provided. The data is provided by other systems or other organizations. To simulate this case a file input stream is used for external event types. The content is provided by a comma separated value file (*.csv). This format is the most common type, besides XML, for data communication.

3.2.6.5.1 Weather Events

The weather takes influence on the shipment process. So the weather stream is used for forecasting shipment failures and too late shipments. For the sample case the event type provides just basic information, which is needed for the simulation.

WeatherData	
Zone	weather zone (is also transport zone)
Date	date when the forecast is valid
Forecast	information about the weather
Period	period the forecast is valid
InfluencePlain	influence factor for plains
InfluenceTruck	influence factor for trucks
InfluenceVan	influence factor for vans
TimeStamp	time when the event occurs

Table 16: WeatherData

3.2.6.5.2 Traffic Events

The current traffic situation also has influence on the shipment. That can be congestion on the highway which has influences on trucks as well as vans.

TrafficData	
Zone	transport zone
Date	current date
Information	traffic information
Period	period the forecast is valid
InfluencePlain	influence factor for plains
InfluenceTruck	influence factor for trucks
InfluenceVan	influence factor for vans
TimeStamp	time when the event occurs

Table 17: TrafficData event

3.2.6.5.3 System Events

The whole shipment, investigation and claim process is based on computer technology. These systems also have influence on the business process. E.g. if the system providing RFID events is down, no RFID events are generated. If CEP

does not retrieve information about system errors, it would detect a shipment error. If CEP knows about the errors in the RFID system, a potential shipment error can be excluded.

SystemData	
InfluenceOn	influenced processes
Date	current date
InfluenceShipment	influence factor for shipment
InfluenceInvest	influence factor for investigation
InfluenceClaim	influence factor for claim
InfluenceRFID	influence factor for RFID
TimeStamp	time when the event occurs

Table 18: SystemData event

3.2.7 Inheritance of Events

According to para. 3.2.5, all events occurring within a SOPERA service, have the field "`BPEL_InstanceID`" for assigning the events to the BPEL process. The field "`TimeStamp`" also occurs in every event and is necessary for further processing like the correlation. To avoid defining this fields again and again, the concept of generalization is applied. That means that instead of defining fields for every event, a super-event exists and has only to be extended with additional fields. This concept is equal to the concept of the inheritance in the discipline of Object Oriented Programming (OOP) and also allows a classification of events into subtypes according to [LUCK 07; p. 151; 3rd chapter]

A similar solution has already been implemented in the event processing language RAPID-EPL. This EPL provides so called "actions" which can be used for implementing the idea of inheritance. "(…) A action declaration specifies a subtype of events. An action declaration has the format **`action`** *`identifier`* (*`list of parameter declarations`*); where the identifier is the action name and the list of parameter in parentheses declares the tuple of data in the events. The parameter list is a list of declarations that consist of the type of a parameter followed by the name of the parameter. The predefined attributes are always implied members of the list of parameters and are not explicitly declared (…)" [LUCK 07; p. 151; 4th chapter].

A similar concept was published in a research paper by the IBM Research Laboratory [ADBO 07; p4]. It describes a generalization and association of an event type definition.

Fig. 23 shows subtypes of events and the concept behind. An event type "`SoperaEvent`" is specified which includes the parameters "`BPEL_InstanceID`" and "`TimeStamp`". These predefined attributes do not have to be explicitly declared in the sub events "`ShipmentEvent`" and "`InvestEvent`", because of the inheritance. In this case, the events "`RFID`" and "`TollException`" are defined on the lowest level. These events automatically inherit all attributes defined in a higher level.

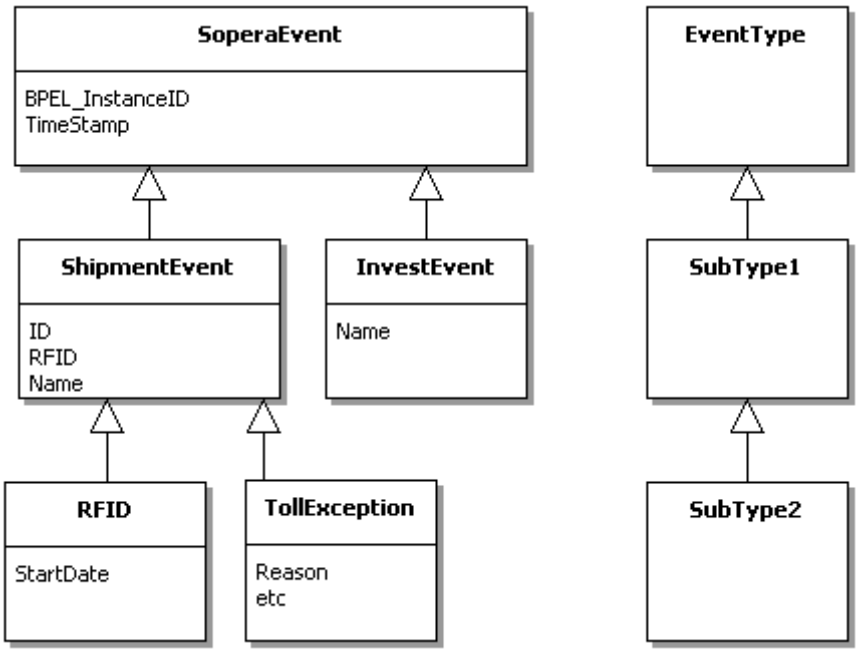


Figure 23: Subtypes of events

The concept allows the possibility to structure events hierarchically. It is also the basis of a better maintenance and extensibility, because if changes in a higher subtype level occur, the changes affect lower levels. E.g., if „`ShipmentEvent`" is extended with the attribute "`Address`", it is also available in the "`RFID`" and "`TollException`" events.

For the prototype, the integration of hierarchically modeled events in SOPERA is not realized in the timeframe of this thesis. But for future real business

applications, it is mandatory to use inheritance of events. Otherwise the complexity of the events increases and the system becomes hard to maintain.

3.2.8 Test Cases for Simulating Potential Scenarios of the Business Process

For the purpose of simulating potential scenarios for the example business process, it's necessary to define test cases. This chapter describes the basic test cases which can occur in the business process, including the scopes of shipment, investigation and claim. By means of these test cases, the capabilities of Oracle CEP and BAM shall be illustrated. Only events fired by the business process are part of the description of the test cases. These events are inserted in the description in brackets/font `Courier`. External events caused by traffic or weather as well as any possibilities of event correlation, analyzing and reacting on these events are not part of this paragraph. Startup events of the services, which occur in the case of a service call, are not mentioned explicitly, because they are not necessary for a better understanding of the test cases and would even make the description unnecessarily complex. An exact explanation of the startup event can be seen from para. 3.2.6.1.

The aim of the test cases is not to create a case for every situation which could happen within the shipment of a parcel. Rather the proper working of the business process, the services, as well as the most important event types for CEP and BAM shall be tested according to the definition of the business process in para. 3.1 and the basic events in para. 3.2.6.

3.2.8.1 Definition of the Term "Test Case"

A test case, according to [GERR 06; #190] is defined as "(…) a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement (…)".

"(…) A test case can be seen as a set of conditions or variables under which a tester will determine if a requirement or use case upon an application is partially or fully satisfied. It may take many test cases to determine that a requirement is fully satisfied. In order to fully test that all the requirements of an application are met, there must be at least one test case for each requirement unless a requirement has sub requirements. (…) Written test cases should include a description of the

functionality to be tested, and the preparation required to ensure that the test can be conducted. (…) Written test cases are usually collected into test suites (…)" [WIKI 07].

According to this theory of software testing, written test cases are used, which include a description of the functionality to be tested and the required event types and their instances, so that the test can be conducted.

With the following test cases, the business process and the services are tested whether they are properly working (Test 1) and if the expected events are sent. The resulting events are also the input for testing CEP and BAM (Test 2). Both, Tests 1 and Test 2 are also used as a kind of a show case.

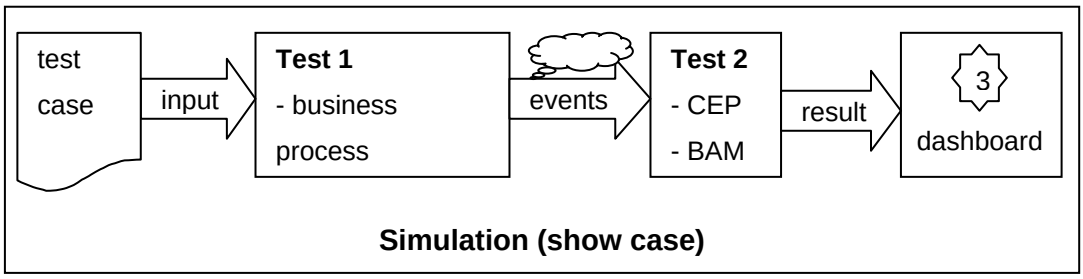


Figure 24: Test cases and simulation for show case

The show case is used to present the power of the product combination SOPERA and Oracle BPEL/CEP/BAM, e.g. in a sales workshop. In this context, a show case contains several test cases which can be started during the presentation. The resulting events are correlated or transformed in CEP/BAM and the results are presented in the dashboard. So the sales man can present the customer the power of the products on a running system instead of just clicking through a power point presentation.

3.2.8.2 Test of the Business Process (Test 1)

For the first goal of testing the implementation of the business process and its associated IT-services, typical scenarios of a shipment process, which occur in real life, are described. The focus is to ensure that all branches in the business process are tested. Also, it is necessary to ensure that the test cases in combination fire all events defined in para. 3.2.6. For the path test on the basis of the business process shown in para. 3.1, all relevant branches have to be identified and to be considered when defining the test cases. Therefore, the flow in the business

process has to be controlled by parameters that allow steering on which path has to be taken in the business process.

For this purpose, variables from a XML-schema are used. The variables are assigned depending on the selected test case after starting the business process.

Example:

In real life, a staffer checks who is responsible for the blame. In our case, there is no staffer who tells us the result of a claim. So, the result of who is responsible for the blame has to be defined in advance within the test case. The parameter `claim.responsibility` is assigned with a value, which controls the further flow. To control that DHL is responsible for the blame, the parameter `claim.responsibility` has to be set to the value "`DHL`", as shown in fig. 25.

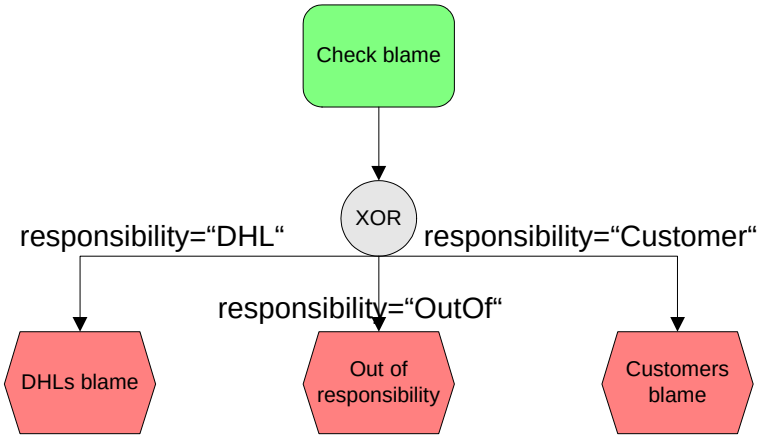


Figure 25: Parameter for controlling the flow

This can take place either via the user interface of Oracle JDeveloper as well as by editing the BPEL source code and can be seen in fig. 26.

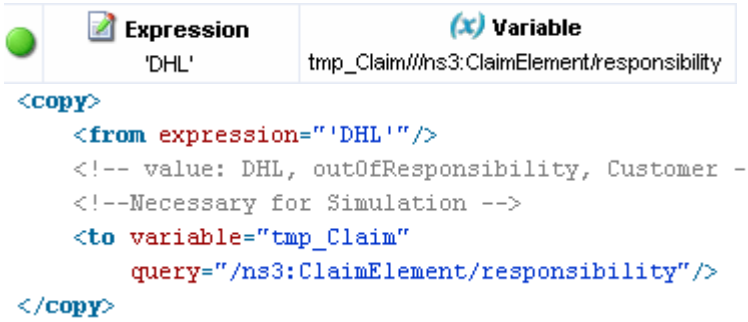


Figure 26: Assignment of test case parameters

All available parameters to control the flow can be seen in fig. 12 and fig. 13. The parameters have to be set directly in the BPEL process. There is an assignment for every test case within the scope "`Testcases`", where all settings regarding the test case are carried out. It is also possible to control values of events or if an event shall be fired or not. An explanation about every parameter, the possible settings and its effects can be seen directly in the assignment sections of every test case in the BPEL process.

Depending on the settings and the values of the parameter within the test case, events are sent. It has to be verified if the sent events are the expected ones. After having verified that the business process, the services, and the fired events work properly, this part of testing is finished. The output events as the result of "Test 1" are now used as input for testing CEP and BAM ("Test 2").

3.2.8.3 Test of CEP and BAM (Test 2)

The second goal of the test cases is to provide events for their processing by the components of CEP and BAM. The test cases defined in the following provide the same events for every started instance of a test case. Therefore, it can be ensured that the results in the CEP and BAM components are equal for every defined test case. A test case has to be started by the BPEL engine and the BAM dashboard shows the same results. Also, the CEP engine has to react in the same way and the BAM component has to report the same results regarding event processing and service monitoring. Only the values regarding the execution time differ, because the execution time of a business process instance depends on the hardware and on the system capacity.

3.2.8.4 Use Case Diagram

The following use case diagram gives an overview about the roles within the test cases.

A use case diagram shows the external behavior of a system from the perspective of users, called actors, by showing the relationship to each other. The purpose is to give a graphical overview of the functionality provided by a system and the dependencies between users and use cases. [JECK 04; p. 175]

To give a better understanding of the test cases, fig. 27 shows in an abstract way, the basic roles and functions of the business process.

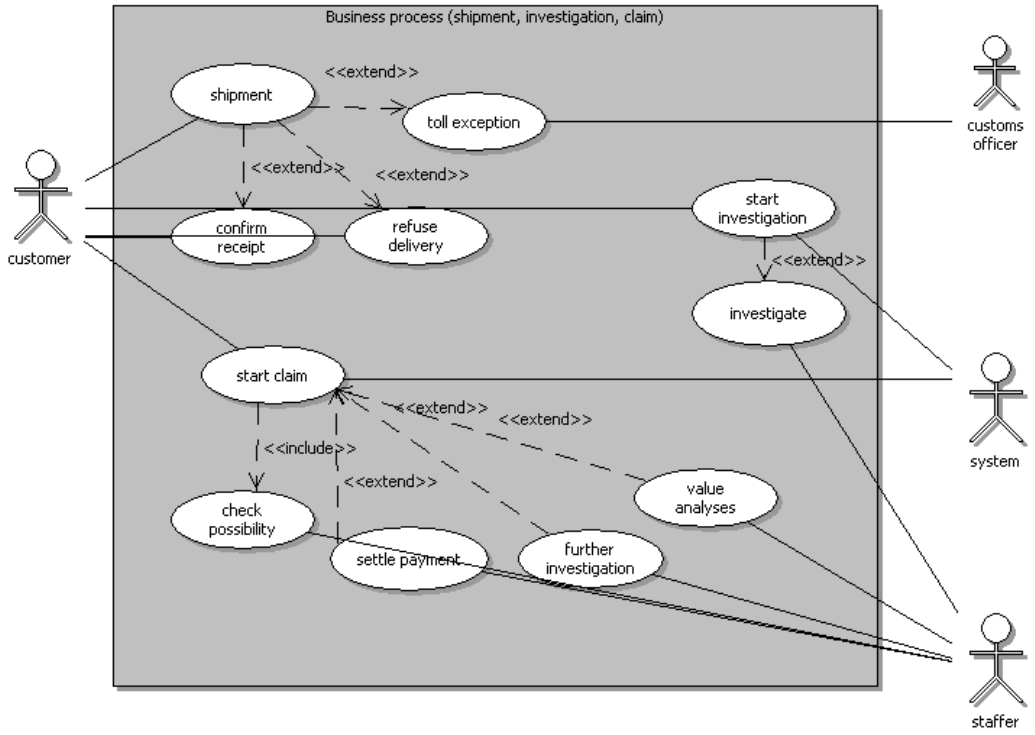


Figure 27: Use case diagramm

3.2.8.5 Test Case 1 (Damage During Shipment)

The first test case concerns shipment and claim. The following situation happens during shipment. Mr. S. wants to order a well known type of a notebook from the United States, because it is much cheaper than in Germany. The online shop also offers transportation into other countries and initiates shipment. The first station is the transport by plane (`RFID`). After the package arrived in Germany, the customs inspection opens the package to control the content and the clearance papers (`TollException`). After this procedure, the shipment continues with the transport by truck (`RFID`). During transshipping, damage on the package is noticed (`Damage`), but nevertheless the shipment continues by van (`RFID`). As the package arrives, Mr. S. refuses the acceptance of the package because of the damage (`DeliveryRefused`). To get his money back, he complains at DHL and starts the claim process. The first step the staffer does is checking if a claim is actually acceptable (`ClaimPoss`). Because the claim is acceptable in this case, the staffer has to figure out who is responsible (`ClaimRespons`). The damage is

the responsibility of DHL and so, the value of the package and the compensation has to be detected (`ClaimValAnalyse`). The value of the notebook is in excess of a defined threshold, so the staffer is not able to make a decision about payment and has to initiate further investigation of the package. After having finished detailed investigation, the expert decides that it was the responsibility of DHL and DHL has to compensate (`ClaimFurtherInvest`). The customer receives a check which has to be paid (`SettlePayment`).

3.2.8.6 Test Case 2 (Customs Inspection Error)

This test case concerns shipment, investigation and claim. Mrs. H. orders a couple of traditional spices from an internet shop in China, because it is not available in Germany right now. The insurance for the package would cost more than the whole order, so she decides to ship the package without any insurance. The shipment starts by sending the package to Germany by plane (`RFID`). The customs inspection opens the package because it was suspected, that drugs were shipped (`TollException`), but the shipment can be continued without further problems regarding duty. But during transshipping to the truck on the airport, damage was noticed (`Damage`), but the shipment is continued (`RFID`). During transport by van no `RFID` event is received by the system. That is the reason why the investigation is started automatically (`InvestStart`). Despite all the efforts, it is not possible to find the package. Therefore, the investigation has been completed with the result, that the package is lost (`InvestResult`). Mrs. H. complains and wants DHL to pay compensation. That is why claim is initiated and the staffer reviewed, as a first step, whether the complaint is basically acceptable (`ClaimPoss`). It is noticed that the transported food does not have any insurance for the package, so DHL decides to decline payment for the lost (`ClaimRespons`).

3.2.8.7 Test Case 3 (Too Late Delivery)

The third test case describes a common situation of a too late shipment. All three parts, shipment, investigation and claim are part of this test case. Mr. E. orders computer hardware for his notebook from a computer reseller in the United States by express. He needs the ordered hardware within the next three days. Therefore, Mr. E. makes a delivery time agreement with DHL. The shipment of the package completes without any accident, but is not delivered in time. So, Mr. E. decides to

start tracking and investigation, because he wants to know, where the package is located. The first step of the investigation process is to check if investigation is admissible (`InvestPossible`). Because the expected delivery date is exceeded, the investigation is started (`InvestStart`). During the investigation, the staffer tries to figure out where the package currently is and what the problem is. The result of the investigation is that the package is still on shipment. So, the investigation is closed and shipment continues despite the exceeded delivery time (`ContinueShipment`). Shipment is done by plane (`RFID`), truck (`RFID`) and van (`RFID`). When the package arrived, the delivery is refused because of too late shipment (`DeliveryRefused`) and Mr. E wants DHL to pay compensation for the too late shipment. So, the claim process is initiated. In this case, it is obvious that the claim is admissible (`ClaimPoss`) and that DHL is responsible for the blame (`ClaimRespons`). In the next step, the staffer analyzes the value of the blame (`ClaimValAnalyse`) and starts further investigation because the amount of the compensation exceeds a defined threshold (`ClaimFurtherInvest`). The investigator has to define the exact value of compensation and has to approve the payment. After the approval, the compensation is paid by check (`SettlePayment`).

3.2.8.8 Test Case 4 (Shipment Without Errors)

The fourth test case simulates the most common shipment situation. The package is delivered without any problems from the sender to the destination. Mrs. W. wants to send a Christmas present from Australia to her family in Germany. The package starts shipping by plane from Australia to Germany (`RFID`). After having arrived at the Munich airport, it is transshipped and goes on by truck (`RFID`) before the shipment continues by van (`RFID`) to her parents. Her parents confirm the reception of the package by signature (`FinishShipment`). Shipment is finished successfully.

3.2.8.9 Test Case 5 (Claim Without a Reason)

The fifth test case describes the situation when a package is delivered without any problems, but the customer complains without any reason. Mr. T. ordered shoes from a mail-order company in Berlin and the package should be shipped to his address. The shipment does not include any delivery time agreement or insurance.

At first, the shipment has been started via truck (`RFID`), then the package is shipped further via van (`RFID`) to Mr. T. who signs the receipt (`FinishShipment`). Afterwards he wants DHL to pay compensation, because the package was in his opinion delivered too late. DHL checks the facts and refuses the compensation payment (`ClaimPoss`).

3.2.9 Conclusion

The technique, which is done with the test cases, can be seen as traditional tracking. Tracking in this context means, try to find out, what was happening within an instance of the business process by supervising business events, as well as *SBB events*. That allows in case of an error, finding the exact reason for an error. That can be on the one hand a technical error caused by an error in a method of a Web service or on the other hand a business error e.g. an error during shipment. Problems are pictured in a BAM dashboard that enables “drill down” functionality for locating the exact reason of an error (para. 8.2.2.1).

In the next step this basic events are correlated to a complex event which allows detecting errors in real time. Creating a complex event out of basic events is described in para. 9.1.

Basic events within the prototype are provided in two different ways. On the one hand, non intrusive – shown in fig. 28 – that means without code modification within existing services and on the other hand intrusive – shown in fig. 29 - where code of the services containing the business logic has to be changed.

3.2.9.1 Non Intrusively Generated Events

SBB events are provided automatically by the SBB, as described in para. 3.2.2.1 and just have to be enabled with the service policy (para. 4.3.2). The mechanism SOPERA provides is the right approach because it is not necessary to change code within the business logic of a service to provide events. But for achieving a consistent event structure, SBB events need to be modified according to a basic event as defined in para. 4.2.6. This would lead to a structure where a common identifier – the attribute “`BPEL_Content`” can be used for correlation within CEP and BAM.

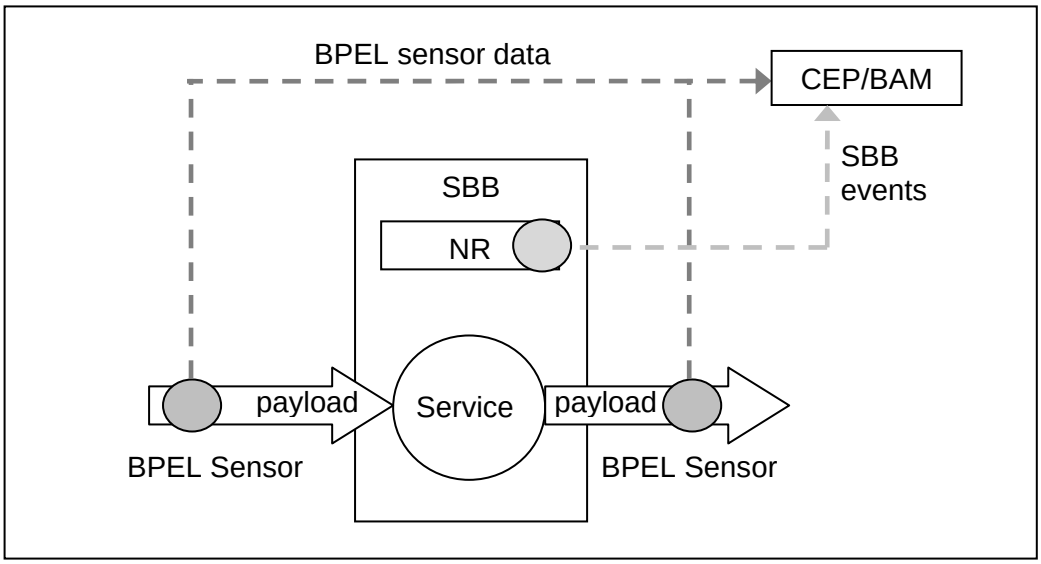


Figure 28: Non intrusive generated events

BPEL sensors (para. 3.2.2.3) follow the non intrusive concept. Data which is sent by using sensors provides information without changing the business logic of the service. The sensor allows tapping data directly from the BPEL process or the payload of the invoked service, as described in para. 5.2.2.

The non intrusive method is recommended for all real business applications to extract events for monitoring out of the business process. The reason for that is no modification of the business logic is necessary. A financial institution for example will not allow any changes in the existing business logic to implement CEP or BAM.

3.2.9.2 Intrusively Generated Events

Another possibility is to generate events directly within the service. Doing this, the business logic of the service has to be extended with the functionality to fire events out of the service as described in para. 4.2. This solution can be implemented for research purposes or for a prototype but is not recommended to be implemented in a real business application.

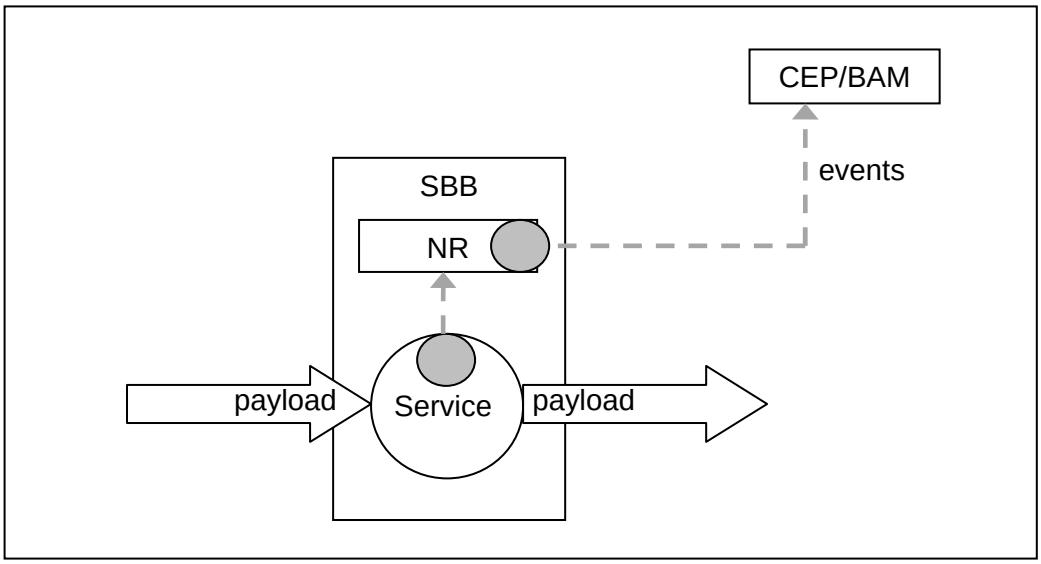


Figure 29: Intrusive generated Events

3.3 Streaming of Events

3.3.1 Difference between ESP and CEP

Before the next paragraph explains how to model event streams, this section discusses the main differences between event stream processing and complex event processing.

According to the article “What’s the difference between ESP and CEP” [LUCK 07], written by David Luckham, the main difference between event stream processing and complex event processing is that event stream processing deals with streams of events where complex event processing deals with an event cloud.

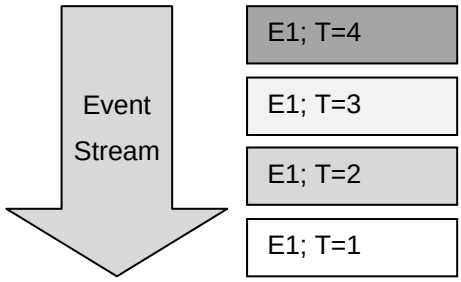


Figure 30: Single event stream

Unlike a stream an “(…) event cloud is the result of many event generating activities (…) “, which means basically that a “(…) cloud might contain many streams (…)”. A single “(…) stream is a special case of a cloud (…)” [LUCK 07].

The graphic illustrates the order by time within a single event stream. All events are generated by a single source and arrive at a single target. This helps in the creation of “algorithms” that “(…) can be very fast (…)”. Because they compute “(…) on events in the stream as they arrive, pass on the events on the next computation and forget the events” [LUCK 07]. The next figure explains an event cloud according to the definition of David Luckham. The figure consists out of a subset of several event streams to understand the relation between event streams and the event cloud. The cloud surrounds all that streams.

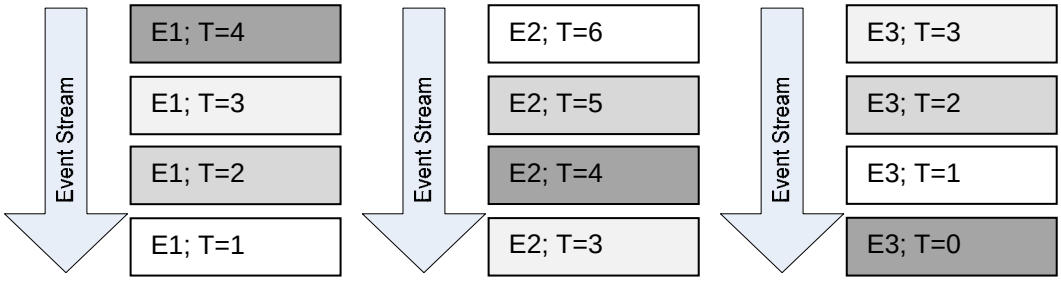


Figure 31: Event cloud pictured as multiple streams with different order in time

This example points out that when the first event from the first event stream, E1, enters the system there is no way to correlate it with the incoming event from the second event stream, E2, with time stamp T=1 because in ESP this event has already left the system.

This is the cause for a caching mechanism within complex event engines. As a result of this discussion, event stream processing and complex event processing have event streams as their common source.

3.3.2 Introduction in Event Stream Modeling

Event stream modeling deals with the topic, how events are going to be sent from the producer to the consumer. Several streaming models are thinkable. As a first step, a simple example without any forks and intersections in the event stream provides an overview.

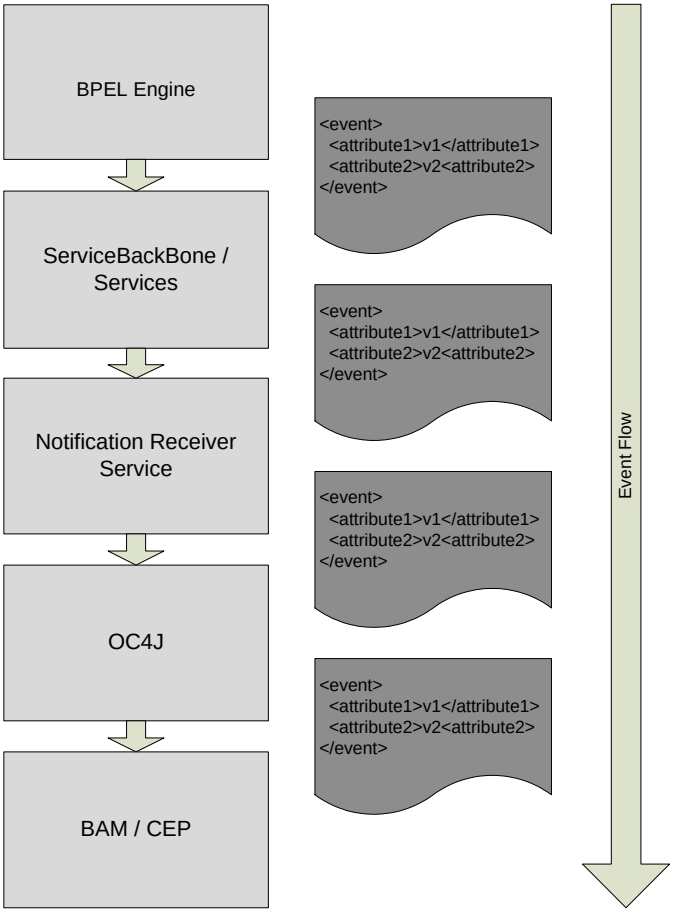


Figure 32: Event flow over all components

The picture should give the principal understanding of how event processing works and what event processing wants to achieve associated with the example implementation on SOPERA, Oracle CEP and BAM.

While the process instance is running events are generated. The data that is included in this event type is related to the business data (para. 3.2.2.2). The engine processes the data and puts it onto the services in the SBB. At this point, additional SBB events are created related to technical information, according to their definition. This information is e.g. about when a service was started or if any exception within an operation was thrown.

After the events are all on the service bus, they are going to be collected by the NR. The collection of the events is performed to send them out for further processing.

The consumer of the NR is the OC4J server. This server contains a JMS service that offers, according to Sun's JMS specification [SUNM 02], the ability to act as

distributor. This whole topic is described in the spec under the section “JMS publish/subscribe model” [SUNM 02] as well as in the implementation section (para. 7.2.2) of this thesis. Its task is to send the events out in the event stream for the real-time BAM on the basis of CEP/ESP. Over there they can be further processed for analysis and reaction.

In reality there is not only a single event stream but many streams that need to be processed. To handle these streams it is necessary to think about all components that deal with these event streams to find the best solution. The next chapter describes some points of view, how this handling could be realized.

3.3.3 Event Stream Processing Patterns

Until today, there is no standard reference book for how to structure event streams unlike in how software design patterns describe how to implement a software architecture. The references for software patterns are e.g. Frank Buschmann et al. “Pattern-Oriented Software Architecture” [BUSC 96] and Erich Gamma et al, “Design Patterns Elements of Reusable Object-Oriented Software” [GAMM 95]. These books can be used in software development as cookbooks that describe how to create a meal.

In contrast to these software pattern descriptions, this section describes how to create event streams over several components. The structure of this section is adopted from the JEE pattern description of Adam Bien [BIEN 07]. In this book Bien describes in a practically way how to design enterprise applications with the Java Enterprise technology. According to that goal on developing enterprise applications, this section explains how to design event streams in the context of the software components SOPERA and Oracle BPEL, BAM and CEP.

For better reading, this section is written in an adopted structure of Adam Bien and is explained in the next few lines:

- Essence:

  The essence provides a rough overview how events are processed in the pattern. This is clarified by an overview picture at the beginning of each description.

- Context:

  The context points out which portions on the business side can be reflected through the design as well as the technical complexity to realize the pattern.

- Application Recommendation:

  The application recommendation describes in which environment the design can be chosen and explains the constraints on the business side as well as on the technical side.

- Example:

  The example section provides a simple example realized over all components with BAM as the consuming component. BAM was chosen for this because of its graphical modeling features. The concerns coming up with the example implementation can be applied to any additional analysis component.

3.3.4 Asynchronous Event Patterns

Asynchronous event stream patterns are based on the idea of an asynchronous communication model. Doug Comer describes in his book “Computer Networks and Internets with Internet Applications” [COME 04] an asynchronous communication in the following way: “(…) Communication is called asynchronous if the sender and receiver do not need to synchronise before each transmission. A sender can wait arbitrarily long between transmissions and the receiver must be ready to receive data when it arrives (…)”. This definition might be expanded in the context of our software components in the way that it covers only two system participants. In our context, asynchronous communication also needs to cover that not all events need to pass all components but can be published from the first node and picked up at the last without touching any other component.

In enterprise architectures, this should be the preferred design because it reduces the complexity and computation resources. A drawback of this stream design is the difficulty in the correlation of event information on the consumer side. The reason for this is that there is not a common identifier in the structure of the

events. One solution of this issue could be the introduction of a timestamp. Based on the timestamp and process information it is possible to realize a correlation.

How to achieve such an asynchronous design is explained in the first pattern description.

3.3.4.1 Multiple Streams Multiple Event Types Processing

Essence:

Multiple event types are sent asynchronously from different components to the consumer. At the consumer, they are caught and correlated for further processing. Asynchronous means in this context, that they are not sent throughout all components but connected directly to the correlation engines as well as that the receiver does not know the point of time an event comes in.

The reason why the figure has dashed arrows from the BPEL component to the SBB is to show that there is still a relation between the orchestration engine and the services. The BPEL engine is still connected to the services to orchestrate them. But the main information about the process state is directly pushed on to the correlation components.

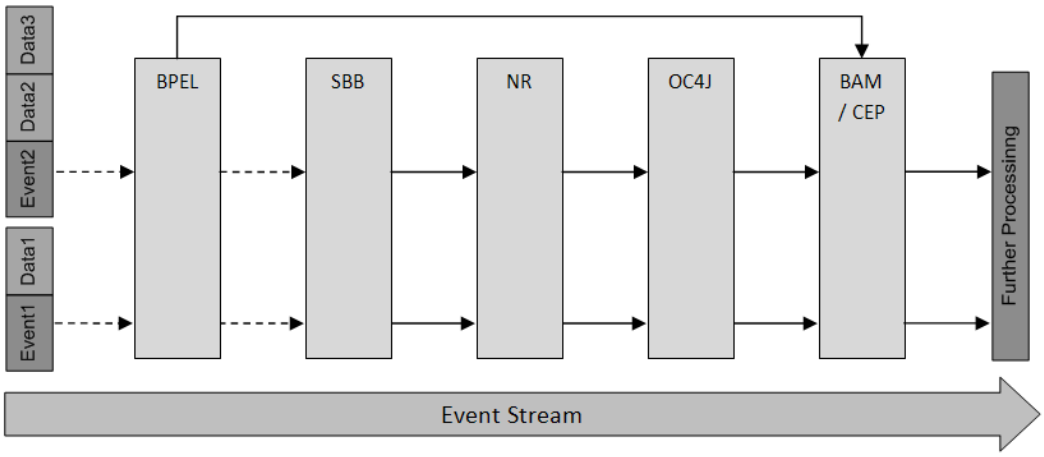


Figure 33: Asynchronous multiple streams multiple event types pattern

Context:

- The context to the business process is explicit distinguished in the design of an event. Each event type contains only the business attributes that contain the information of the executed business process step.

- The context to the business process can be expressed in the event flow. Each event stream represents an information flow of the business process.
- The structure of the event information is flexible. Adding or removing events has no influence on the existing events because there is no relation between them.
- The project needs full control over all components to realize such architecture. If one of the components is not under the control of the project, it is obvious that it incorporates is in the project as requirement.
- Many event interfaces in the event flow. Each stream between the components has one interface at each component. In contrast to a synchronous pattern (para. 3.3.5), the asynchronous pattern with multiple streams reduces the number of interfaces but there are still more than having a single stream with only one interface at each component input and output.
- Low complexity in the interfaces. Each interface has to cover only a small amount of attributes that really contain the business information of the specific event. This reduces the complexity and the chance of human error when changes in the system are required. A change also only affects one stream and not the whole system.
- Less interacting components on the event flow. By creating an asynchronous stream not all components have to interact with each other. This is also a benefit in reducing complexity and reducing human error if not that many components need to be controlled for all streams.

Application Recommendation:

The multiple streams, multiple events processing pattern in an asynchronous way is recommended when the development team has full control over all components and they really know in which way they want to aggregate and correlate the events in the CEP and BAM components. The asynchronous design is best for lowering the system interfaces to achieve a loosely coupled system. If any system

changes occur in this system, these are only changes necessary in the components nearby. All other components do not have any effects.

Another advantage of this architecture is that there are not as many computation steps necessary for a single event to reach the correlation component. This lowers the CPU utilization on the machines.

A mayor advantage of this pattern is that in case of a component failure there is still information on the correlation side that a process got started. This information might be very useful to lower the reaction time on failure.

On the other side there is a disadvantage on the event correlation side. It is more difficult to correlate and aggregate the information without having a common identifier base on the events. This can be resolved by using event generation time stamps for correlating the event information within certain time frames. This context is what CEP was originally designed for.

Example:

Multiple event types are sent asynchronously from different components to the consumer. The event types do not contain any ID for correlation. The only way to correlate these events is the introduction of a timestamp.

Event 1	
Attribute1	ASDF
Tag1	1234
Time_Stamp	1–1-08 12:00:03

Event 2	
Tag2	9876
Time_Stamp	1-1-08 12:00:14

Table 19: Event types in an asynchronous multiple streams multiple events pattern

One of the most common examples for this asynchronous pattern is the BPEL service orchestration. BPEL sends its information about the process ID directly through BPEL sensors in its own event stream to the correlation engine.

Figure 34: BPEL activity with sensor

A sensor opens the event stream directly from BPEL to a consuming component (an explanation about how to implement a direct BPEL connection to BAM as a consuming component is explained in para. 5.2.2). This avoids pushing all the BPEL related information through the whole system components and lowers the computation time on the events. The result of this event stream architecture is shown in fig. 35

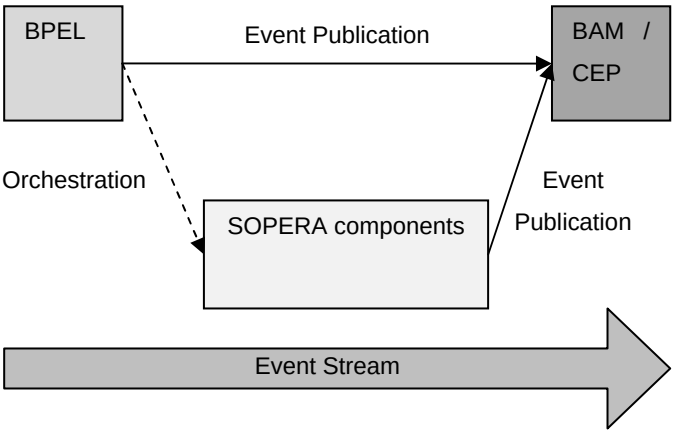


Figure 35: Overview of an asynchronous event processing

On the CEP and BAM side, this design has an impact on the correlation. Incoming events do not contain an identifier which can be used to correlate the information over the incoming events on the correlation side. One solution of this correlation and aggregation issue is a correlation by a previously introduced timestamp for incoming events. Another solution for correlating the events can be done with the complex type “BPELContent” as described in para 3.2.5.

An incoming BPEL event will cause within a certain time frame an event from a SOPERA component. If both events are not coming in the correlation within a certain time frame, a conclusion can be found referring to some issues within the system. Issues can be whether problems within the system with hardware like network delays occur or that processes within the SBB do not react with the performance they are expected to work.

3.3.5 Synchronous Event Stream Patterns

In contrast to the description of a synchronous communication model that can be found in the Oracle AQ developer’s description [ORAC 01], the term

synchronous in the context of synchronous event patterns refers not to a request-reply model but to the model of how events are processed and arrive at the correlation engine.

The synchronous event stream pattern deals with theories how events can be processed when they are passed over all components. Each sends the event after it has received an incoming event to the next node in the system. This ensures that all events arriving at the correlation engine contain a common identifier which can be used for correlation. An event contains, as attributes, the process information as well as the technical service information. The time stamp is not needed to bring process and service information in relation.

The design of a synchronous processing is not recommended in enterprise architectures because of the high computation time. Each component has to pick every event sent through the system and push it at least on to the next component.

The reason why these patterns are also considered is the theoretical approach how the event streams can be created using the recommended components and where a stream design affects the components.

3.3.5.1 Single Stream Single Event Type Processing

Essence:

A single event type containing the data of all events is sent through a single event stream over all components till it arrives at the CEP/BAM component. Over there, it is split to correlate the different event information. Afterwards the information is displayed and appropriate reactions can take place.

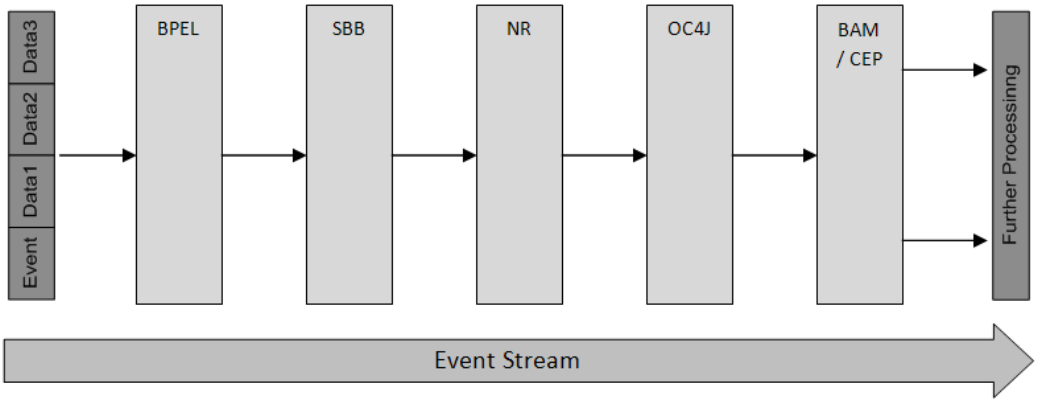


Figure 36: Synchronous single stream single event type pattern

Context:

- The context to the business process is not distinguished in the design of an event. The event attributes do not reflect the information the business process fires at the current step. The event attributes are equal for all events no matter where they come from.

- The context to the business process cannot be expressed in the event flow. The single stream does not reflect where the events are generated. Each component needs to pick up the single stream analyses, the event data, process it and push it out again on a single stream.

- The structure of the event information needs to be constant. Adding or removing event types, with different attributes than the common event type, is hard to achieve because it affects the whole system of analyzing and processing the information.

- Less event interfaces during the whole event flow. The number of event interfaces reduces. At each component there is only an input and an output interface.

- High complexity in the event interfaces. The complexity within the interfaces increases. The interfaces have to deal with a variety of different event types that are broken down to a basic event type. This common event type needs to be analyzed, split and later on put together for further analysis.

Application Recommendation:

The single stream single event processing is only recommended when the developer of the event stream only has influence on the CEP and BAM components. The single stream is seen in this pattern as a prerequisite and cannot be split in any system component processing the event stream before the correlation takes place.

This design has huge restrictions in the design of the event. An event combines all attributes over all possible event types. This means when a new event type with different attributes is created it affects all other event types as well. The context of

the event type to the business process needs to be defined at the beginning of the project.

On the other hand there is a drawback at the point of the correlation interfaces (e.g. business activity monitoring and complex event processing component). Developers need to create complex transformations to divide all events into the right channels for further processing.

The big advantage of this stream design is the reduction of the interfaces in the components before the correlating components join the event flow. As an example, this design reduces the administration of JMS (para. 7.3) topic down to a single one. Another example that points out the lower complexity of creating the streams is that the push mechanism explained in para. 6.3.1 of the NR gets easier. Only a single JMS topic needs to be addressed.

Example:

Two events are sent over all components in a single stream. It is mandatory that both events have the same data structure, although they belong to different business domains. As a consequence of this, the structure of both events looks like table number 20.

Even in this short example it becomes obvious that the overhead increases by sending empty attribute information over the network. Empty attributes still contain information about attribute names even if these attributes do not contain valuable information in terms of aggregation.

The opposite of this design with a lower network traffic rate is described in para. 3.3.4.1, para. 3.3.5.2 and para. 3.3.5..

Event 1	
Attribute1	ASDF
Tag1	1234
Tag2	

Event 2	
Attribute1	
Tag1	
Tag2	9876

Table 20: Event types in a synchronous single stream single event type pattern

Another issue appears by focusing on the interfaces that are responsible for translating and channelizing the incoming events at the components for further processing and correlating the information.

As an example, for that issue we assume that the incoming event stream arrives at the Oracle BAM component. The basic task of business activity monitoring is at this point to pick up the stream and store the data in a memory from where it can be displayed within a dashboard. To store the data, BAM uses a database as backend memory. For this, there are basically two contrary methods of resolution thinkable.

The first one picks up the stream and stores all incoming event information within a single database table no matter if attributes contain values or not. This solution violates the rules of relational database theory. "(…) The normalization rules are designed to prevent update anomalies and data inconsistencies (…)" [KENT 82]. In his research article, he describes how to achieve the normal forms. The theory about normal forms provides criteria to figure out how vulnerable a database table is against these anomalies and data inconsistencies.

Another research paper by Park and Leinen investigates the situation of having NULL values within a temporal database. The paper admits having NULL values in a temporal database is common because at the point of data insertion not all values are known. [IBRA 00, p. 477] In this example case, we are not dealing at all with a temporal database as the authors of this article describe the behavior of them. Incoming events come into the memory and are stored there for further processing, not for being manipulated afterwards. This means explicitly that the information is already known at the point of the insertion. In that case, it is to avoid having NULL values stored in the database.

Also from a practical point based on the example implementation in Oracle CEP and BAM, NULL values do not bring any advantage in querying the data for displaying them in a dashboard or creating complex events out of a combination of several basic events. The only effect that takes place is a rising complexity within the WHERE clause of the query and makes it harder to maintain the application.

A second method how to process the single event stream is illustrated in fig. 36. The idea behind this processing way is to avoid the unessential NULL values by splitting the single stream into multiple streams. Afterwards, the event information is stored in data sinks according to their event type. Data sinks are in

the example implementation seen as database tables. A detailed explanation about data sinks is given in the description about BAM (para. 8.2.1).

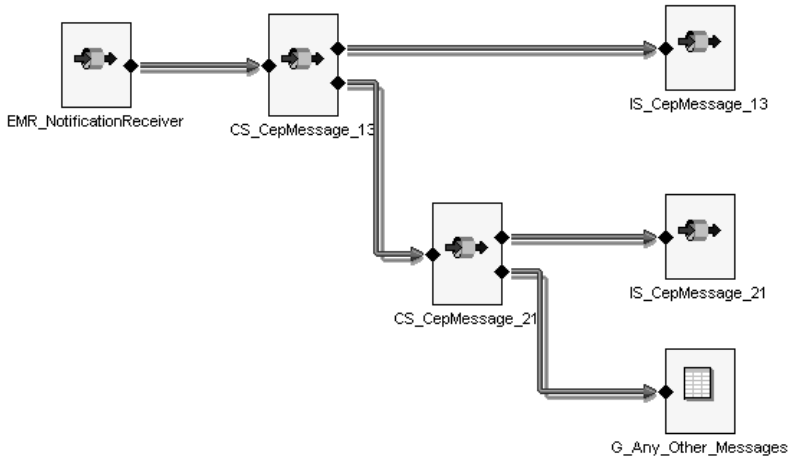


Figure 37: Example of further processing in BAM plan with two different events

The figure is captured in Oracle BAM Enterprise Link and illustrates the following scenario:

Two event types (CepMessage_13 and CepMessage_21) arrive at Oracle BAM at an enterprise message receiver (EMR_NotificationReceiver). The receiver passes the events on to a conditional splitter (CS_CepMessge_13) which checks if the incoming event is an event of type CepMessage_13. In case of “CepMessage_13” the event is passed on to the data sink (IS_CepMessage_13) where it is stored.

At this point of the explanation, it is necessary to point out that the emphasis of the graphic is on the split of the event flow and not the elimination of attributes. An advance of the storage process without NULL values is that some event transformation nodes need to be added. An option to achieve an elimination of unneeded columns is the elimination through Visual Basic scripts.

All other events besides “CepMessage_13” events are passed on to the next conditional splitter (CS_CepMessage_21). CS_CepMessage_21 decides if it is a message of the CS_CepMessage_21 or not. In case of CepMessage_21, the procedure is the same as for CepMessage_13, including the VB scripting for attribute elimination. All other events cannot be handled in this example.

This simple example gives a visual overview on how complexity increases by splitting a single incoming event stream into many.

Considering that each event type needs three nodes (conditional splitter node, Visual Basic scripting node and data sink node) plus the receiver node to be stored in a database table, the complexity increases linearly by the factor three for each event. If the system contains one event type, the number of nodes can be expressed as $(3*1)+1=4$, in case of two as $(3*2)+1=7$. Assuming that n event types are possible and x is the complexity, the complexity can be expressed in $(3*n)+1=x$.

3.3.5.2 Single Stream Multiple Event Types Processing

Essence:

Multiple event types containing the business data are sent through a single event stream over all components till they arrive at the correlating CEP and BAM components. To finish the processing, the event stream needs to be divided at this point before it enters the correlation components. The stream interface needs to be able to transform the incoming event types on the stream to pass them on to the consuming components.

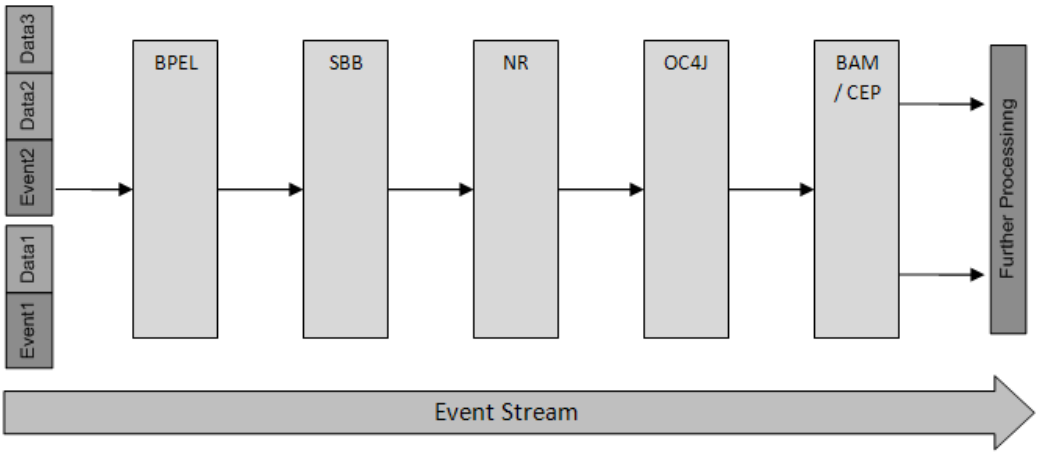


Figure 38: Synchronous single stream multiple events types pattern

The consuming components have basically the same functionality as already described in para. 3.3.5.1, single stream single event type processing. This model derives from the model in the earlier section and extends this model by the assumption of not sending a basic event in a single stream but to send multiple

event types reflecting the business objects. Business objects are e.g. RFID tags as well as a claim by a customer.

In addition to the derived pattern this pattern, introduces an alternative of event type transformation using Visual Basic coding. The transformation can also be put into the receiving component of the correlating component and executed by using XSLT technology [CLAR 99].

Context:

- The context to the business process is explicitly distinguished in the design of the event types. Event types reflect the actual attributes needed by the business process.
- The context to the business process cannot be expressed in the event flow. The single stream through which the events are pushed can not reflect the actual business.
- The structure of the event information is flexible. Event types can be modified without having any influence to other event types. The modification impacts the streaming interface at the consuming components.
- Reduction of the number of event interfaces. Each component has a single input and output interface to enable the event flow through the stream.
- Increasing complexity in transforming event interfaces. Each event needs to be transformed suitably for correlation and aggregation.

Application Recommendation:

The single stream multiple event processing is recommended to achieve flexibility in the design of events. This processing type is derived from the single stream, single processing pattern and extends it with the functionality of designing events reflecting the business problem. It is possible to add, remove or change them during the development process as well as at application runtime.

Further explanation about the event design and inheritance of events can be found in section 3.2.6 and 3.2.7. If any change in the business logic affects the design of an event, events can be adapted.

An opportunity in this pattern is the reduction of the complexity in the event stream split by eliminating the Visual Basic scripts. The complexity reduction would result by lowering the number of transformation nodes from three to one per event type. This means that the complexity coefficient is equal to two and the complexity formula can be pictured as $(2*n)+1=x$ where n is the number of nodes and x is the degree of complexity.

The previous models are also based on the assumption that all events can be passed on to the next component where they are caught in their elementary basis. Based on the experience working with the projects components, this assumption does not match the reality. Transformations need to be executed within the input and output interfaces to match the expectations of the next / previous components. After the previous pattern in section 3.3.5.1 described a way to transform the incoming events while splitting the events into a single stream per event type, this section point out an alternative by using XML, and over here especially the XSLT technology, at the incoming event interface of BAM. This alternative can also be applied to the previous pattern reducing the complexity of nodes.

The specification about the XSL transformation [CLAR 99] can be found on the website of the W3C organization. As a definition for what an XSLT transformation is and how it is achieved the document says: “(…) A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added (…)” [CLAR 99].

Based on that statement and the technical description how to achieve such a transformation which is also included in the document, it is possible to create a

transformation on any incoming XML document which is characterized being "well-formed" [CLAR 99].

There are two advantages in this design. First it provides flexibility and independence on the side of the correlating components. And secondly it allows centralizing the per event type transformation described in the pattern of para. 3.3.5.1. A drawback of this design is the increasing complexity in the event transformations interface. Increasing complexity means in this context that for each incoming event type the structure of the well-formed XML document [BRAY 06] needs to be parsed from the top level, called root element, down to the lowest level of the child elements and transformed into the format the further processing component can be interpreted. Using a transformation structure, according to the event hierarchy by applying the template structure, is one solution to get a maintainable architecture.

According to the diagram shown in section 3.2.8 a similar hierarchy can be built for the XML transformation based on the event hierarchy. Fig. 39 pictures a proposal of a transformation hierarchy. Assuming that all event types derive their common attributes from a basic event, this transformation can be executed at the highest level of the template tree. All derived events extend the transformation by applying these templates according to their event types to the basic event transformation.

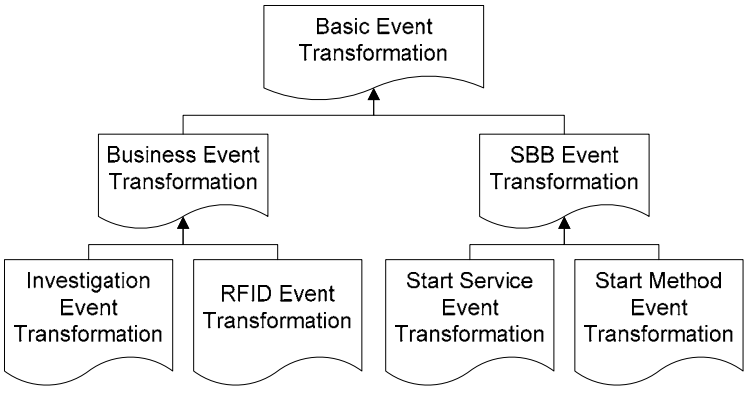


Figure 39: Introduction of an inheritance tree by applying templates in XSLT

As already mentioned, this method helps to reduce the complexity while splitting the single stream into multiple streams by reducing the processing nodes.

This single stream pattern, like every single stream model, has the advantage in reducing the number of event interfaces between the system components. Each system component needs only one input and one output interface. This lowers the administration effort according to the number of event interfaces but not the effort in maintaining every single interface.

Example:

Two events are pushed through all components of BPEL, SBB, NR, OC4J up to the correlation components CEP/BAM within a single event stream. Both events have a business distinguished design. This means that the events directly reflect the data needed in the business process. As an example for a business reflecting event structure, table 21 gives an introduction.

Event 1	
Attribute1	ASDF
Tag1	1234

Event 2	
Tag2	9876

Table 21: Event types in a synchronous single stream multiple events pattern

This event design has the advantage that it offers more flexibility in designing and adding new events. Drawbacks are that the complex split of the event stream needs to be made in the same way as already described under the section single stream, single event processing (see para. 3.3.5.1). Before the complex split is possible, an XSL transformation needs to be implemented so that BAM is able to interpret the event data and can proceed with splitting the single stream.

This XSL transformation takes place when the events of the event stream arrive at the enterprise message receiver (the receiver is pictured in fig. 37 as EMR_NotificationReceiver). When an event attends at this place, it needs to be transformation. The transformation is executed by applying templates according to the previous mentioned section.

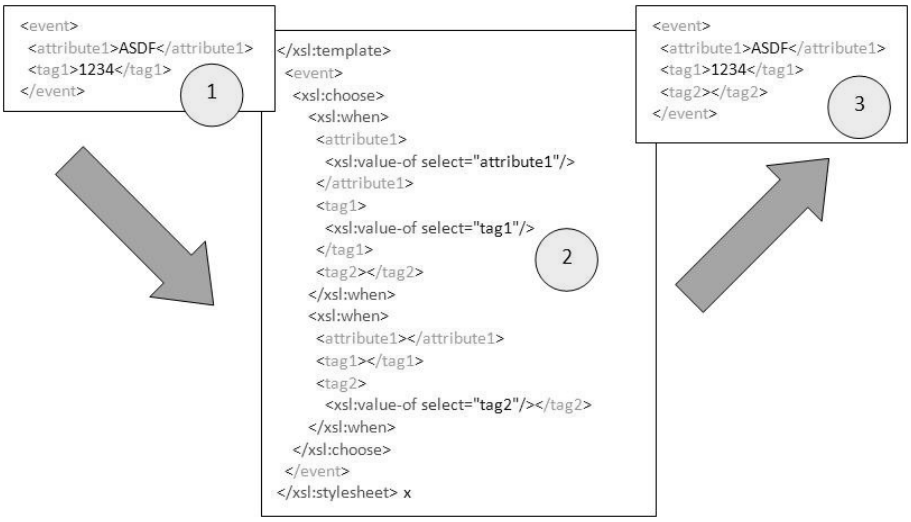


Figure 40: Basic XSLT Event Transformation Example

The example in fig. 40 shows a simple XSLT transformation where number 1 is one event of the incoming event stream. This event is picked up by a receiver and transformed at number 2. The example shows a conditional statement which can be avoided applying transformation hierarchies that take advantage of XSLT templates, to the capabilities of BAM. After a transformation has taken place, a new event is pushed on a single stream to the splitting condition already described in para. 3.3.5.1.

The single stream multiple event types pattern provides advantages in the design of events but it has to be considered if this model provides enough modularity in terms of lowering the interface complexity.

This design consideration takes place when the design decision for the project is made.

3.3.5.3 Multiple Streams Multiple Event Types Processing

Essence:

Multiple events types are sent through multiple event streams up to CEP/BAM and handled there for further processing.

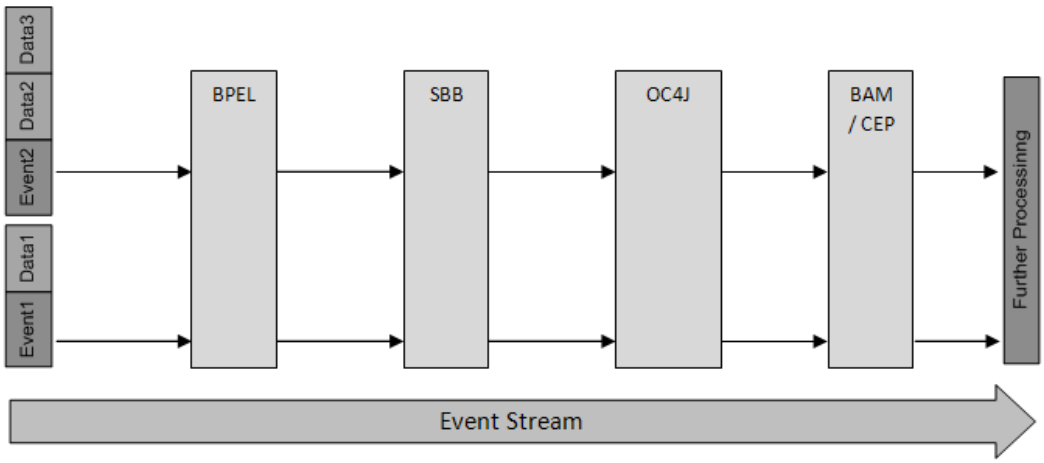


Figure 41: Synchronous Multiple Streams Multiple Events Types Pattern

Context:

- The context to the business process is explicitly distinguished in the design of an event. This means that the event explicitly reflects the business data.
- The context to the business process can be expressed in the event flow. Processing each single event as a single stream throughout the whole system reflects business activities also in their streams.
- The structure of the event information is flexible. Events can be added and removed dynamically without affecting any other event types.
- The project needs full control over all components to realize such architecture. Every component needs to be flexible in terms of catching events as well as passing them on in a single stream per event type.
- There is an increasing number of event interfaces in the event flow. Each event type needs an input and output interface at each component.
- There is decreasing complexity within the interfaces. Interfaces deal only with a single event type. This leads to the assumption that the complexity within an interface is lower than the complexity of an overall interface for all event types.

- There is less interacting components on the event flow. In case of an event stream per event type, the NR is not necessary to catch the events by fetching them from the cloud.

Application Recommendation:

The first impression of this pattern is the effort on the SBB side. Each service needs the ability to send messages itself on to the event stream per event type. This mechanism can be achieved using intrusive and non intrusive paradigm. Further explanations about these paradigms can be found in para. 3.2.9.

The mechanism needs to be able to allow the service itself to provide technical status information as well as business information. The information is passed on to an OC4J server to handle this status information in its JMS capabilities (para. 7).

The advantage of this architecture is on the business side. The event creation, modification and deletion is designed in the most flexible way so that changes within a single event type do not affect any other business participants.

Another benefit is pointed out within the correlating CEP/BAM components. Within these components there are many interfaces which contain low complexity. Low complexity can be counted as the complexity which is necessary to achieve an event transformation. A single event with ten attributes has a degree of complexity by ten. Putting a single interface per event type in the system means that each event has a complexity degree of ten. In case of a centralized event interface, the complexity will increase through the addition of all attributes over all events. As an example, ten events types with ten attributes per event type have, in a central file, a complexity of 100. 10,000 event types with a complexity of ten per event type rise up to a figure of 100,000 in terms of complexity.

The low complexity within these interfaces is the advantage. The number of interfaces that need to be administered is a drawback. On the drawback side is one solution by ordering the interfaces in a business domain structure. A business domain structure is derived from creating an event type. Pointing out an example for this means that one business domain could be, according to the example

business process (para. 3.1), the investigation department as well as the shipment department of a logistic company.

One of the opportunities this type of architecture offers is to decompose components that are not necessarily needed for the event stream. If the SBB already has the functionality to publish events directly on to a consumer through JMS, the NR component becomes redundant. This is an advantage in terms of processing time.

BAM_Event 1	
Attribute	ASDF
Tag	1234

BAM_Event 2	
Tag	9876

Table 22: Event types in a Synchronous Multiple Stream Multiple Events Pattern

On the other hand, more effort needs to be done within the SBB. Services need the ability to push events on to event publishers directly. E.g. in the case of the chosen components (see fig. 41), services must be able to publish events directly over single JMS topics within the OC4J component. A drawback of this design is the administration of many topics within the OC4J as well as the processing capabilities JMS provides.

An advantage of processing by the “multiple streams, multiple events pattern” is the detachment of the transformation and splitting procedure within CEP and BAM. It also provides - as a result - the ability to be more flexible in adding other components, besides CEP and BAM, for further processing.

The simple graph in fig. 42 points out the advantages related to the modularity of this design. In this example two events go on a JMS topic and can be consummated by a variety of analysis tools.

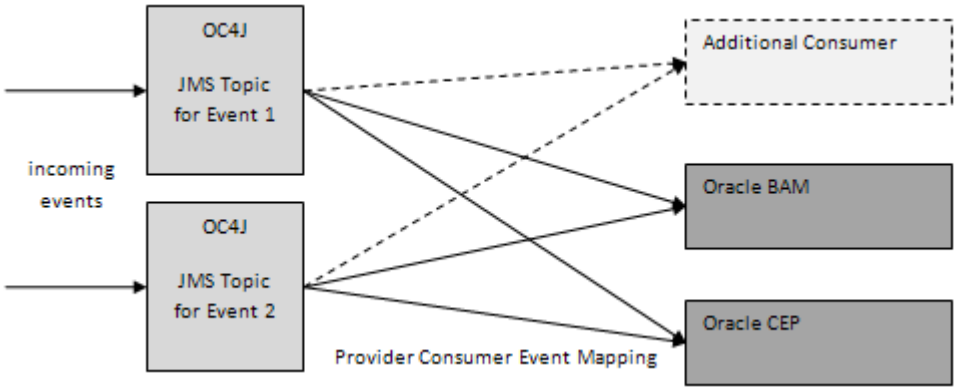


Figure 42: Extensibility of a multiple streams architecture

In case of additional need of a different analyzing technology, this design provides the most convenient plugging mechanism compared to all other patterns in this paragraph.

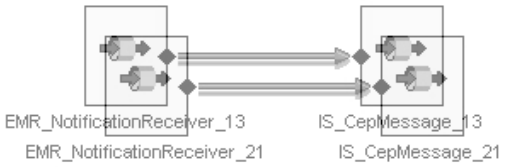


Figure 43: Multiple Enterprise Link Plans

The drawback of this design is, if only two events run throughout the system it is easy to handle the interfaces between the event provider (e.g. JMS topic) and the event consumer (e.g. Oracle BAM). The edited Enterprise Link picture (see fig. 43) exemplifies the problems for two incoming events.

On a large real application system, it is still questionable if this design can be implemented. Real business applications contain a large scale of different event types and one way how to handle this number of events is to order them to their business domain. This enables developers to keep track about the system without losing the context to the application.

Conclusion on the Patterns

All these patterns of this paragraph describe approaches how an event stream can be implemented. It depends on the technical frameworks that are used to decide which approach is the best.

In the case of the used components, a combination of two patterns is the best solution because the project team is not within the control of all components or needs some incorporation time into all components. A combination helps to give more flexibility to the team.

If the arguments of this chapter are gathered in a short list, the following example constraints are given:

- How experienced is the project team?
- How many components are under the influence of the team?
- How capable are the used technologies in terms of processing events in a real environment?
- How often do the business requirements changes?
- Which components can be added after the project was successfully implemented?

4 SOPERA Services

This section describes the implementation of SOPERA services with the focus on the integration into Oracle BPEL and the generation of monitoring information for CEP and BAM. The SOA concept of SOPERA is already described in para. 2.2. Basically a SOPERA service meets the standard of a common Web service. The differences are described in para. 2.2.3.

Para. 4.1 shows the necessary steps for creating a SOPERA service, which can be integrated into Oracle BPEL. The implementation is based on the concepts and decisions done in the paragraphs before. The focus is to create a SOPERA service, which can be orchestrated with less effort in Oracle BPEL.

4.1 Implementation of a SOPERA Service

This paragraph describes the creation of a Java based SOPERA service, the integration into Oracle BPEL and starting the business process in Oracles BPEL engine. The basic steps in developing SOPERA services and the integration of services into BPEL are described. This paragraph also discusses the integration of the NR application into the system environment. This component is responsible for providing events to Oracle CEP and BAM. A detailed explanation about NR can be seen in para. 6. It describes the implementation of one service of the business process, the "`shipment-service`". The whole business process contains much more services, which are not considered in detail. During development and integration, several problems occurred which avoided simple integration of SOPERA services into the Oracle system environment. So, it is necessary to pay attention on additional steps, which are needed to enable proper processing of SOPERA services.

4.1.1 Creating Service Descriptions and Policies within SOPERA Eclipse Plug-in

SOPERA offers a powerful Eclipse plug in, called *SOPERA Service* Editor (para 2.2), which allows the developer to create service descriptions by using a graphical user interface. As mentioned in para. 2.2.3.1, SOPERA does not use only one WSDL file to describe the services. They differ between *service description* and *service provider description*.

4.1.1.1 Service Description

The first step is to define the *service description*, which contains the description of service interfaces. For this, the XML *schema editor*, shown in fig. 44, is used. It enables the user defining methods, the parameter and attributes with a graphical user interface. On defining the name of the service, elements for request (input) and response (output) are automatically created and assigned to the defined method.

After the methods are designed, parameters have to be defined and assigned to request and response elements of that method. This step can also be done with the graphical user interface. The programmer can add *simple* or *complex data types* as parameters for the request and response method. As default the *Service Editor* creates input and output values as string. [SOPE 07a; p. 114]

The "`shipment-service`" needs more sophisticated input and output values. So, it is necessary to create input and output parameters with complex data types, which include several sub-elements. Fig. 44 shows the method "`callAllocatePackage`" and its parameters "`inAddress`" and "`inPackage`" which are *complex data types* including several sub-elements, like "`ID`" and "`street`" for example.

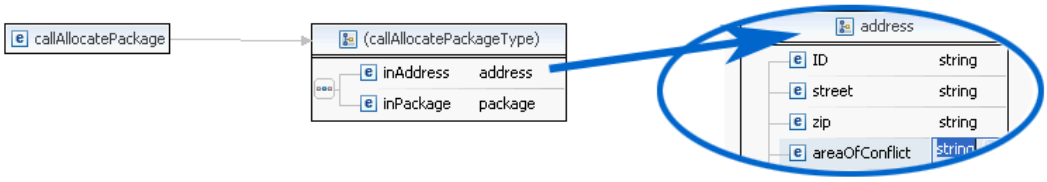


Figure 44: Service description created with SOPWARE XML schema editor

The elements - also called types - are assigned with basic XML data types like `string`, `int`, `float`, `date`, `dateTime`, etc. During modeling the interface, SOPERA generates the XML-code automatically in the background. Fig. 45 shows the user interface of the *description builder* on the left and the generated source code on the right.

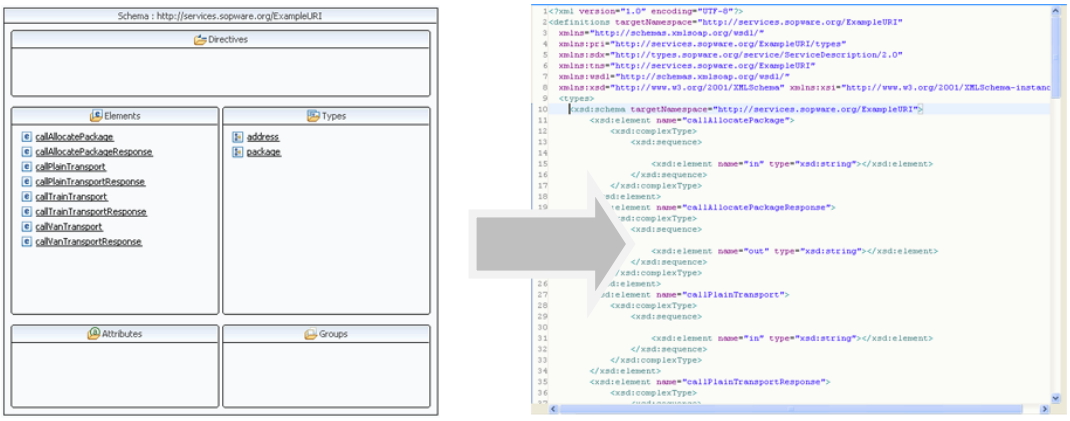


Figure 45: XML generation of a SOPERA service with the Description Builder

The *complex type* "`BPELContent`" is not part of the business logic of the service, however necessary for extracting process information out of Oracle BPEL to correlate the Oracle BPEL side with SOPERA side. That *complex type* includes the unique identifier "`InstanceID`" to identify, which BPEL instance has called the service. Without that information, the assignment between Oracle BPEL and a SOPERA service is not possible. All other parameters provide additional information about the BPEL instance, which starts the service. A detailed explanation about the concepts and why it is necessary to use the "`BPELContent`" in every service call can be seen in para. 3.2.3.

Listing XX shows the complete syntax of the complex type "`BPELContent`". Every request method must include that type as an input parameter.

```
1  <xsd:complexType name="bpelContent">
2    <xsd:sequence>
3      <xsd:element name="InstanceID" type="xsd:string"/>
4      <xsd:element name="ProcessID" type="xsd:string"/>
5      <xsd:element name="ProcessURL" type="xsd:string"/>
6      <xsd:element name="ProcessOwner" type="xsd:string"/>
7      <xsd:element name="ProcessVersion" type="xsd:string"/>
8      <xsd:element name="Time" type="xsd:string"/>
9    </xsd:sequence>
10 </xsd:complexType>
```

Listing 1: XSD BPELContent

4.1.1.2 Service Provider Description

After the *service description* is defined, the *service provider description* has to be created. It defines, as mentioned in para. 2.2 the endpoints for a service provider. That is necessary to offer a particular service with an interface defined by a *service description*. It contains the concrete binding of a service to a location.

The *service provider description* is also created by using the *SOPERA Service Editor*. After creation of the pattern for a description, the binding of the service has to be changed. There are three different types of binding protocols available, SOAP, HTTP, and SOAP-JMS. For the services of the prototype SOAP, binding is used. The property for the SOAP address must be changed to a unique service port. That is necessary, because each service needs its own port to operate. [SOPE 07a; p.88]

Fig. 46 shows the generated service provider description and the associated XML code.

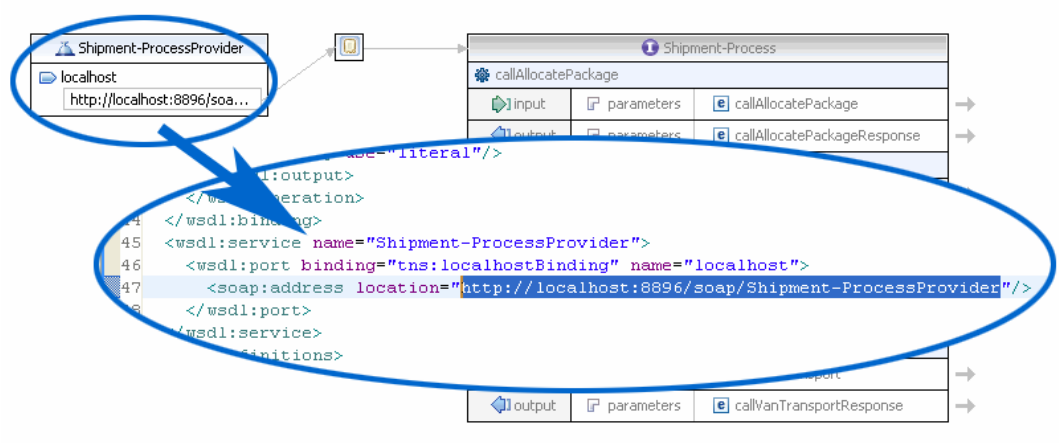


Figure 46: SOPERA Service Provider Description

4.1.2 Generation of an Universal Valid WSDL

The most common way for interaction between SOPERA service participants is using the *PAPI (Participant Programming Application Interface)*. During the process of java code generation, a consumer and a provider are created. These participants are able to interact via SOPERA specific interfaces. If only SOPERA services are used in the system environment and only SOPERA participants are communicating, this is probably the best solution.

But Oracle BPEL - and also nearly all other applications - are not able to use this proprietary way of communication. That is the reason for using a universal valid WSDL file for the integration of SOPERA services in BPEL. This solution uses common standards and allows the integration with less effort. Using the SOPERA specific interfaces is more complex, time consuming and error-prone, because every service has to be integrated with embedded Java code. That is contradicting

to the concept of BPEL, which should enable a fast and easy way of service integration and orchestration.

Using the WSDL is a common way to integrate a Web service. For this, the specification of BPEL offers a component named “Partner Link”, which allows the integration of a Web service via WSDL. “(…) A partner link type declares how two parties interact and what each party offers.” [MATJ 06; p. 75] According to the OASIS, the “(…) introduction of service reference container,” what a partner link is, “(…) avoids inventing a private WS-BPEL mechanism for web service endpoint references. It also provides plug-ability of different versions of service referencing or endpoint addressing schemes being used within WS-BPEL(…)” [OASI 07a, p. 36]. Because it is BPEL standard, Oracle also offers the possibility to use partner link for implementing a Web service in a BPEL process. The exact implementation of a SOPERA service into BPEL is described in para. 5.2.

As a consequence of the integration of the SOPERA service into BPEL with a partner link, a common WSDL has to be created. The *SOPERA service editor* provides the capacity to consolidate *service description* and *service provider description*. The result is a WS-I Basic Profile 1.1 compliant WSDL file.

It contains types, portType and message nodes from the SOPERA service description. The binding and service nodes are included from the service provider description. Namespace declarations are merged. For this, the user only has to select the descriptions and SOPERA automatically generates the *.wsdl file. [SOTS; p. 130]

4.1.2.1 Additional Step in Order to Complete the Generated WSDL File

During creation of the WSDL - based on the SOPERA *service description* and *service provider description* - in Eclipse, the *SOAP Action* HTTP-header field is not created automatically by the beta version of SOPERA. But this tag is necessary for calling the defined Web service outside the SOPERA environment. “(…) The “`soapAction`” attribute specifies the value of the *SOAP Action* header for this operation.” (…) ”For the HTTP protocol binding of SOAP, this value is required (it has no default value) (…)” [W3C 01; para. 3.4].

The following example describes how to complete the generated WSDL with the *SOAP Action*-tag. After every "`wsdl:operation`" tag defining the name of the operation, a "`soap:operation`" tag must be inserted. It includes the *SOAP Action* information, which is necessary for calling the operation by using the HTTP protocol. The information inside the tags are identically to the operation name, defined one line above. The *SOAP Action* must be defined after every operation tag, for each operation defined in the WSDL. The listing 2 and 3 show that on an example.

Before:

```
1 …
2 <wsdl:binding name="localhostBinding" type="isdx:Shipment-
   Process">
3 <soap:binding style="document"
   transport="http://schemas..."/>
4 <wsdl:operation name="callAllocatePackage">
5 <wsdl:input>
6 <soap:body use="literal"/>
7 ….
```

Listing 2: SoapAction before

After:

```
1 …
2 <wsdl:binding name="localhostBinding" type="isdx:Shipment-
   Process">
3 <soap:binding style="document"
   transport="http://schemas..."/>
4 <wsdl:operation name="callAllocatePackage">
5 <soap:operation
   soapAction="callAllocatePackage"/><wsdl:input>
6 <soap:body use="literal"/>
7 …
```

Listing 3: SoapAction after

4.1.3 Generating Java Code Based on Service Descriptions

After defining the descriptions with *SOPERA Service Editor* and performing the additional step of creating a proper WSDL, Java Code generation can be executed. The *Code Generator* uses a SOPERA *service description* and a *service provider description* as input. It generates on the basis of the XML descriptions Java source code, for the provider necessary interfaces, classes and methods to receive requests or to send responses. The source code also includes routines for marshaling and unmarshaling the incoming and outgoing XML based payload messages. [SOPE 07a; p. 140]

Code generation creates 3 Java projects based on the description files as shown in fig. 47.

- *ShippingProcess*-common: Contains commonly used objects for service consumer and service provider.
- *ShippingProcess*-consumer: Contains generated source code for the service consumer and will not be used for this prototype.
- *ShippingProcess*-provider: Contains generated source code for the service provider. This project is used for further development and implementation of the business logic to the service.

  The *custom* directory includes files which can be changed. In *ShipmentProcessProviderImpl.java*, the business logic has to be implemented.

  Files inside the generic directory should not be modified by the developer. There are interfaces, constants and the skeleton defined. [SOPE 07a; p. 67]:

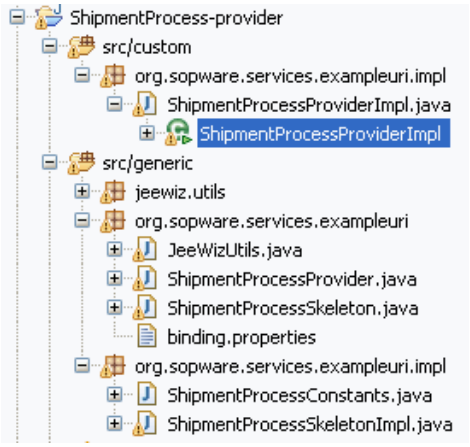


Figure 47: Packet structure of a service participant

4.1.4 Implementation of Business Logic

After the creation of the framework for the service by the *Java Code Generator*, business logic has to be implemented to the SOPERA service. The source code for the business logic depends on the service requirements and what the service distinguishes for every implemented function. This is the reason to waive a detailed description at this point.

4.1.5 Policies

The next step is creating policies for the Web service. SOPERA policies describe the complete set of QoS aspects for a service participant within SOPERA. There are 3 types of policies necessary:

- Participant policy
- Agreed policy
- Operation policy

SOPERA uses policies to ensure that a given service consumer interacts with only matching service providers. It describes the service requirements and capabilities of a service participant. Service requirements describe attributes that a service participant requires for communication. Detailed information about SOPERA policies are described in para. 2.2.3.3.

4.1.5.1 Participant Policy (Provider Policy)

The policy is based on the existing service description and the generation considers all existing operations and a proposal for corresponding operation policies can also be generated automatically or defined manually. Because it is not an advantage to create the policy manually, it is recommended to use the *SOPERA Policy Editor* for creation. The tool creates in one step both, the *participant policy* as well as the *operation policy*. For purpose of the shipment service, no changes of the created participant policy are necessary.

Within the participant policy, the service provider (as well as the service consumer) specifies its characteristics regarding calls to a particular service. It is regarded as an “implicit contract” that is offered by a participant to any communication partner that specifies a compatible policy. It contains a description of how calls to the operation have to be handled in communication with participants for each operation (method). The communication within a SOA is described in para. 2.1.4.

4.1.5.2 Operation Policy

The *Operation policy* enables technical events, also named *SBB events* (para. 3.2.2.1), which can be used for service monitoring. That functionality can be

enabled by extending the *operation policy* with *SOPERA correlation assertion and Message Tracking*. That is a non-intrusive mechanism to create events directly out of the SBB. For this prototype, these events are forwarded to CEP and BAM, but not used for analysis and further processing. A detailed description of enabling the SBB events is described in para. 4.3.2.

4.1.5.3 Agreed Policy

The agreed policy is the policy used during a conversation between a given service consumer instance and a service provider. It is automatically generated by the *SOPERA Service Registry*. That means nothing has to be done by the developer in this step. Further information can be seen in para. 2.2.2.

4.1.6 Publish (Deploy) the Service to Service Registry

The next step is to deploy the SOPERA service to the service registry, before it can be started. During development, the generated service provider runs by default as a simple Java application in Tomcat. In a production environment, that is usually not an option. In this case, it is necessary to run the service in a servlet container, for example.

But during development, it is more comfortable to start the service directly within Eclipse. It allows deploying the service descriptions as well as the policies and also starting and debugging the service. For starting and debugging the service directly in Eclipse, a launch configuration must be created. Because the service was created directly with the *Java Code Generator*, this configuration is already created and can be used. But before the service can be started in the development environment, it is necessary to register the descriptions and policies. For this it is necessary that all SOPERA components (Tomcat, LDAP, Joram) are running. Deploying can be done directly in Eclipse, by using the context menu and the “export to Service Registry”-function. After that, the service is registered in the *SOPERA service registry* and has to be started in Eclipse by using “Run…”. The console of Eclipse shows if the service starts correctly.

4.2 Generating Events within a SOPERA Service

4.2.1 Exemplified Implementation with the Event "STARTUP"

The following code snippets give an overview about the technical implementation of events within a SOPERA service using the intrusive method (para. 3.2.9.2). The example service-provider is extended with the functionality to send events via the SBB to a defined endpoint. The following example concerns with the "STARTUP" event. It is fired every time a method is started. This event provides technical information. All other events definitions provide business content.

4.2.1.1 Creating an Event Pattern

First of all, a properties file has to be created which defines the event pattern. This file is only needed once for a service and can include more than one event type. The property file is named "CEPMessages.properties" and is hooked into the same file system level as the provider implementation Java source file.

Every event has the same design as explained in the following table and needs an individual identifier behind the URL. In this case, number "10".

serverity	Gives information about the graveness of the event
category	Categorical classification of the event
message	The actual message including all attributes defining the content
params	Amount of parameters defined in the message
description	Event description (optional)
instruction	Instruction (optional)

Table 23: Event Pattern

The actual content of the event is defined within the message part. For every attribute, it is possible to define a placeholder with a bracket statement, e.g. '{1}'. The placeholder will be replaced by the value of the attribute when the event is generated.

Definition for the Event "STARTUP"

```
1 [URL].CepMessage.10.severity=INFO
2 [URL].CepMessage.10.category=PROCESS
3 [URL].CepMessage.10.message=EVENT=STARTService
   BPELInstanceID='{0}' ID='{1}' Operation='{2}'
   startupTime='{3}' ProcessID='{4}' ProcessURL='{5}'
   ProcessOwner='{6}' ProcessVersion='{7}' BPELTime='{8}'

4 [URL].CepMessage.10.params=9
5 [URL].CepMessage.10.description=Startupinformation
6 [URL].CepMessage.10.instruction=Nothing
```

Listing 4: Eventdefinition oft he STARTUP event

4.2.1.2 Creating an Event Class

In the next step, the event pattern has to be defined within a Java class, which enables simple creating and handling of the event. For this, its own Java class extending the SOPERA class "`AbstractOperationalMessage`" has to be created. This class enables the definition of every event type based on a basic event type, defined in para. 3.2.6. To do this, the functionality of the Java ResourceBundle [SUNM 03] is used, which makes the initialization of the basic events, defined in the property file much more comfortable. The Java project also has to include the "`sbb-papi-extension-operations.jar`" which provides all functionality for sending messages as well as events. This file is part of the SOPERA framework.

The basic implementation of the event class "`CEPMessage.java`" is shown in following code snippet:

```
1 import java.util.ResourceBundle;
2 import
   org.sopware.papi.extension.operations.AbstractOperationalMe
   ssage;
3 public class CepMessage extends AbstractOperationalMessage {
4       public static final CepMessage START_Service
5          = new  CepMessage(10);
6     …
7    protected CepMessage(int val) {
8       super(val);
9    }
10   …
11 }
```

Listing 5: Event implementation

The next step is sending or publishing an event within the service. That is also done by using the "`notify()`"-method defined within the class "`Operations`" which is also part of "`sbb-papi-extension-operations.jar`"

First of all, an "`Operations`" object has to be created. The instance has to be extracted out of the SBB via the local skeleton. After the object is available, it is used for sending events via the SBB. To send an event, the event type and the placeholder has to be parameters of the "`notify()`"-method. The placeholder has to be defined by an Object array, which includes all values defined in the event pattern.

The following code snippet (list. 6) is part of the service provider class and describes the basic steps:

```
1  public void fire_StartEvent(BPELContent bpel, …){
2    try{
3      Object[] oEvent = new Object[9];
4      oEvent[0] = bpel.getInstanceID();
5      oEvent[n] = …
6
7      Operations myOperations =
   (Operations)skeleton.getSBB().getEnvironment().
   getComponent(org.sopware.papi.extension.operations.Operatio
   ns.class, null);
8      myOperations.notify(CepMessage.START_Service, oEvent);
9
10   }catch(Exception e) {
11       …
12   }
13 }
```

Listing 6: Event implementation

4.3 Enabling SBB Events of SOPERA Services

This paragraph describes how to enable the technical *SBB events*. This technique is based on the non-intrusive concept (para. 3.2.9.1). Doing that, two steps are necessary. First of all, *Message Tracking* has to be enabled, to activate the *SBB events*. That is done in the policy which uses XML for storing the settings. The second step is to provide a message with the "`BPELContent`", which allows identifying what BPEL process instance caused the event. This is done by using the *Correlation Assertion* of SOPERA, which allows extracting values of the message payload by using the XPath statement.

4.3.1 SOPERA Policy Assertions

The settings for *Assertions*, like *Message Tracking* have to be defined within a *SOPERA policy alternative*. So before *Message Tracking* can be enabled, an alternative has to be created. They are not ordered and the order created by the *SOPERA Policy Editor* has no impact on the policy matching routine at run-time. An alternative is defined by using tags "`<wsp:ExactlyOne>`" for type choice or "`<wsp:All>`" for type union.

Each attribute of the message exchange is described by a policy assertion. The assertions inside an alternative are also not ordered and the order also has no impact on the policy matching routine at runtime. Assertions are used for service provider and also consumer. In this case, just the provider policy has to be extended, because the SOPERA service consumer is not used.

Assertions express Quality of Service requirements as well as capabilities of the participants, controlled with the attributes "`mandatory`" and "`optional`". The assertion setting *mandatory* expresses a requirement, *optional* expresses a capability.

SOPERA policies are divided in following groups:

- Transport Assertions
- Security Assertions
- Validation Assertions
- Generic Assertions

The assertion for *Message Tracking* and *Correlation Assertion* is part of the *Generic Assertions*. Extended information about SOPERA policies can be seen in [SOPE 07a; p 194].

4.3.2 Enabling SBB Events

The first step is to enable *Message Tracking* what is necessary to receive service information. Therefore the level of detail - the Tracking Level - has to be defined. SOPERA provides a couple of mechanisms allowing tracking of service messages while they are propagated between SBB Library instances. Such events are generated whenever a message reaches one of the several processing stages. The

events, also named “notifications”, are sent to a central repository, known as notification receiver (NR), using the management notification mechanism. Notifications are sent by the NR to Oracle CEP or BAM for further processing and analyzing (para. 6).

For specifying the level of detail that the tracking mechanism generates during a conversation, *Tracking Level Assertions* have to be defined. Following levels are possible:

detail:	An event is generated whenever a message in this conversation enters or leaves processing within an SBB library instance. This setting generates a huge amount of notifications. It should be restricted to debugging and development scenarios.
trace:	An event is generated whenever a message in this conversation is handed over from an SBB library instance to a network transport or participant application
operation:	An event is generated for each call to a service operation upon completion
summary:	Aggregated information about calls to a service operation (number of calls, statistics on processing time, number of failed and late calls) is generated.
none:	No additional tracking information is generated for service calls in this conversation.

Table 24: Trace level of SBBevents

For prototype purposes the *Tracking Level* is set to “`detail`”. To reduce the number of messages, a lower level is recommended. *Tracking Level* “`operational`” is the best solution for real operation of the software.

Every participant has to specify the minimum and maximum *Tracking Level* for the generated events. The attribute “`min`” defines the minimum level of tracking information, the attribute “`max`” defines the maximum level which has to be generated by the participant.

As default, the assertion enables message tracking with only defining the minimum and maximum value of tracking detail. This means message tracking is enabled when just one participant includes a tracking level assertion in its policy and the other participant does not explicitly prohibit message tracking at the request level. Assertions can also be used to restrict the capabilities of the participant with the attribute "`wsp:Optional`".

Its default attribute "`wsp:Optional=″false″`", is used to indicate that the participant wants to enable *Message Tracking*. The resulting agreed policy contains a *Tracking Level* assertion if the partner specifies a compatible assertion or no tracking level assertion at all.

The attribute "`ws:Optional=″true″`" is used to indicate that the participant does not want to initiate *Message Tracking* and wants to restrict the possible *Tracking Levels* that can be used. If the partner specifies a *Tracking Level* assertion with "`wsp:Optional=″false″`", the matching result is determined by the *Tracking Level* specified by both participants. If the partner does not specify a non-optional *Tracking Level* requirement, the policies always match and the resulting agreed policy will not contain a *Tracking Level* assertion.

In this case, only the service provider specifies *Tracking Level* in its operation policy. The service consumer does the service-call by using the WSDL interface. Consequently the consumer does not provide its own SOPERA policy. But SOPERA provides as mentioned in para. 2.2.3.3, a default policy which is used anytime when one participant does not define its own policy. Therefore the default policy is used during a service-call using the WSDL. Detailed information can be seen in [SOPE 07b; p. 49].

Listing 7 shows the configuration within the policy for enabling the Tracking Level with detail "operation".

```
1  <wsp:ExactlyOne>
2    <sopa:TrackingLevel min="operation" max="operation"
       wsp:Optional="false"/>
3 </wsp:ExactlyOne>
```

Listing 7: Tracking level definition

4.3.3 Configuring Correlation Assertion

Correlation allows extracting message data for correlation of the technical events out of the service payload, to assign the technical *SBB events* to a process instance. The complex type "`BPELContent`", defined in listing 1, includes the process information and is sent to the service in its message payload. The attribute, "`ProcessID`", allows identifying which BPEL process instance has invoked the service. During when the service is started, an event including the values of the "`BPELContent`" and a SOPERA "`CorrelationID`" is sent. The SBB events sent afterwards also include the "`CorrelationID`". That allows assignment of the BPEL process instance with the tracking messages. That is necessary because the SBB, which sends the SBB events, is one layer above the service (fig. 15 (2)) and has no information about the service. The following listing block includes simplified *SBB events* of a service call to explain the coherences. Line 1 shows the correlation message, including "`CorrelationID`" and "`ProcessID`". Line 2-4 shows SBB events, including service information and also the "`CorrelationID`".

```
1 CorrelationMSG: uID; time; CorrelationID; ProcessID; …;
2 TrackingMSG: uID; time; CorrelationID; Operation; …;
3 TrackingMSG: uID; time; CorrelationID; Operation; …;
4 TrackingMSG: uID; time; CorrelationID; Operation; …;
```

Listing 8: SBBevent sample

So the next step is configuring *Correlation Assertion* to extract the "`ProcessID`" out of the payload of the service request using an XPath statement. The policy can either be extended by using the SOPERA policy editor or by editing the sour code of the policy. The operation policy has to be extended with XPath statements depending on the structure of the complex type "`BPELContent`" of the payload. An example for that is shown in listing 9.

```
1  <wsp:All>
2    <sopa:Correlation name="BPELContent" message="request"
   location="receiver">
3        <sopa:Namespace prefix="exam"
       uri="http://services.sopware.org/ExampleURI"/>
4        <sopa:Part name="InstanceID"
       xpath="//exam:inBpelContent/exam:InstanceID"/>
5        <sopa:Part name="ProcessID"
       xpath="//exam:inBpelContent/exam:ProcessID"/>
6        <sopa:Part name="ProcessURL"
       xpath="//exam:inBpelContent/exam:ProcessURL"/>
7        <sopa:Part name="ProcessOwner"
       xpath="//exam:inBpelContent/exam:ProcessOwner"/>
8        <sopa:Part name="ProcessVersion"
       xpath="//exam:inBpelContent/exam:ProcessVersion"/>
9        <sopa:Part name="Time"
       xpath="//exam:inBpelContent/exam:Time"/>
10   </sopa:Correlation>
11 </wsp:All>
```

Listing 9: Correlation example

5 BPEL and Integration of SOPERA into Oracle BPEL

One goal of the thesis is the orchestration of SOPERA services to an executable business process, because only a complete business process is useful for monitoring and provides the necessary events. BPEL offers the possibility to orchestrate Web services to a business process and allows its execution. This chapter gives an introduction into BPEL and the concept behind. In a second step, it shows how to integrate a SOPERA service into Oracle BPEL and the possibility to implement non-intrusive events using BPEL sensors.

5.1 BPEL

This paragraph provides, at the beginning, an introduction about BPEL and shows in a next step the possibility to integrate SOPERA services into Oracle BPEL as well as event generation using BPEL sensors.

5.1.1 Introduction into BPEL

Information systems have become a fundamental part of companies, because they offer a company the ability to perform business operations. They can improve the efficiency of the business through automation of the business processes. The aim of the applications is, to provide a comprehensive support for the business and have to align closely with the business processes. As shown in para. 1.1, this demand is in the real world difficult to fulfill, because the business processes are usually a dynamic structure compared to the static applications used in companies. That means companies have to improve and modify, act in an agile manner, optimize and adapt their business processes to their customers needs to improve the responsiveness of the whole company. These changes reflect in mostly static applications which provide support for the business processes and causes changes of the application. These modifications have to be implemented, tested and deployed, which is time consuming (information system gap time). Because of this it is not possible to react instantly on the changes. To reduce this gap time, which depends on several factors: the complexity, the size of the modification and the state of the application needing to be modified a new technique is necessary [BPWS 07; p.6]. The aim is to provide a mechanism to react faster on changes in business processes and to reduce the information system gap. Therefore

modification as well as automation of business processes has to be done in an easier and more flexible way to handle than in the past.

Regarding to Matjaz B. Juric (University of Maribor) [BPWS 07; p.6] requirements for that are necessary:

1. Standardized way to expose and access the functionality of the applications as service:

   This functionality is provided by using the Web Service technology based on standards as described in para. 5.1.6. This requirement is also fulfilled by SOPERA services.

2. An enterprise bus infrastructure for communication and management the services:

   This functionality is fulfilled by an ESB as well by the DSB of SOPERA.

3. An integration architecture between the various services and applications involved in the business process:

   Best methods and practices for building integration architectures today is SOA, as described in para. 2.1. SOPERA is based on the SOA concept and fulfills with its open source SOA framework all requirements.

4. A specialized language for composition of exposed functionalities of application into the business process:

   For achieving that requirement a language for composition and orchestration is necessary (para. 2.1.3.7). A commonly accepted and specialized language for that is the Business Process Execution Language (BPEL) which is part of this paragraph.

BPEL (or also called WS-BPEL) is described as "(…) a language used for composition, orchestration, and coordination of Web services. It provides a rich vocabulary for expressing the behavior of business processes. (…)" [BPWS 07; p. 6].

5.1.2 Relation of BPEL to other Languages

BPEL went from a proprietary language to the de facto standard for defining and executing business processes. The following timeline (fig. 48) shows the history of BPEL and explains the influences on the language. In 2000, Microsoft developed the language XLang which describes the message exchange of two partners. IBM developed its own language in 2001 called WSFL (Web Services Flow Language) which allows the orchestration of business processes based on XML. Based on the two languages XLang and WSFL BPEL4WS was specified and published in 2002 [DOST 07; p. 204]. In 2003 BPEL4WS 1.1 was standardized by OASIS and 2007 the new specification BPEL 2.0 was standardized by OASIS. During that time several other vendors tried to develop their own languages for describing the business processes, but were not accepted by the industry (grey figures fig. 48).

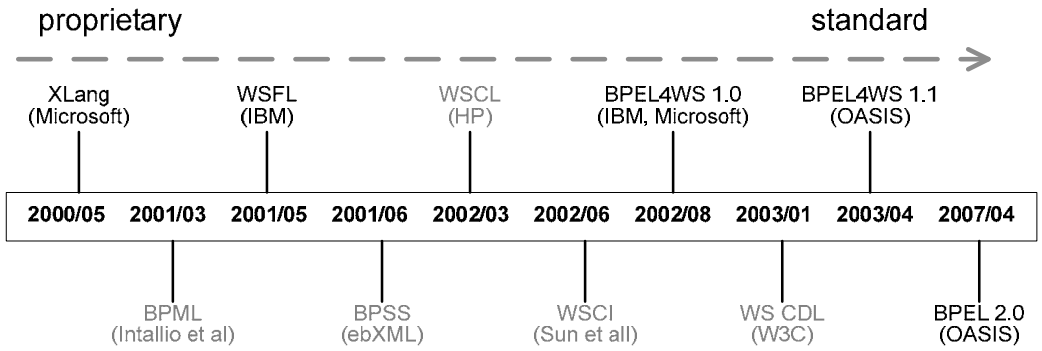


Figure 48: BPEL timeline

5.1.3 Aspects in BPEL

Regarding [DOST 05; p202] BPEL includes two different aspects for designing business processes:

- **Orchestration:** "(…) describes the executable aspects of a business process (…)"
- **Choreography:** "(…) describes the tasks and the interaction between several processes with the focus on collaboration (…)"

Fig. 49 explains the differences.

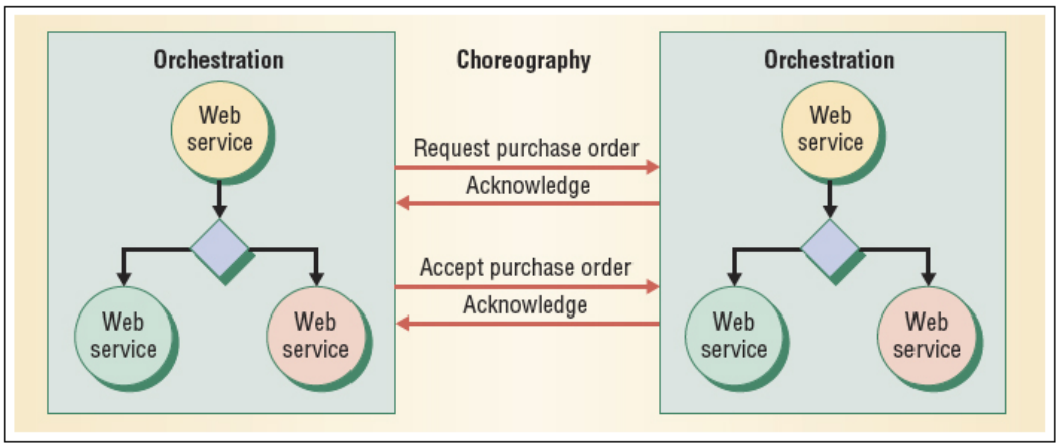


Figure 49: Orchestration vs. choreography [PELT 03]

Besides these aspects it is also possible to distinct BPEL in two ways:

Executable business processes specify the exact detail of business processes, follow the orchestration paradigm and can be executed by an orchestration engine.

Abstract business processes specify the public message exchange between parties only, they do not include internal details of process flows and follow the choreography paradigm.

In the following paragraphs, we take the focus on the orchestration aspects which allow the design of business processes, the integration of Web services and the execution of the business process.

5.1.4 Two Programming Levels

The concept realized with BPEL is based on two programming levels. Programming in the large and programming in the small. The concept behind that is not totally new, but is based on the article “Programming-in-the-Large versus Programming-in-the-Small” published by F. DeRemer and H. Kron in 1976 [DEKR 76].

Programming in the small is based on exact interface definition and a detailed description of the functions in the software component. The components are independent from the implementation and the language, because the communication is based on the interface. Therefore, these software components can be implemented parallel and independent from each other. Based on a detailed

description of the interfaces and the functionality it is also possible to outsource the programming of the components. This concept also allows replacing components with other ones, providing similar interfaces and functionality. Web services are such software components described in programming in the small. They are developed by programmers independently, based on interface and functionality descriptions. Such a service is for example, allocation of a package. Developing Web services deals with discrete fine-grained tasks [OASI 07b; p. 7]. An example of such an implementation is described in para. 4.1.

Programming in the large regards to the interaction of the small components (or Web services) described before. That means the services have to be orchestrated (para. 5.2.1) in the right order to define the interaction between each other and to create executable larger program scopes, the business processes. The focus here is not the programming, but the knowledge of the in-house business cases and the processes, to define business processes fulfilling the aims of enterprises and customers. Outsourcing is not recommended for programming in the large, because external consultants do not have detailed knowledge of the internal business processes. BPEL is focused on programming in the large and deals with combining functions in order to solve a more complex problem [OASI 07b], such as the orchestration of the shipment process.

5.1.5 Integration of BPEL in the SOA Architecture

Fig. 50 describes, according to the SOA architecture in para 2.1.3.8 and the two programming concepts in para. 5.1.4. The resulting architecture stack with the focus on process orchestration. Several applications provide their services via the ESB. The services are accessible and described (interface and functionality) in the service repository. The services provide fine-grained business tasks, as described before, and have to be orchestrated in a business process. Orchestration is done in a process designer, like the Oracle BPEL Process Designer, which is part of JDeveloper. The orchestrated business process, e.g. the shipment process, is after deployment stored in the process repository. The predefined business process, based on BPEL is executed in the process engine, like the Oracle BPEL engine, after a start action occurs. The process engine is responsible for message routing, message transformation, error handling and validation. Interaction and visualization is done in the presentation engine. That can be e.g. a web portal, a

Java application or a cell phone. Programming in the small is done within the applications and the service repository. Process repository and process engine are part of programming in the large.

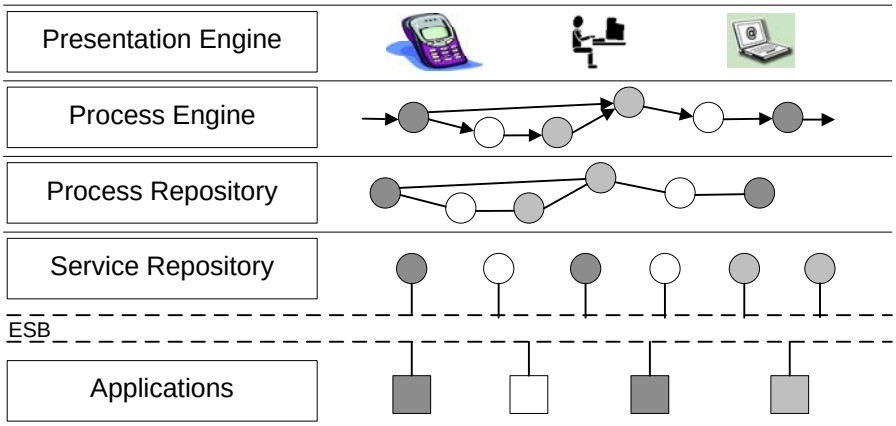


Figure 50: Architecture stack

5.1.6 Core Concepts of BPEL

The program code used in BPEL is serialized in XML and based on the standards *XML Schema, XPath, WSDL and WS-Addressing* [DOST 06; p. 207]. Specifying the version number is waved on this point because of the development and changing standards. XPath is a language for selecting nodes from a XML document, e.g. the payload of a Web service. WSDL, as described in para. 2.1.3.2, describes the interface of a Web service. A BPEL process is also a Web service and therefore also uses WSDL for describing its interfaces. WS-Addressing describes the endpoints of the services.

BPEL consists of a couple of activities [BPWS 07; p.6]. On the one hand basic activities used for common tasks. On the other hand it structures activities to define more complex algorithms.

Basic activities:

<invoke>	Invoking (synchronous/asynchronous) of a Web service
<receive>	Waiting for the client to invoke the business process through
<reply>	Generating a response for synchronous operation
<assign>	Manipulating of variable data
<throw>	Indicating faults and exceptions
<wait>	Waiting for a defined period of time
<terminate>	Terminating the entire process

Table 25: BPEL basic activities

Structured activities:

<sequence>	Defining a set of activities which are invoked in an ordered sequence
<flow>	Defining a set of parallel invoked activities
<switch>	Defining case-switch constructs
<while>	Defining loops
<pick>	Defining a number of alternative paths (not deterministic)

Table 26: BPEL structured activities

The expression <variable> is not part of the activities but also part of BPEL for defining variables within the BPEL process and the tag <partnerLink> for implementation of an external Web service, as done in para. 4.1.

Listing 10 shows a simplified BPEL source code:

```
1 <process name="Shipment" ...>
2     <partnerLinks>
3         <partnerLink name="Shipment" … />
4     ...
5     </partnerLinks>
6     <variables>
7       <variable name="Input" messageType="ns1:call…"/>
8       ...
9     </variables>
10    <sequence name="main">
11 <receive name="receiveInput" partnerLink="client"
     portType="client:BAMSensor" operation="initiate"
     variable="inputVariable" createInstance="yes"/>
12 ...
13    <assign name="assign_outcallAllocatePackage">
14   <copy>
15        <from variable="Package" query="/ns2:packageEl"/>
16        <to variable="Input" part="parameters"
         query="/ns3:callAPackage/inAP"/>
17     </copy>
18     ...
19        </assign>
20        <invoke name="AllocatePackage"
         partnerLink="Shipment"
21       portType="ns1:ZShipment"
         operation="callAllocatePackage"
22          inputVariable="Input"
23          outputVariable="Output"/>
24 ...
25    </sequence>
26</process>
```

Listing 10: BPEL code example

5.1.7 Oracle BPEL Process Manager

This paragraph gives a short overview about the Oracle BPEL Process Manager and its components shown in fig. 51. A detailed explanation goes beyond the scope and is not necessary at this point. Because of this, only the directly involved components, related to this prototype, are described.

Regarding to the Oracle whitepaper [ORAC 06b; p.7], the BPEL Process manager is supposed to provide a high-performance, reliable execution of processes, that are defined within the BPEL standard to reduce costs and complexity of the integration projects. It provides a comprehensive and native BPEL implementation (BPEL 1.1 fully, BPEL 2.0 partial) to design, develop and deploy business processeses in a vendor independent way. Therefore it is also possible to port the implemented processes to other BPEL standard based products. Besides

this, there are several tools shipped for integration purpose, e.g. adaptors for back-end systems, transformation tools, etc. as shown in fig. 51.

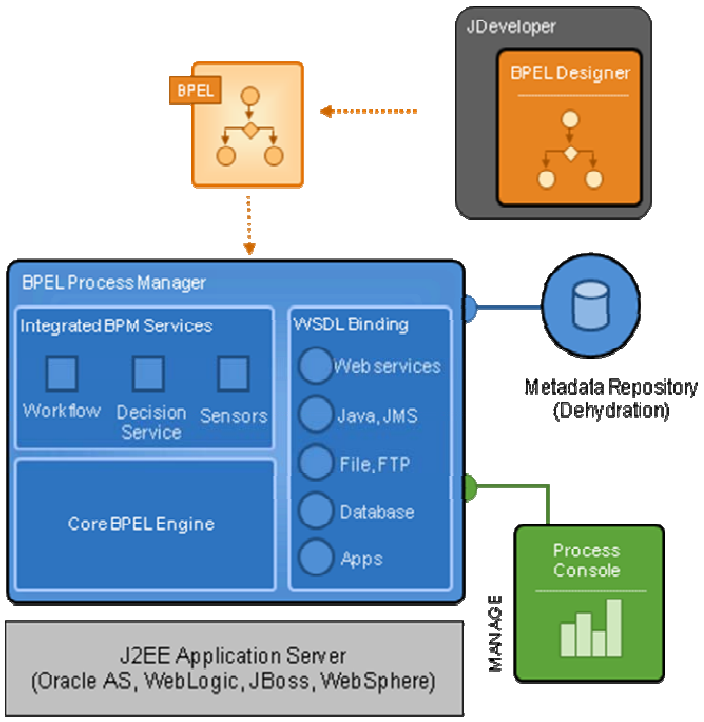


Figure 51: BPEL Process Manager [ORAC 06b]

5.1.7.1 BPEL Designer

The BPEL designer enables the user to develop BPEL processes visually in a graphical environment. Instead of writing the BPEL code by hand, user-friendly mechanism, like drag and drop for activities and other BPEL elements, as well as wizards allowing fast development of business processes are implemented. These mechanisms are also used in para. 5.2.1. But JDeveloper also offers the ability to view and modify the generated source code. The BPEL Designer is part of Oracles JDeveloper, but also available as Eclipse plug-in. The BPEL Designer allows directly deploying the processes to the BPEL Engine.

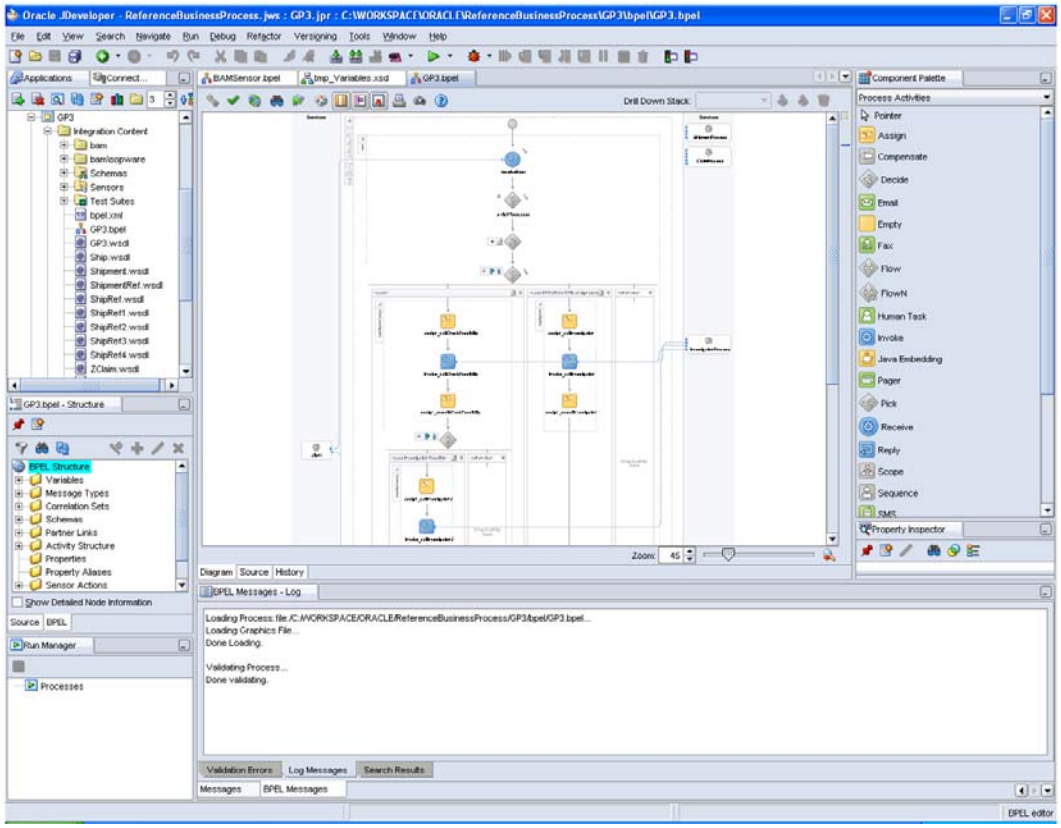


Figure 52: BPEL Designer

5.1.7.2 BPEL Engine

The BPEL Engine is the runtime environment for deploying and running standard BPEL processes. It also provides support for version control, which allows deployment and execution of different versions of a business process. To reduce the demands of the hardware it offers a mechanism called “dehydration” which stores business processes running over a longer period of time in a database. Clustering for increasing reliability is also offered. The BPEL Engine runs on a J2EE AS usually on the Oracle AS the OC4J, but is also possible to use other AS like WebLogic.

5.1.7.3 BPEL Console

The console (fig. 53) is a Web-based user interface based on JSP (Java Server Pages) and servlet, for management, administration, testing and debugging processes deployed on the BPEL Engine. After execution of a process instance, it is possible to view the visual process flow, which is helpful for testing and debugging. Also included in the application are process histories and audit trails

for better management. In case of the prototype, the process console is used for executing, debugging and managing the test cases, defined in para. 3.2.

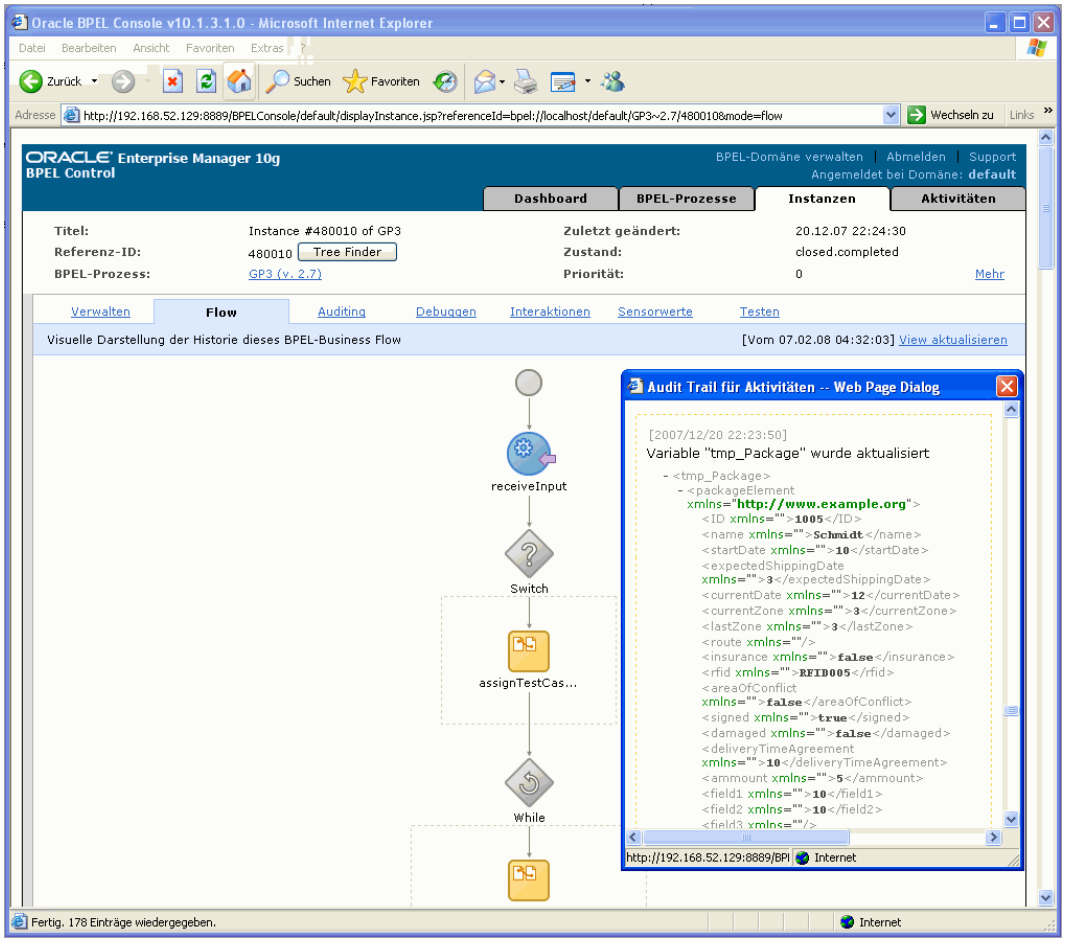


Figure 53: BPEL console

5.1.8 Advantages and Disadvantages

The approach of BPEL is to create, modify and deploy business processes in a fraction of the time compared to hard coded implementations. That reduces the information system gap time, as described in para. 5.1.1, and allows reacting faster and with less effort on the demands of the business which saves time and money. Most software vendors provide a graphical user interface allowing staffers in the operating department to change business processes by themselves. Based on that, BPEL code is generated. Therefore a programmer is not in every case mandatory but for extensive changes a programmer is recommended.

An advantage of BPEL compared to other orchestration language is, that it is based on the common standards XML Schema, XPath, WSDL and WS-Addressing as described and supported by all major software vendors like BEA,

IBM, Oracle, Sun, SAP or Microsoft. Therefore BPEL promises a universal interoperability that makes it possible to modify and execute business processes in every standard based environment. Because of this, BPEL is seen as the de facto standard for process modeling. That makes BPEL future-proof and the investment in the technology safe.

Several Web services are orchestrated in BPEL to a business process and each business process itself is always a Web service. Because of this, it is possible to orchestrate processes again in even greater business processes, which allows the creation of larger ones, in which several department specific processes are involved.

BPEL engines of most vendors, like Oracle, contain tools for auditing and reporting. That provides the basic methods to monitor business processes. But these tools just provide basic information about executed processes, e.g. a process faulted, but not the information why an error occurred. Therefore other implementations like CEP and BAM are necessary.

A disadvantage of the standard implementation of BPEL is that no human interaction is supported. BPEL was made for orchestration of Web services and therefore does not include concepts for the interaction with people. A human is e.g. the approval of payment. In nearly every business process, people are involved. Because of this, the vendors Adobe, BEA, IBM, ORACLE and SAP published the specifications BPEL4People [SAP 07b] and WS-HumanTask [SAP 07a] which extend the core BPEL4WS [OASI 07a] specification.

5.2 Integration of SOPERA services into BPEL

This paragraph describes how to implement SOPERA services into Oracle BPEL using JDeveloper 10g, based on the SOPERA shipment service and the “allocatePackage” operation. After that, the possibility to create events directly out of the BPEL process is described. These events are used for correlation in CEP and reporting purpose in BAM.

5.2.1 Technical Implementation of a SOPERA Service in Oracle BPEL

The example of the shipment service explains how to implement a SOPERA service into Oracle BPEL via the WSDL interface. It does not describe each step

how to create a complete business process with the Oracle BPEL designer. The focus is, to integrate a SOPERA service by using the BPEL components *Partner Link*, *Invoke activity* and *Assign activity*, which enable the service call.

5.2.1.1 Creating of a Partner Link

As described in para. 4.1.2, the integration of a SOPERA service is done by using a standard WSDL instead of using the proprietary SOPERA PAPI. That concept allows a plain orchestration of services.

A *Partner Link* describes a service which is used in the business process and includes the service description or simply the „user manual“ on how to use the service. It also describes possible operations and communication channels. [RUEB 07; chap. 7] It “(...) defines the location and the role of the Web services through which the BPEL process interacts, as well as the variables used to carry information between the Web service and the BPEL process. A *Partner Link* is required for each Web service the BPEL process calls (...)” [ORAC 06a; p. 218]

So the first step is to create a partner link, which refers to the WSDL file generated by SOPERA and extended with the SOAPAction. Doing that, the *Component Palette* within Oracles JDeveloper – release 10g – offers the component Partner Link within the *Services* section. The *Partner Link* is added to the existing BPEL process by dragging it from the *Component Palette* to the BPEL process, shown in fig. 54.

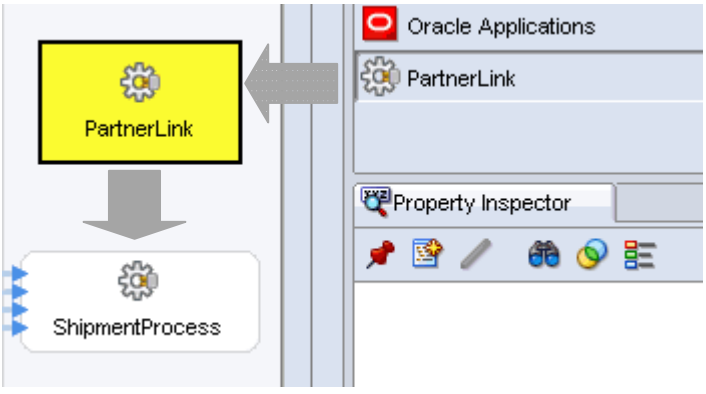


Figure 54: Add a Partner Link to the BPEL process

After adding the partner link a configuration window appears, where the reference to the service file has to be added by choosing the WSDL file from the file system. The attributes “`Partner Link Type`”, “`Partner Role`” and “`My`

`Role`" are set to the default values, because the generated WSDL file does not offer other values. These changes, shown in fig. 55, have to be applied and the creation of a *Partner Link* is finished.

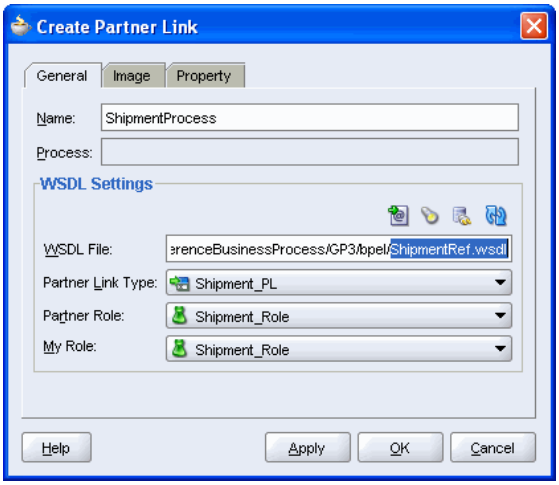


Figure 55: Partner link configuration

5.2.1.2 Creating an Invoke Activity

Based on a *Partner Link* an *Invoke activity* is needed which enables calling operations offered by the SOPERA service. The invoke activity "(…) opens a port in the BPEL process to send and receive data. A BPEL process uses this port to send the required data and return a response. (…)" [ORAC 06a; p. 218].

The *Invoke activity* is within the *Process Activities*, which are also part of the *Component Palette* and have to be dragged to the place needed in the BPEL process, as shown in fig. 56.

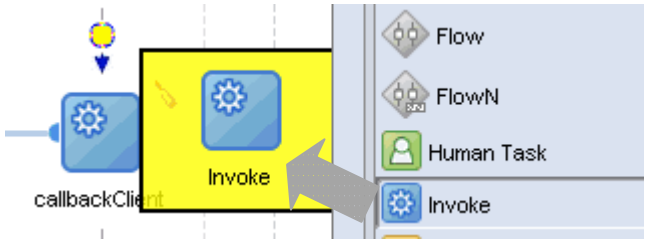


Figure 56: Insert Invoke activity

Because an *Invoke activity* sends information - typically a request - to a Web service, which is identified by its *Partner Link*, and specifies an operation for the Web service to be executed, the settings for that have to be finished. The first step is to connect the *Invoke activity* and the *Partner Link*. That is also done by the

drag and drop mechanism, as shown in fig. 57. After connecting, the configuration dialog (fig. 58) appears where the needed operation has to be selected. In this case, the "`callAllocationPackage`" operation is selected. The operation needs input and output variables for the request and response communication. These variables can be created automatically, based on the inline schemas of the WSDL, by using the wizard button and defining a variable name.

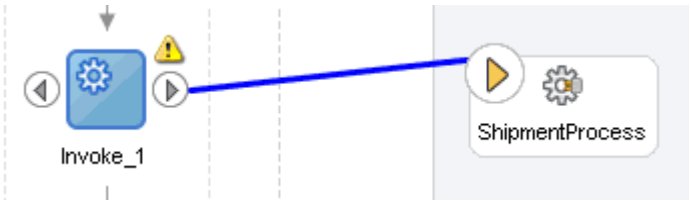


Figure 57: Connection of Partner Link and Invoke Activity

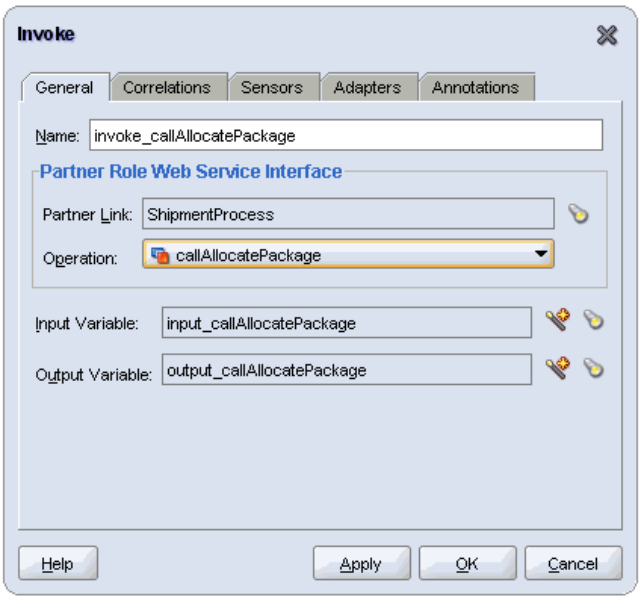


Figure 58: Invoke settings

5.2.1.3 Assigning Request and Response Variables

For calling an operation of a SOPERA service, usually input and output parameters are necessary. These values have to be assigned by using the BPEL *Assign activity*, which "(…) transfers data between variables, expressions and other elements (…)" [ORAC 06a; p. 222].

The values for assignment to the variables of the *Invoke activity* have to be defined in advance. The most common way to do that is defining temporary variables, out of the WSDL file, by using its inline schema. This is not possible in

case of a SOPERA service. Because of that, the variables have to be created out of an external XSD-schema, which has to be created first.

5.2.1.3.1 Creating an External XSD-Schema

To create the external XSD-schema, JDeveloper is also used. A right click on the folder "`Integration Content`" allows that by clicking "New", as shown in fig. 59. The wizard guides to the next steps and the result is a XSD-file within the "`Schemas`" folder of the "`Integration Content`" directory. It is very important to select "`Integration Content`". Otherwise the XSD-schema is created outside the BPEL project. The variables can be used for development, but results errors during deployment. The reason for that is if an external file is not created within the "`Integration Content`" it will not be part of the package created during the compile and deployment process. Result of that are crashed BPEL projects caused by a bug within the JDeveloper.

Figure 59: Create external XSD-schema

The created schema has to be extended with the necessary variables and elements, either by the source view or using the design view, which allows the modification of the XSD-schema with a user interface. The easiest way of defining the temporary variables is, to copy the element definitions of the WSDL into the created XSD.

But before the schema can be used it has to be extended with additional attributes. Otherwise the defined schema does not work proper with SOPERA services, because the automatically generated services are not able to unmarshal the qualified identifier of the message payload. For that the schema header of the file has to be changed in the source view, as described in following listings.

Before:

```
1 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2             xmlns="http://www.example.org"
3             targetNamespace="http://www.example.org"
4             elementFormDefault="qualified">
```

Listing 11: XSD-schema settings before

After:

```
1 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2             xmlns="http://www.example.org"
3             targetNamespace="http://www.example.org"
4             elementFormDefault="unqualified"
5             attributeFormDefault="unqualified">
```

Listing 12: XSD-schema settings after

5.2.1.4 Creating Temporary Variables

Based on the elements in the XSD schema, variables have to be created. These variables are used for mapping values within the business process to the request and response payload of the SOPERA service, by using the input and output variables of the *Invoke activity*. Within the "`BPEL Structure`" panel, XSDs shown in fig. 60, variables can be created. After choosing the element of the -schema, which should be the pattern for the variable, the wizard guides through the next steps.

Figure 60: Create temporary variable

5.2.1.5 Assigning Temporary Variable to Service Request and Response

The temporarily created variable has to be assigned within the business process with values. In the case of the prototype these values are defined in the test cases described in para. 3.2.8.2 and contain information about, e.g. the address, or the package. In real business applications these values come from the database, user entries or other applications. To transport these values, the temporary variable has to be assigned to the request and response payload of the SOPERA service. That has to be done by using the BPEL *Assign activity*, which is also part of the *Process Activities* within the *Component Palette*.

The *Assign activity* has to be dragged from the *Component Palette* to the right place in the BPEL process, for the request of the service before the *Invoke activity* and for the response after the *Invoke activity*. A double click on the *Assign activity* creates a *Copy Operation*, which allows assigning the values from the temporary variable of the business process to the parameters (payload) of the SOPERA

service. Fig. 61 shows that in the example of the element "`BPELContent`". On the left there is the temporary variable and on the right the input Variable of the created *Invoke activity*. The variables just have to be selected as shown in the picture and committed. The copy mechanism of the *BPEL Assign activity* uses an internal *XPath* statement (fig. 61) of the variables for mapping the values.

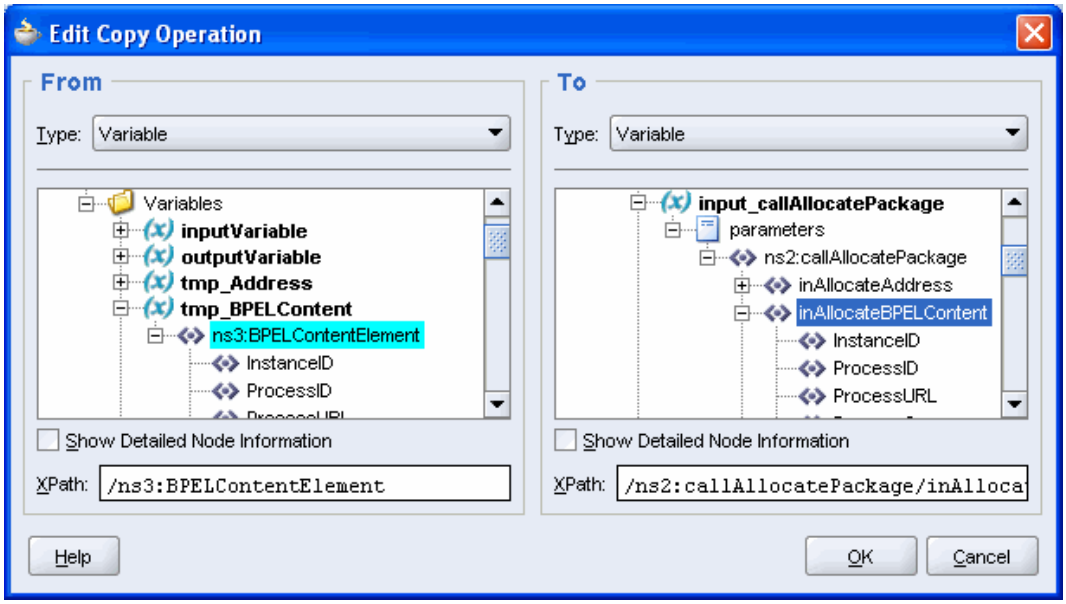


Figure 61: Copy operation

For each parameter of the service payload - for the request, as well as for the response - a *Copy Operation* with the temporary variables has to be created. After a service is called successfully, the output values of the service are automatically in the temporary variable available and are used for further processing within the business process. They are also used for providing data to BAM by using BAM sensors, described in para. 5.2.2.

5.2.1.6 Result

Each SOPERA service has to be implemented in Oracle BPEL in the same way, as described before and the result looks similar for every service, as shown in fig. 62. Before an *Invoke activity*, an *Assign activity* is necessary to map the values of the BPEL process to the request payload of the service. The *Invoke activity* calls an operation of the service, by using a *Partner Link*. The *Partner Link* includes the exact description of the SOPERA service, which is based on a WSDL file. After processing the service call, the data of the response payload are mapped

back to the variables of the business process, by using the *Assign activity* after the *Invoke activity*.

The aim of that paragraph is not to explain every detail of the integration, but give a short overview about the basic steps and the anomalies. As mentioned, this is not the only way how to integrate a SOPERA service in BPEL, but a flexible and plain solution.

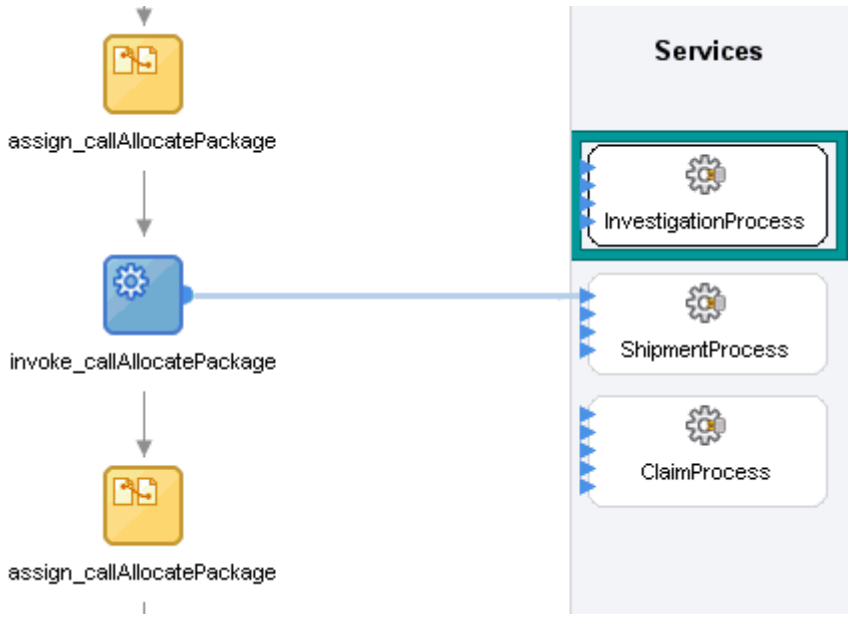


Figure 62: SOPERA service call in Oracle BPEL

5.2.2 Technical Implementation of BAM Sensor Events

Oracle BPEL offers in its 10g version the possibility to extend the process with sensors for monitoring, as described in para. 3.2.2.3. Using BPEL sensors, it is possible to extract information directly out of the BPEL process components, e.g. an *Assign activity* or an *Invoke activity*. The extracted information is in our case an event and published either directly to a *BAM Data Object* or to CEP using a *JMS topic*. Because CEP is not part of *Oracle Application Server 10g*, there is no way to publish the events directly to CEP with an embedded component of JDeveloper. In the *11g version*, it is planned to offer a sensor, which is able to publish events directly to CEP. Because of this circumstance, sensors publishing events via a *JMS topic* to CEP is just a temporary solution. The following paragraph describes how to extend the existing BPEL process - including SOPERA services - with sensors for CEP and BAM.

5.2.2.1 Creating an *Activity sensor*

For creating sensor events, the first step is to create an *Activity sensor*. It is used to monitor the execution of activities within a BPEL process, e.g. execution time of an *Invoke activity*, or how long it takes to complete an *activity* or *scope*. It is also possible to monitor variables, but not directly the data of the payload [ORAC 06a; p. 262]. Because of this, it is necessary to create temporary variables (para. 5.2.1.4). To create an *Activity sensor*, the component which has to be monitored, needs to be selected. In the properties tab of the component – e.g. an *Invoke activity* – there is a tab called "`Sensors`", allowing the creation of a new *Activity sensor*. Fig. 63 shows the creation window for the *Activity sensor*. The "`Activity Name`" shows the activity on which the sensor is based on, in this case on the "`callInvokeShipment`". In the *Configuration* block, the "`Evaluation Time`" has been chosen. This property means, on which state of the activity the event has to be fired and can have the values "`Activation, Completion, Fault, Compensation, Retry, All`". In this case, the event is fired after the *Invoke activity* completes, which means after the SOPERA service call is finished. The section *Activity Variable Sensors* allows selecting a "`variable`" of the BPEL process, which has to be monitored. That must not be the payload data, but has to be data of a temporary variable, as defined in para. 5.2.1.5. The values of the selected variable are available for transmission to CEP or BAM after adding to the *Activity Sensor*.

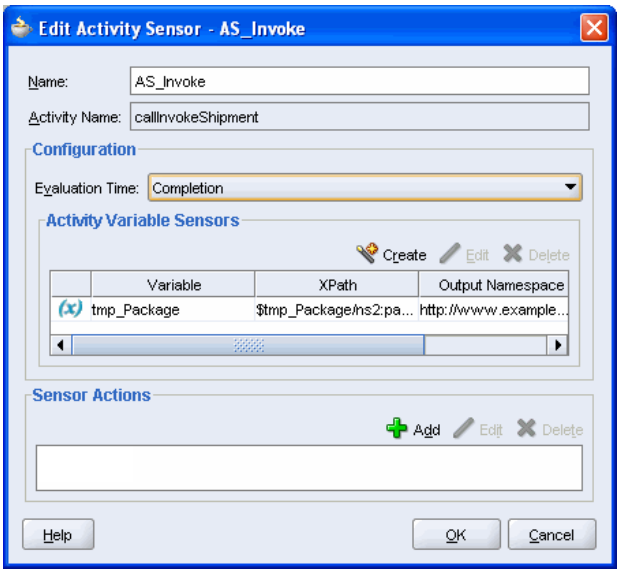


Figure 63: Activity Sensor

5.2.2.2 Publishing to BAM

These steps describe the approaches how to publish data to BAM. That can be done either by using JMS or the BAM Sensor Action, what is described in this paragraph.

5.2.2.2.1 Creation of a Data Object in BAM

The sensor data is published to a kind of database table called *Data Object*, which holds the data for BAM reports. The *Data Object* includes the fields, which have to be filled with the sensor. A detailed description about *Data Objects* and its creation is described in para. 8.2.1.

5.2.2.2.2 Creation of a *BAM Sensor Action*

The *Sensor Action* is responsible for publishing the data of a sensor to an endpoint, what is in this case the BAM *Data Object*. The *BAM Sensor Action* has to be created as shown in fig. 64 in the *Structure Panel* of the BPEL process.

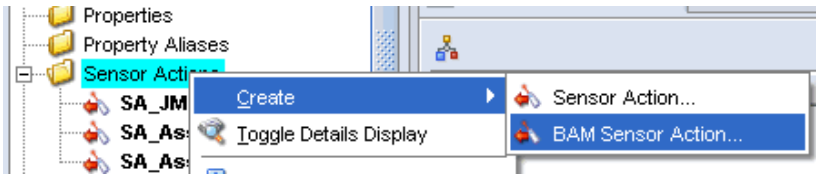


Figure 64: Create BAM Sensor Action

After that, the settings of the *BAM Sensor Action* have to be completed, as shown in fig. 65. The attribute "`Sensor`" allows selecting *Sensor activity* which was in advance created. Only valid ones can be selected, *Sensor activities* including an inline WSDL schema are not supported. That is the reason for using temporary variables. The attribute "`Data Object`" has to be filled with the one, to which the data should be published. The "`Operation`" specifies the operation which has to be finished on the *Data Object* side. That can be "`Insert, Update, Delete, Upsert`". "`Upsert`" is the most common operation and combines "`Insert`" and "`Update`". If a record with the specified key is already in the database, an update is done, otherwise an insert.

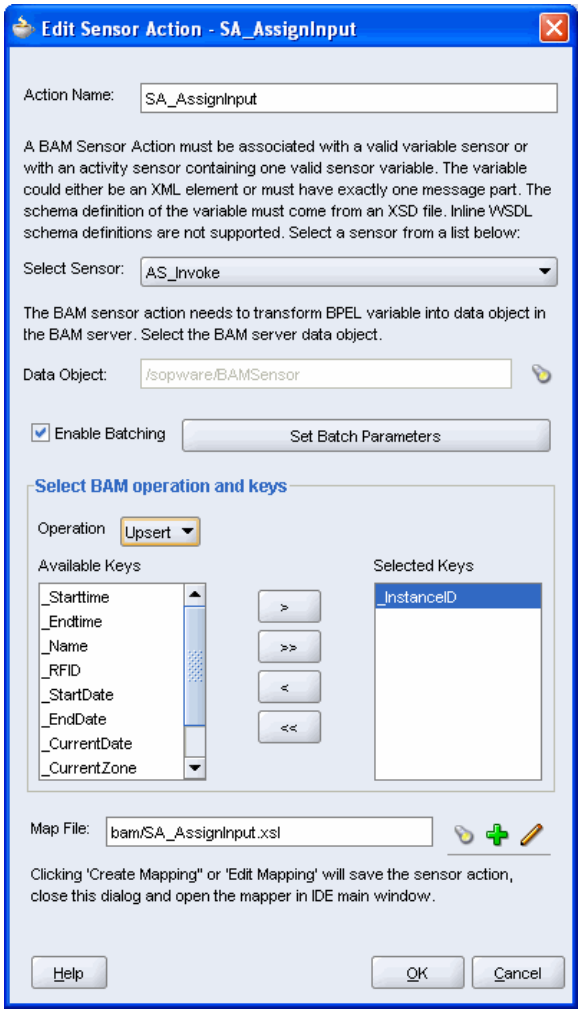


Figure 65: Sensor Action

The "`Map File`" defines the association of the fields specified in the *Activity sensor* to the corresponding fields of the *Data Object*. The fields are assigned by using the drag and drop mechanism. It is also possible to convert the values before mapping to the destination field, as shown in fig. 66. There are several String-functions available which can be plugged between the associations.

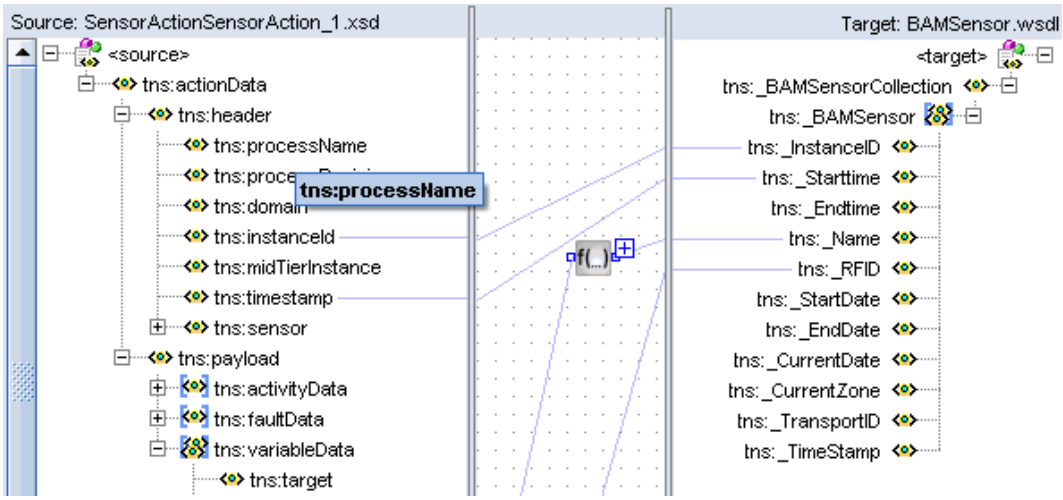


Figure 66: Map File

5.2.2.3 Publishing to CEP

As mentioned, it is also possible to publish events directly out of the BPEL process to CEP using a *JMS topic*. This *non-intrusive* mechanism allows generating events without changing the business logic of a service with some restrictions, as described in para. 4.2.9.2.

Instead of using the *BAM Sensor Action* which publishes the data directly to a *Data Object*, a regular *Sensor Action* has to be created, which sends its data to a *JMS topic* as shown in fig. 64. For using JMS as an endpoint, the specific settings have to be uesd as shown in fig. 67. The attribute "`Publish Type`" has been set to "`JMS topic`". The attributes "`JMS Connection Factory`" and "`Publish Target`" have to be filled with the settings of the endpoints topic. Further information about JMS topics can be seen in para. 7.2.2.

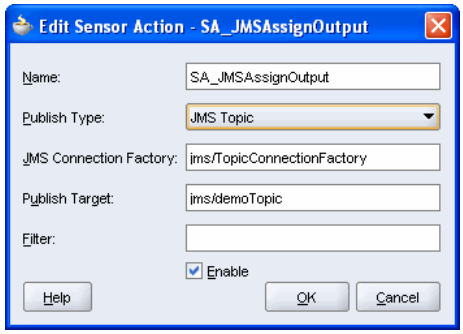


Figure 67: JMS settings

5.2.2.4 Result

After deployment the changes affect the business process. During when a process instance is running, the sensor publishes the events, depending on the *Sensor Action* either directly to the BAM *Data Object* where the information is available for monitoring purpose in a BAM dashboard or via *JMS topic* to CEP for further processing.

5.2.2.4.1 Data Object

The values in a *Data Object* are similarly stored to a database table as shown in fig. 68. The field mapping, as described before, allows assigning of corresponding fields. BAM reports access the *Data Object* to select the content for the reports.

Row ID	InstanceID	Starttime	Endtime	Name	RFID
2	490004	29.01.2008 21:23:31	29.01.2008 21:23:34	Weber	--null--
4	490005	29.01.2008 21:30:49	29.01.2008 21:30:53	Weber	RFID004
6	490006	29.01.2008 23:01:03	29.01.2008 23:01:14	Weber	RFID004

Figure 68: Data Object example

5.2.2.4.2 JMS topic

The values sent to a JMS topic are not stored persistently, but sent using XML to the endpoint. CEP, for example, accesses the *JMS topic* and extracts the values for further processing. The structure of the XML is very complicated, because all values defined in the *Activity sensor* as well as *header information* about the BPEL process are included. Because of this circumstance, the consumer of the *JMS topic* has to transform the received XML before it can be used for further processing. A simplified XML document sent from a sensor to a *JMS topic* is shown in listing 13. It shows the basic data to provide a better understanding. The whole XML document includes many more values.

```
1 <actionData xmlns="http://xmlns.oracle.com/bpel/sensor">
2   <header>
3     <sensor sensorName="AS_Invoke" ...
   target="callInvokeAllocatePackage"...>
4       <activityConfig evalTime="completion">
5         <variable outputDataType="packageElement" .../>
6       </activityConfig>
7     </sensor>
8     <instanceId>500005</instanceId>
9     <timestamp>2008-01-31T02:10:15.390+01:00</timestamp>
10  </header>
11  <payload>
12    <activityData>
13      <activityType>invoke</activityType>
14      <evalPoint>completion</evalPoint>
15    </activityData>
16    <variableData>
17      <data>
18        <packageElement xmlns="http://www.example.org">
19          <name xmlns="">Weber</name>
20          <rfid xmlns="">RFID004</rfid>
21        </packageElement>
22      </data>
23      <updaterName>callInvokeAllocatePackage</updaterName>
24      <updaterType>invoke</updaterType>
25    </variableData>
26  </payload>
27 </actionData>
```

Listing 13: JMS event example

6 Notification Receiver

According to the SOPERA documentation, "(…) The Notification Receiver is the central aggregation point for management notifications." [SOPE 07, p. 5, 6] In the example implementation, not only management notifications such as SBB events are published, but also business events. The basic functionality of the NR is the transportation of events from the SBB to the consumer. The NR catches all events on the service bus and processes them. The processing of the events can be an entry in a log file as well as putting the events on a message bus e.g. by using JMS technology.

Any other processing way is also useable because of the dynamic structure of the NR operations. Each operation describes one way of processing the events.

6.1 Architectural Overview

To understand all portions the NR contains, it is best to explain the structure shown in fig. 69.

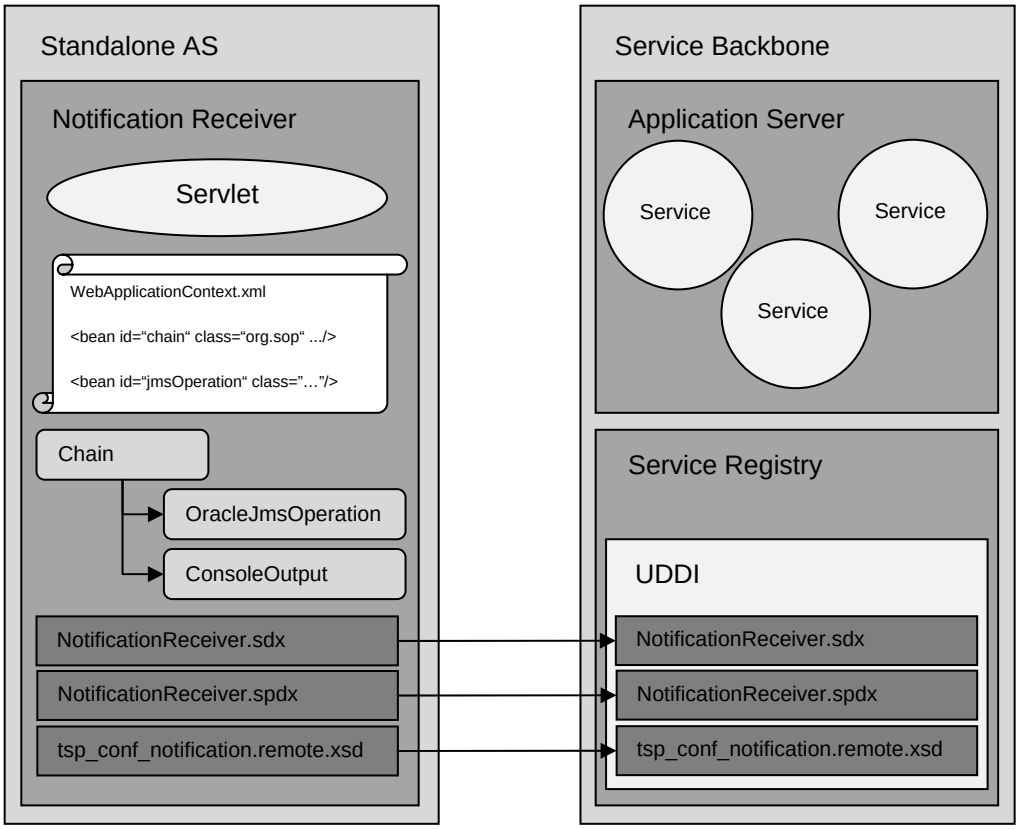


Figure 69: Architectural overview of the NR

The NR is basically a SOPERA service running on a different servlet engine (in the example implementation within a Tomcat server). The fact that the NR is a SOPERA service means that it needs to be deployed on the SBB through the deployment descriptors (fig.69 – `NotificationReceiver.sdx` and `NotificationReceiver.spdx`). The deployment takes place in the service registry together with other configurations. The other configurations will be explained in paragraph 6.2.3, configuring the Service Registry for “Additional Operations”, in a more detailed way.

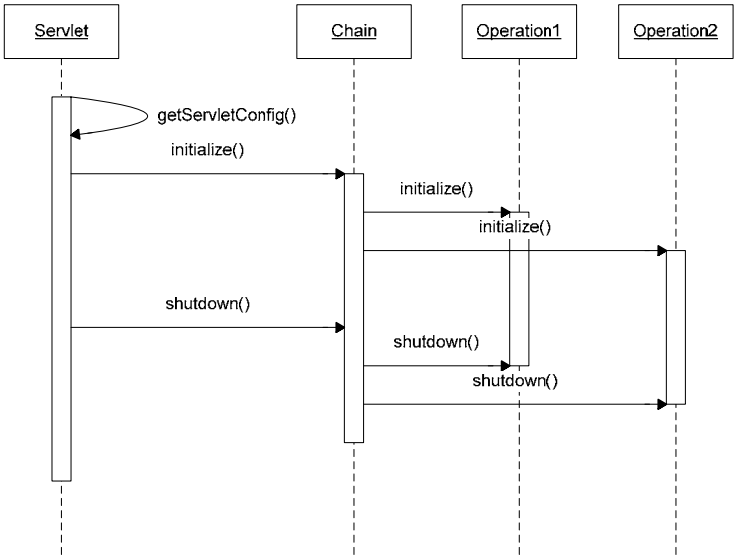


Figure 70: NR operation startup model

To run the service, a servlet starts the service provider during startup. The service provider starts a chain containing all configured operations. The operations are defined in the `tsp_conf_notification.remote.xsd` (para. 6.2.3) as well as in the `WebApplicationContext.xml` (para. 6.2.1) and process the incoming events.

6.2 Extending the NR with Additional Operations

To extend the functionality of the NR, with additional operations, three basic actions need to be taken. First of all, writing and configuring a new operation within the component, secondly running a maven build on the java files and thirdly modifying the service registry as well as the SOPERA Configuration Plugin within the SOPERA Administration view.

6.2.1 Writing and Configuring Additional Operations

To understand the technical structure of operations, the class diagram in fig. 71 gives an overview.

All operation implementations are in the package `org.sopware.management.tools.op.impl`. They extend the class `AbstractOperation` of the package `org.sopware.management.tools.op`. `AbstractOperation` offers methods to fetch the configuration and defines a method to process notifications that has to be implemented in each of the specific operations.

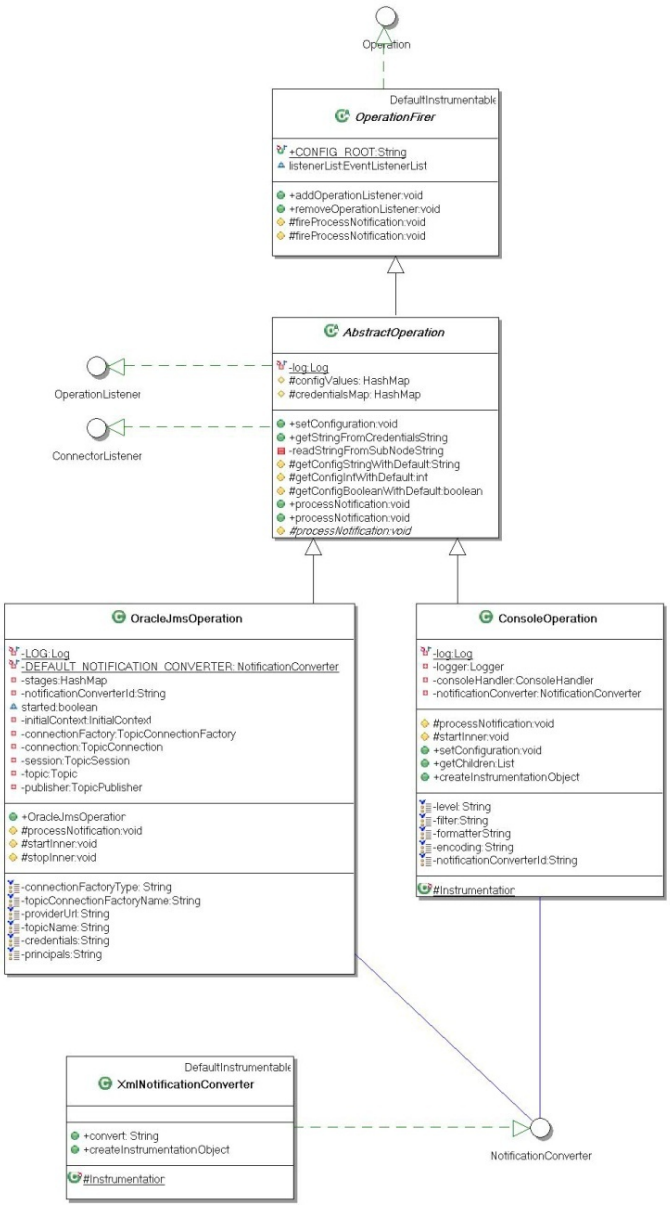


Figure 71: NR operations class diagram

`AbstractOperation` extends the `OperationFirer` and implements the interfaces of the `OperationListener` and the `ConnectorListener`. The `OperationListener` defines a method to process notifications coming as `OperationEvents`, the `ConnectorListener` describes how to process notifications coming as `ConnectorEvents`. The `OperationFirer` extends the `DefaultInstrumentable` class of `org.sopware.management.tools.common` and expands the functionality of `DefaultInstrumentable` to the implementation of the interface `Operation`. This is e.g. the functionality to get events from the `EventListener`.

On the bottom, there are several `NotificationConverter` implementations. One of them is the `XmlNotificationConverter` that changes a message format to an XML format. This converter type is used in `OracleJmsOperation` to change the output to an XML conform format.

For adding additional operations, it is only necessary to add additional classes that extend the `AbstractOperation` and write the code to process notifications. For starting and stopping purposes, the operation needs also to include a `startInner()` and a `stopInner()` method. Within these two methods, all functionality should be implemented that offers the ability for executing the operation. Within the `processNotification()` method, the actual logic of how the notification should be processed needs to be coded. After this is done, there is only some configuration left.

To get the operation up and running on startup of the servlet engine, the operation needs to be put into the Spring configuration as a bean. The following listing gives a simple example of the Spring configuration in the `WebApplicationContext.xml` assuming that the `OracleJmsOperation` is the one lately created.

```
1 <bean id="jmsOutput"
2    class="org.sopware.management.tools.nr.op.impl.
3       OracleJmsOperation" singleton="false">
4    <property name="topicName" value="jms/demoTopic" />
5    <property name="topicConnectionFactoryName"
6       value="jms/TopicConnectionFactory" />
7    <property name="connectionFactoryType"
8       value="com.evermind.server.rmi.
9          RMIInitialContextFactory" />
10   <property name="providerUrl"
11      value="opmn:ormi://localhost:6003:home" />
12   <property name="principals" value="oc4jadmin" />
13   <property name="credentials" value="oc4jadmin" />s
14 </bean>
```

Listing 14: Excerpt of the WebApplicationContext.xml

In the `WebApplicationContext.xml` there is a bean definition with the id equal to jmsOutput. This bean contains all properties with their configured values. The values of this example match the values to connect to a JMS provider. These properties need to also be defined in the class file as empty attributes, including their getters and setters. At startup, these attributes will be filled with the values from the configuration file. To ensure that the right types of the properties are loaded, it is necessary to create an instrumentation file. In the example case, this is an instrumentation called `OracleJmsOperationInstrumentation.xml`. This file defines the data types as shown in listing 15 as well as the actions.

The example listing shows to which domain the operation belongs and which class file is loaded. All of the previously defined property values will be interpreted as `java.util.String` and only the property `topicName` can be overwritten during runtime. Why this behavior is needed is explained in a later section.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE mbeans-descriptors PUBLIC "//Apache Software
3 Foundation//DTD Model MBeans Configuration File"
4 "http://jakarta.apache.org/commons/dtds/
5 mbeansdescriptors.dtd">
6 <mbeans-descriptors>
7    <mbean name="OracleJmsOperation.Instrumentation"
8       domain="sbb/participant"
9       type="org.sopware.management.tools.nr.op.impl.
10      OracleJmsOperation.Instrumentation"
11      description="Participant component">
12
13      <!-- DefaultInstrumentabletable -->
14      <attribute type="java.util.String"
15         name="beanName" writeable="false">
16      </attribute>
17
18      <!-- OracleJmsOperation -->
19      <attribute type="java.util.String"
20         name="topicName" writeable="true">
21      </attribute>
22      <attribute type="java.util.String"
23         name="topicConnectionFactoryName"
24         writeable="false">
25      </attribute>
26      <attribute type="java.util.String"
27         name="connectionFactoryType" writeable="false">
28      </attribute>
29      <attribute type="java.util.String"
30         name="providerUrl" writeable="false">
31      </attribute>
32      <attribute type="java.util.String"
33         name="principals" writeable="false">
34      </attribute>
35      <attribute type="java.util.String"
36         name="credentials" writeable="false">
37      </attribute>
38      <attribute type="java.util.String"
39         name="tofile" writeable="false">
40      </attribute>
41      <attribute type="java.util.String"
42         name="file" writeable="false">
43      </attribute>
44
45      <!-- DefaultInstrumentabletable -->
46      <operation name="start" impact="ACTION"/>
47      <operation name="stop" impact="ACTION"/>
48   </mbean>
49 </mbeans-descriptors>
```

Listing 15: Instrumentation on the example of the OracleJmsOperation

To create a relation between the Spring beans and the operations that are configured to startup in the SBB (description how to configure the chain on the SBB can be found in para. 6.2.3,), it is necessary to make an entry in the `NrConstants` class (this class can be found in the package `org.sopware.management.tools.nr`). This class contains a `HashMap` called

`SPRING_CONFIG_BEAN_MAPPING` where the keys of the mapping are the names of the operation in the chain and the objects are the names of the operation as they are configured in the Spring `WebApplicationContext.xml`. An example for this is shown the following listing.

```
1 SPRING_CONFIG_BEAN_MAPPING.put(
2    "sopnot:jmsOutput", "jmsOutput");
```

Listing 16: Mapping of Spring beans to SBB configuration

After all these actions have taken place, the NR is ready to get a new build.

6.2.2 Building the NR with Apache Maven

To build the NR project, SOPERA is using the Apache Maven project in the version 1.0.2. According to the projects web site, Maven "(…) is a software project management and comprehension tool. Based on the concept of a project object mode (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. (…)" [MAVE 08]. The objectives Maven wants to provide are:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practice development
- Allowing transparent migration to new features

To understand the structure of Maven, it is necessary to talk about the plug-in concept. Maven is using multiple plug-ins that are already bundled with the Maven distribution. The most common plug-in that is used for java projects is the "java" plug-in. This plug-in is responsible for executing the compilation goal of a java project. Other plug-ins that are not shipped with the installation as well as custom plug-ins need to be declared in the dependency section of the project object model.

```
1 <dependency>
2    <groupId>spring-framework</groupId>
3    <artifactId>spring</artifactId>
4    <version>1.2.5</version>
5    <type>jar</type>
6    <properties>
7       <war.bundle>true</war.bundle>
8    </properties>
9 </dependency>
```

Listing 17: POM dependency description within project.xml

From that point, Maven tries to download the projects from the web into the local repository. Alternatively, there is also an option to manually install plug-ins into the central Maven repository.

After all project plug-ins are placed in the repository, it is necessary to create a `project.xml` that contains the POM. It is also possible to define parent projects where the basic configurations can be completed. In case of the NR, this parent project is called `MavenCommonProject`. Within that project all general settings are done. All extending projects inherit these settings. An overview about the most important settings files is shown in table 27.

project.xml	Contains the project object model “POM”
project.properties	Contains project properties. This is the file that enables to customize maven specific to each single project
maven.xml	Contains customized processing information.

Table 27: Overview of the Maven settings structure

Further information about the build configuration through Maven can be found on the projects web site.

When all files are configured in a proper way and all project dependencies are resolved it is possible to run the build by executing the following command within the root of the NR project:

```
1 C:\WORKSPACE_NR\mss-notificationreceiver>
2    %maven_home%\bin\maven war
```

Listing 18: Maven execution within command line

After this command has been executed, Maven starts to create a war-file that has to be deployed in the servlet engine.

6.2.3 Configuring the Service Registry for Additional Operations

As already mentioned at the end of para. 6.1, the NR is also a SOPERA service. The service must be deployed within the central service registry as well as all other SOPERA services. Besides the standard deployment of the service, it is also necessary to configure the chain that contains all operations remotely. To achieve this goal, the `tsp_conf_notification.remote.xsd` file needs to be modified. Three things need to be done before the file is ready to install:

1. Adding a complex type to that XML schema

```
1 <xs:complexType name="JmsOperationType">
2    <xs:attributeGroup ref="sopcs:elementMetaAttribGroup" />
3 </xs:complexType>
```

Listing 19: Schema definition example for operation type

   This type can also contain more sub elements for configuration like information about the notification converter.

2. Adding an operation element

```
1 <xs:element name="jmsOutput" type="JmsOperationType">
2    <xs:annotation>
3       <xs:documentation>JMS Operation</xs:documentation>
4    </xs:annotation>
5 </xs:element>
```

Listing 20: Schema definition example for operation element

   This operation element points to the complex type created earlier.

3. Reconfiguring the chain including the new operations

```
1 <xs:complexType name="ChainDescType">
2    <xs:complexContent>
3       <xs:extension base="sopcs:ComponentType">
4          <xs:choice minOccurs="1" maxOccurs="unbounded">
5             <xs:element ref="consoleOutput" />
6             <xs:element ref="jmsOutput" />
7             <xs:element ref="cepOutput" />
8          </xs:choice>
9       </xs:extension>
10   </xs:complexContent>
11</xs:complexType>
```

Listing 21: Schema definition for operations chain

   Contains the before created operation.

To install the configuration on the service provider side, there is a small script, shipped with the SOPERA, called `setup_nr.cmd`. After the configuration was

copied to `%SOPWARE_HOME%\Devbox\devbox-modules\tsp-notification-receiver\conf\`, the script needs to be executed to extend the LDAP service registry for the NR. After the script has been executed, there are two other places where schema definition needs to be copied to. The first one is `%SOPWARE_HOME%\ServiceBackbone\conf\schemas\configsys\tsp\notificationReceiver\remote` because this is the place the service back bone validates against and the second place is `%SOPWARE_HOME%\AdminTool\install\tool\plugins\org.sopware.toolsuite.admintool.plugin.AdminToolPlugin_1.0.0\conf\cfg\cache-schemas\tsp\notificationReceiver\remote`, the place where the Eclipse plug-in validates against.

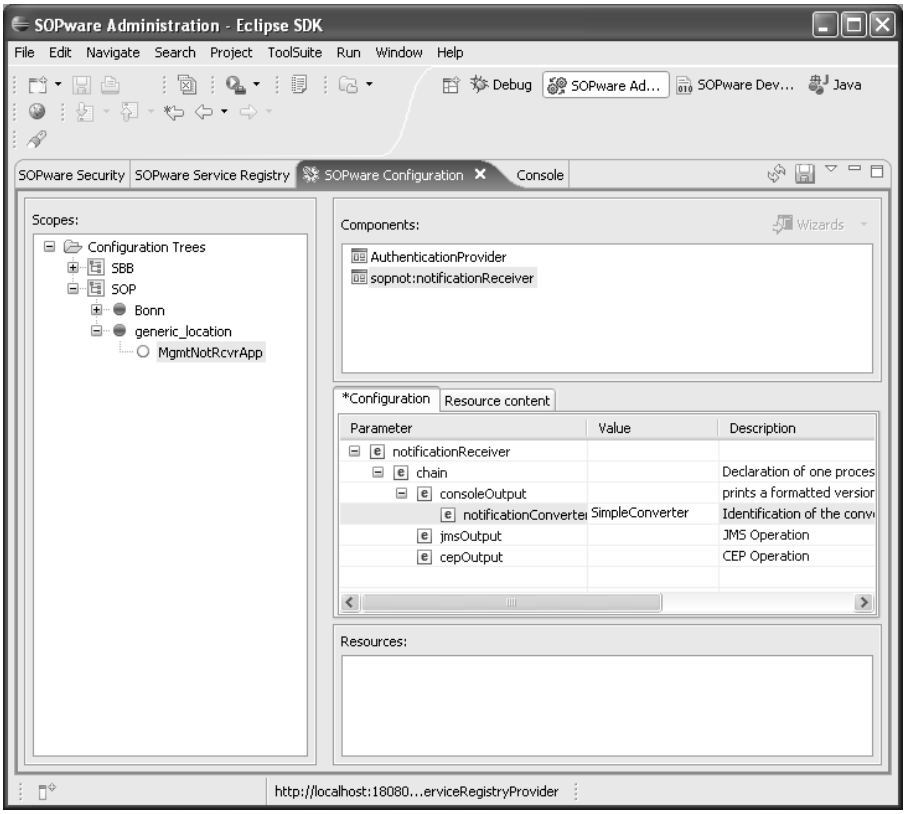


Figure 72: SOPERA configuration plugin

After these settings are done, the chain has to be configured thorough the SOPERA Administration tool and after a restart of the servlet engine, the NR contains all configured operations at start up.

6.3 Processing Events through the NR

As already mentioned in para. 3.3, there are several ways of processing events throughout the whole system. And not even only different ways of how to stream these events but also where to transform them. The NR is one point within the system that can take effect in transforming events.

For the example implementation, the decision came up to use a single JMS topic for each event type. This means that the NR acts as a dispatcher, getting all types of events in a single stream from the event cloud in the SBB and pushing them out onto different type-sorted streams.

Another issue the architecture has to deal with is that Oracle CEP and BAM have different ways of interpreting the event types and different ways of accessing the event data. This means that basically two ways of processing the event data were necessary to implement. First of all an, implementation that BAM is able to deal with the events and secondly an extension of the achieved implementation to be able to interpret the event data with CEP. The two implementations are described in the next two sections.

6.3.1 Push Through Mechanism for Oracle BAM

The first idea of processing the events is derived from para. 3.3. The basic idea behind it is to achieve a flexible system that allows adding additional event types without having major effects on the existing architecture and without having an overhead of configuration in the event type transformation.

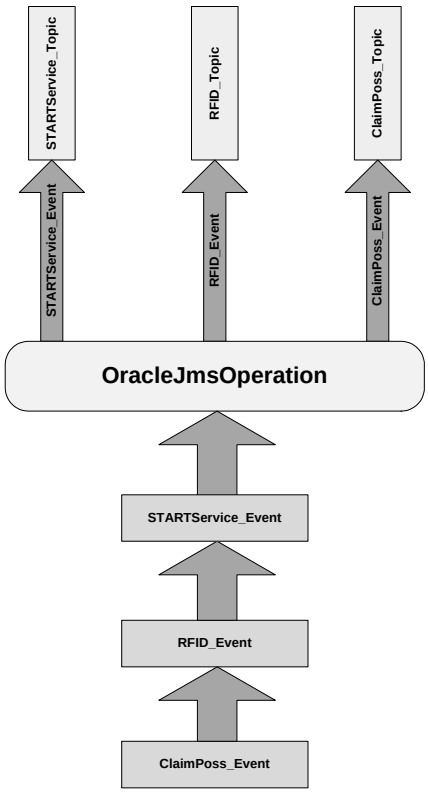


Figure 73: Push through mechanism overview

To achieve that purpose, the design is a combination of the previously described patterns, Multiple Streams Multiple Event Processing and Single Stream Multiple Event Processing. This means basically that a single event stream containing all event types arrives at the NR where it is split into single streams according to their event types. The NR pushes the events trough the system on to the JMS publisher. Fig. 73 visualizes this processing mechanism.

To improve the processing performance a caching mechanism is included in the operation. The mechanism stores the used topics to avoid continuous network lookups in the JMS registry.

```
1 String topicName = “”;
2 HashMap topicHash = new HashMap();
3
4 if(event.getEventType() != null) {
5    topicName = "jms/" + event.getEventType();
6 } else {
7    // default value for events not having an event type
8    topicName = "jms/defaultTopic";
9 }
10
11 if(!topicHash.containsKey(topicName)) {
12   // in case the topic has never been used make a lookup
13   // and store the topic information
14   topic = (Topic) context.lookup(topicName);
15   topicHash.put(topicName, topic);
16 } else {
17   // otherwise, just fetch the existing topic
18   topic = (Topic) topicHash.get(topicName);
19 }
```

Listing 22: Topic caching mechanism within NR operation avoiding lookups

The code snippet describes briefly the caching mechanism as wells as the split mechanism. First of all it is checked if there is an event type set in the description of the event. If there is no event type, the set event belongs to the unknown events and cannot be used for aggregation. That is the reason why it is fired against the `defaultTopic`. If there is an event type set, this type needs to be the same name as the JMS topic.

The caching mechanism first looks up the topic in its container and if it does not find an object with the event type as a name, it will look the topic up in the JMS registry and store the object in the container.

After this is done, the operation is ready to fire the event to the event type specific JMS topic and the data can be processed with the consuming applications.

6.3.2 XML Transformation Mechanism for Oracle CEP

After the design of the implementation for the streaming mechanism is defined, it is necessary to think about where the events can be transformed in a way that all connected applications can handle the event data.

Basically there are two ways that can be thought about. The first one is that the consuming applications need to take care of the transformation themselves. This means that the implementation can be done in the way the previous section describes and can be done in combination with Oracle BAM. BAM offers at this

point a flexible XSLT transformation way with which each incoming XML format can be changed to be able to process and display the data.

Oracle CEP does not have the same flexibility at this development stage. CEP has XML processing capabilities for only one layer (e.g. all sub elements need to be placed directly below the root element). To achieve this in a dynamic way, the following class diagram describes the Oracle CEP operation and all classes needed for the conversion.

Figure 74: NR XML event transformation

A CEP operation owns a `NotificationConverter`. This interface defines only that a converter needs the ability to convert incoming notifications. The `AbstractXmlConverter` implements the convert method of the `NotificationConverter` and extends the `DefaultInstrumentable`. In the convert method, this class fetches a string as input and tries to create a valid W3C XML document.

Additionally this class also defines that all sub classes need to implement an operation called `processElement()`.

The `XmlCepConverter` is a derived class from the `AbstractXmlConverter`. Its function is to process all elements that are equal over all event types. For each specific event type, this class contains a collection that has all conversion classes for each event type. Classes that transform a specific event type implement the `TransformEvent` interface so that the `XmlCepConverter` does not need to know which subclass it calls and can operate through the interface definition.

To add any new event type, it is necessary to create a class implementing the `TransformEvent` interface and add an entry in the mapping. Afterwards, the NR automatically passes the new event to the transformation class to convert the event in a CEP readable format.

7 JMS Provider

The Java Messaging Service [SUNM 02] is a standard implementation that enables enterprise applications to communicate with each other in a loosely coupled way. In contrast to existing peer-to-peer messaging systems, the JMS technology enables users to get the ability of implementing a central messaging instance within their computation system. This central instance of a MOM [SUNM 02, p.13] is becoming an essential component for integrating and combining business components in a reliable and flexible way.

7.1 General Architectural Overview

Because it "(…) is expected that JMS providers will differ significantly in their underlying message technology (…)" and that "(…) there will be major differences in how a provider's system is installed and administered (…)" [SUNM 02], the JMS specification generally describes only the following components:

JMS Provider: Software vendor that offers the ability to get connections through a remote lookup. These connections are used to pass the messages from producing to consuming clients.

JNDI Namespace: Central repository that defines the namespace through which providers can pass their proprietary way of the connection establishment (fig. 75 CF – connection factory) and to which destination the message will be passed (fig. 75 D - destination)

JMS Client: Clients are getting through a lookup in the JNDI Namespace their information about the connection factory and where the destinations of the messages are.

Generally speaking there are two different types of clients. On the one side message producing clients, that push messages on to the destination defined in the JNDI namespace, on the other side consuming clients that fetch the information from that endpoint.

Administration: The administration is bound to the JNDI Namespace and defines the connection factories (fig. 75 CF) and destinations (fig. 75 D) there.

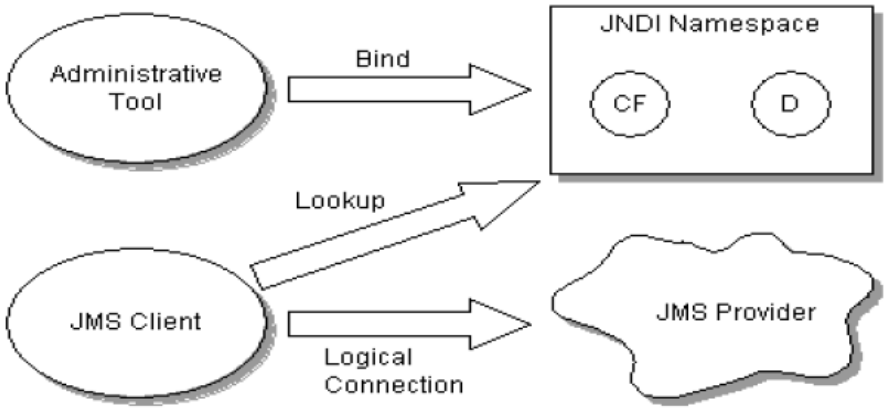


Figure 75: JMS administration [SUNM 02]

Clients are seen in this architecture as portable components. This is done by the administered objects connection factory and destination to encapsulate them from the proprietary provider systems. The client uses these objects only through portable interfaces.

Fig. 76 shows the common interfaces of JMS and how they play together to enable a JMS communication. The explanation about the flow is enriched with short code snippets that point out the way a JMS communication can be established.

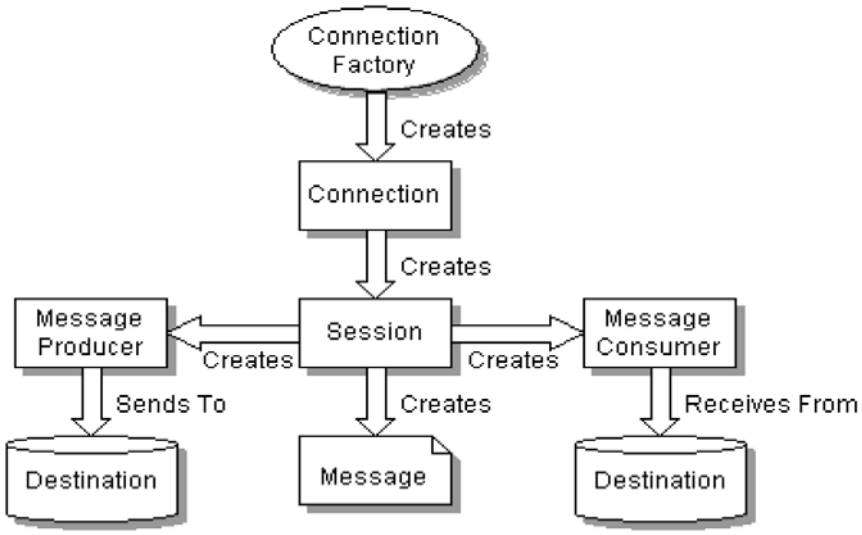


Figure 76: Overview of JMS object relationships [SUNM 02]

At this point, it is assumed that the administration of setting up a connection factory, adding destinations and running a JMS provider is already finished. It is also assumed that the JNDI settings on the client side have been completed within their properties file.

First of all, a client no matter if message producer or consumer needs to retrieve an object of a connection factory. To realize this, it is necessary to execute a JNDI look up on the name of the connection factory.

```
1 Context context = new InitialContext();
2 ConnectionFactory connectionFactory =
3    (ConnectionFactory)
4    context.lookup(“ConnectionFactory”);
```

Listing 23: Context lookup fetching the connection factory

In the same way, it is necessary to fetch a destination for the messages from the JNDI namespace. To understand a destination it is necessary to read the next section about the communication styles (para 7.2). At this point, the object description is held in a consistent structure by adding some pseudo code to the description, but, this code portion cannot be applied into a real Java JMS application.

```
1 Destination destination =
2    (Destination)context.lookup(“Destination”);
```

Listing 24: Context lookup feching the destination

As a second step an active connection to a JMS provider needs to be established. Further communication runs through this connection.

```
1 Connection connection =
2    connectionFactory.createConnection();
```

Listing 25: Connection creation through the factory

In the next step, a session needs to be established. A session is a single threaded context for sending and receiving messages.

```
1 Session session =
2    connection.createSession(false,
3    Session.AUTO_ACKNOWLEDGE);
```

Listing 26: Session creation via the connection

Now it is necessary to know if the application is a message producer or a consuming application. In case of a producing application, a message producer

needs to be created. The message producer is created by a session and used for sending messages to a destination.

```
1 MessageProducer sender =
2    session.createProducer(destination);
```

Listing 27: Message producer creation through the session with destination as parameter

Message consumers are also created by a session and are used for receiving messages that have been sent to a destination.

```
1 MessageConsumer receiver =
2    session.createConsumer(destination);
```

Listing 28: Message consumer creation through the session with destination as parameter

After the message producer and receiver are active there is still one open question on JMS. In which way is a communication between the messaging participants possible? The next section explains this question and also closes the open gap of this paragraph on how the implementation of the destination can look in a real Java application.

7.2 JMS Communication Styles

JMS applications can use basically two communication styles. The one is a point-to-point method, the other a publish/subscribe model.

Point-to-point models can be seen as from their communication style like the communication through a letter. The sender is writing a letter, encloses it and puts it into an addressed envelope. Afterwards, the message transmission unit is the post service that brings the letter directly to the recipient. The recipient gets the envelope in which the letter is included and is able to read the message as a single consumer.

The publish/subscribe model in real life is a communication style of a bill-board. The sender wants to address as many people as possible that might be interested in his topic. That is the reason why he is putting out his message on to a board where anyone is able to read the information.

7.2.1 JMS Point-to-Point Model

“Point-to-point systems are about working with queues of messages.” [SUNM 02, p. 75] A message queue can be described like a mailbox. If someone is about

retrieving his emails, he always connects to a single message queue, the inbox, where he fetches all his data from.

Figure 77: Message queuing with JMS

Message queues are in most cases created in an administrative operation. This also means that queues are seen in most systems as static components, even if some product vendors provide the ability to generate dynamic queues at runtime.

Point-to-point interfaces are derived from the common JMS interfaces on which the earlier code description is based on. The following table gives an overview about the communication style specific interfaces in a point-to-point communication.

PTP Domain Interfaces	**JMS Common Interfaces** ***Preferred***
QueueConnectionFactory	ConnectionFactory
QueueConnection	Connection
Queue	Destination
QueueSession	Session
QueueSender	MessageProducer
QueueReceiver	MessageConsumer

Table 28: PTP domain interfaces and JMS common interfaces [SUNM 02s]

This section describes how to establish a point-to-point communication model based on the JMS technology. It shows how to create a message producing client as well as a message consuming client application.

In contrast to the general destination creation description of the architectural overview section (para. 7.1), it is necessary to create for both message producer and message consumer a queue as a destination.

```
1 Queue destination =
2    (Queue)context.lookup(“MessageQueue”);
```

Listing 29: Queue destination creation via context lookup

- Message producing client application:

  After the queue is defined, the next step is to create a message producing client. This can be done by creating a `QueueSender` object in the application.

```
1 MessageProducer sender =
2    session.createProducer(destination);
3 connection.start();
```

Listing 30: Message producer publishing to a JMS queue

  The start of the connection takes the effect that the message producing client is ready to send messages on to the JMS queue that can be consummated by the client application.

  In the next step, this example demonstrates how to send a text message. There are several types of messages. One of them is text. The types can be looked up in the JMS provider specific documentation.

```
1 TextMessage message = session.createTextMessage();
2 message.setText("my text message");
3 sender.send(message);
```

Listing 31: Sending a text message via a message producer

- Message consuming client application:

  In the same way as the message producer has been created it is also necessary to create the message consumer.

```
1 MessageConsumer receiver =
2    session.createConsumer(destination);
3 connection.start();
```

Listing 32: Creating a receiver via a session from a destination

  Starting the message consuming connection enables the client to fetch the messages from the queue that a sender has previously put into that queue.

  To receive messages, it is necessary to start the receiving mechanism of the message consumer.

```
1 TextMessage message =
2    (TextMessage)receiver.receiver();
3 String message = message.getText();
```

Listing 33: Receiving a text message via a receiver

Before the application shuts down, all open connections need to be closed. That ensures that no errors occur during the communication and no messages will be lost.

7.2.2 JMS Publish/Subscribe Model

The publish/subscribe model is realized in JMS technology through topics as message brokers. “A topic can be thought of as a mini message broker that gathers and distributes messages addressed to it” [SUNM 02, p.79].

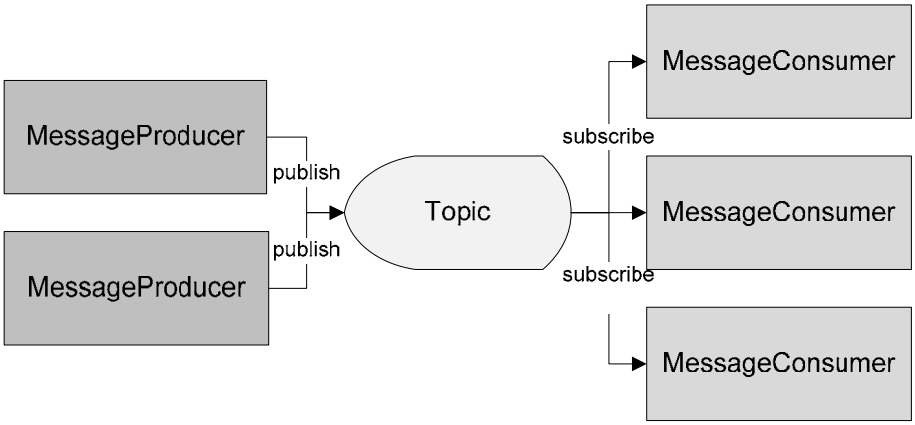


Figure 78: Publish subscribe model with JMS

Publishers and subscribers are active when Java objects that represent them exist in the memory. JMS clients use whether the common interfaces JMS provides in its specification of the Pub/Sub Domain interfaces. The following table gives an overview about which pub/sub domain interface maps to the common interface.

Pub/Sub Domain interfaces	**JMS Common Interfaces** ***Preferred***
TopicConnectionFactory	ConnectionFactory
TopicConnection	Connection
Topic	Destination
TopicSession	Session
TopicSender	MessageProducer
TopicReceiver	MessageConsumer

Table 29: Pub/sub domain interfaces and JMS common interfaces [SUNM 02]

Problems that can occur in the JMS public/subscribe system are that messages can be lost. Until all system participants know that there is a subscriber for a topic

available, it may take a second. Messages that are sent during that period of time are lost.

At the cost of a higher overhead, it is also possible to change a system to a durable subscriber application. If there is no durable subscriber available in the system, JMS retains the message till a subscriber was found in the system or till the subscription has expired.

The following text describes how to establish a publish/subscribe system by using topics that have been configured in the JMS provider.

```
1 Topic destination =
2    (Topic)context.lookup(“MessageTopic”);
```

Listing 34: Creating a topic destination via a context lookup

First of all, a topic needs to be created in the client side. It fetches the topic information through a JNDI lookup. After that is done, it is again necessary to differ between producing and consuming client application.

All the rest has to be done in the same way as already described in para. 7.2.1, by using the JMS common interfaces. For moving deeper in to the durable subscription technology, it is necessary to change the application structure on to the pub/sub domain interfaces. This is because this model cannot be applied into the common JMS structure.

7.3 Implementation of the Prototype

The implementation of the prototype has no influence on how the JMS clients process the messages on their side. The clients are implemented in Oracle CEP and BAM. The implementation decision came up using the publish/subscribe model because in a productive environment the number of processed messages is enormously high and the capabilities of message queuing does not fulfill the requirement on pushing all messages through the system.

On the other hand, there is a disadvantage in using the standard JMS description instead of moving towards a durable subscriber system. Some of the messages that are sent will be lost and cannot be used in the aggregation.

The next description explains the setup of the JMS topic on the OC4J JMS provider. Setting up a topic within that environment is not as hard as it looks at

first look. Administrators need to add several entries to the `jms.xml` configuration file that already contains some examples.

The file can be found in the 10.1.3.1 AS in the directory `%AS_HOME%\j2ee\home\config` and after modifying that file and restarting the server the new configuration is loaded and the JMS provider contains the topics used for messaging.

The following excerpt shows the important entries a `jms.xml` must contain being able to communicate through message queues and pub/sub style topics.

```
1  <?xml version = '1.0' encoding = 'UTF-8'?>
2  <jms xmlns:xsi=
3     http://www.w3.org/2001/XMLSchema-instance
4     xsi:noNamespaceSchemaLocation=
5     "http://www.oracle.com/technology/oracleas/schema/
6        jms-server-10_1.xsd" schema-major-version="10"
7     schema-minor-version="1">
8  ...
9     <jms-server port="9129">
10 ...
11       <topic-connection-factory
12          name="CEPTopicConnectionFactory"
13          location="jms/TopicConnectionFactory"/>
14 ...
15       <queue-connection-factory
16          name="CEPQueueConnectionFactory"
17          location="jms/QueueConnectionFactory"/>
18 ...
19       <topic name="Start_Service CEP Topic"
20          location="jms/cep/STARTService"/>
21       <topic name="FinishShipment CEP Topic"
22          location="jms/cep/FinishShipment" />
23       <topic name="DeliveryRefused CEP Topic"
24          location="jms/cep/DeliveryRefused" />
25 ...
26       <queue name="CEPWorkerQueue"
27          location="jms/cep/BPELWorkerQueue"/>
28 ...
29    </jms-server>
30 </jms>
```

Listing 35: JMS configuration within the jms.xml of an OC4J

8 Monitoring Business Activities

"(…) In today's competitive, service-oriented business climate, operational managers and executives demand visibility into the status of their business process networks (…)" based on "(…) key performance indicators (KPIs) in real time (…)" [KOCH 05]. This statement by Harpal Kochar describes today's business situation. Managers demand real-time information about their processes to be able to modify the processes for better execution as well as reacting to changes in the business environment (para. 1.1).

To achieve such a real-time system, the Aberdeen Group divides the monitoring of the business in two general sections. On the one hand, they take a look at the business process; on the other hand they also want to gather information about their technical information. Their terms for dividing the two topics logically are:

- Business Process Monitoring (BPM):

  "(…) Business Process Monitoring analyzes the execution of business processes created in a BPM system in near real-time, controls process execution in response to events that indicate a change in key performance indicators (KPIs) indicating business condition, and directs changes or alerts. (…)" [ABER 06]

- Business Activity Monitoring (BAM):

  Business Activity Monitoring "(…) monitors instrumented business processes and triggers notifications when a process-oriented event falls outside pre-established boundaries (…)". It "(…) focuses on awareness of and response to critical operational data and events (…)" [ABER 06].

Generally spoken, BPM takes care of the execution of business processes; BAM focuses on the data/event and technical side during the execution of a business process.

In contrast to this classification, Gartner has a generalized definition of BAM. According to the Gartner Group, the definition of BAM is "(…) the concept of

providing real-time access to critical business performance indicators to improve the speed and effectiveness of business operations." [CORR 02].

This definition includes that the business process need to be controlled as well as the data and events are part of the interest. Based on the experience with Oracle BAM, the definition after Gartner is the only one that provides a complete picture of the business. So when the next sections talk about BAM, it covers the process side as well as the technical side.

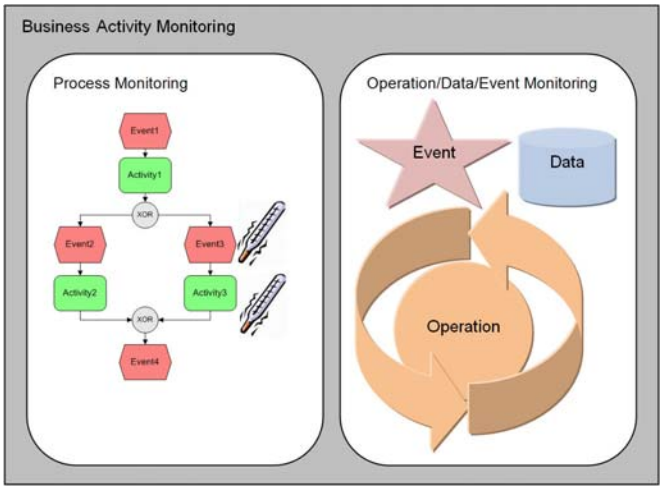


Figure 79: Business activity monitoring topics

BAM products have some key features which are common in most of the applications available on the market. The features are gathered by taking a look at the product overview of Systar's [SYST 08], Oracle's [ORAC 08] and IBM's BAM tools.

- Event-driven architecture (EDA)
- Real-time information on KPIs
- Data integration
- Drill down functionality
- Web based dashboard reporting
- Alerting functionality
- User/Roles management

Based on an EDA ,it is possible to provide real-time information about processes and services. The real-time information needs to be correlated based on predefined key performance indicators (KPIs).

In addition to the event based approach, BAM tools also provide the functionality to integrate existing data sources into their architecture.

Both the real-time event information and the data integration combined enable a drill down functionality for researching the root cause of an event.

Real-time information is displayed on Web based applications. The information can be also published through alerting functionalities (e.g. email, text message, pager).

On top of BAM, it is also important to put a user management ensuring, that only the application user who is supposed to retrieve the actual information, can see it. All others are not able to look at this information.

Having all these functionalities, the potential of BAM "(…) is at the business level, enabling new business strategies, reducing operating costs, and improving process performance (…)" [CORR 02]. According to that research paper, the implementation of BAM could reduce latency in decision making in transportation and logistics e.g. airline operation, package shipment.

8.1 Oracle BAM Architecture

The following sections provide an architectural overview about the components of Oracle BAM as well as a picture how these components interact with each other.

The architecture is based on Oracle BAM 10.1.3 which is implemented as a .NET framework running on MS IIS.

8.1.1 Oracle BAM Component Overview

Fig. 80 gives a basic overview of the Oracle BAM components. The figure does not contain all components included in Oracle BAM, but includes the main components used to gather events and report them to consuming devices and dashboards.

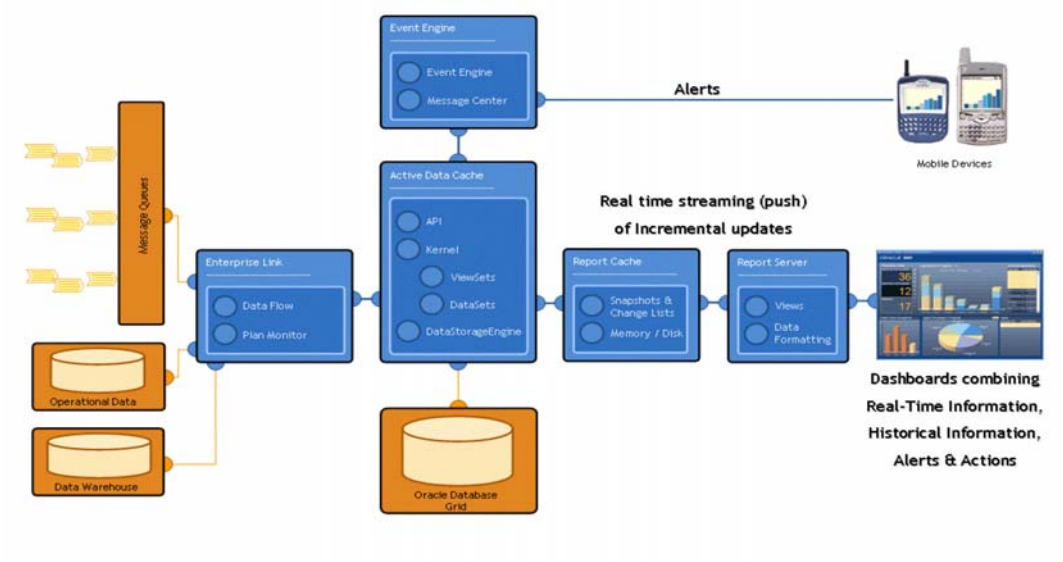


Figure 80: Oracle BAM basic component overview [ORAC 06d]

As a prerequisite for the component description message queues, operational data and data warehouses are given. The task of Oracle BAM is to gather the information from these sources and push it out in dashboards as well as on mobile devices through alerts.

Enterprise Link:

"(…) Oracle BAM Enterprise Link is an integrated high-performance environment that provides tools for extracting information from multiple sources, optimizing information for decision support and delivering it to users in formats that meet their business needs (…)" [ORAC 05b]. A selection of the Oracle Enterprise Link components is needed to fetch the data from the sources and push them in are:

- Design Studio:

  The Design Studio is an application to build population plans. This means that input sources can be populated. The event streams coming from the input sources can be modified and the data is stored in data marts or decision support databases. All this can be accomplished by drawing a visual flow diagram (e.g. fig.37 and fig. 43). By that way, basically input sources can be populated, stream data modified by queries and data marts or decision support databases can be created.

- Enterprise Link Server:

  The Enterprise Link Server contains a repository and a data flow service. The repository is a metadata database with information about retrieving, manipulating and displaying results. The data flow service is a server that collects the information via a pull mechanism.

- Oracle BAM Enterprise Link Admin:

  Enterprise Link Admin is an application to administer the Enterprise Link environment. It is possible to manage several servers with the application, perform tasks, and configure the repository and the data flow service. It is also possible to define security for users and user groups.

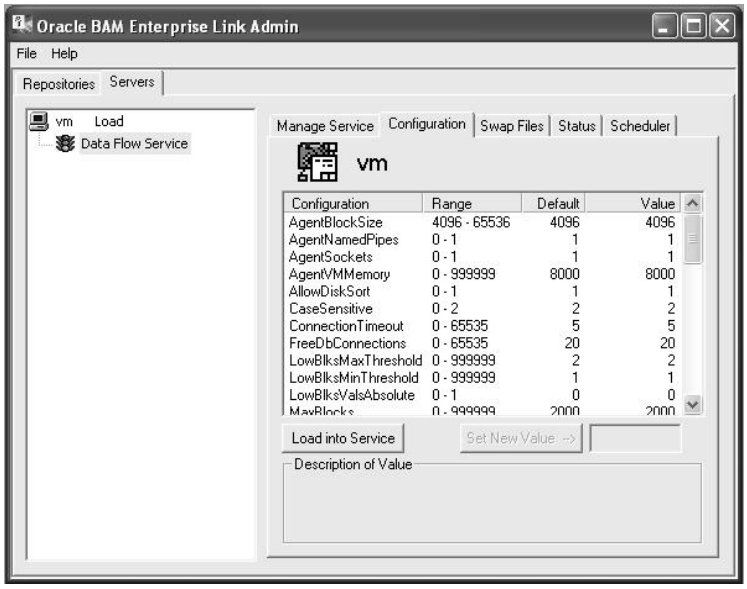


Figure 81: Oracle BAM Enterprise Link Admin

Active Data Cache:

"(…) Active Data Cache (ADC) is designed and optimized to provide access to current business information for event based reporting and alerting. It offers real-time intelligent analytical data cache capabilities.(…)" [ORAC 05b].

The ADC stores the data in an underling database. The data is sent on the one hand to dashboards as XML messages to update the real-time visualization, on the other hand to the Event Engine to send out alerts (e.g. email, text message, …).

Report Cache and Report Server:

The report cache and the report server act together to fetch the data from the ADC and generate reports. The report cache's main task is memory administration. It offloads the active data cache and maintains change lists immediately report any changes in real time to the report server. The report server administers the connections to the open reports and streams data to them in case they have changed.

Event Engine:

The Event Engine recognizes changes in data or time by continuously monitoring the ADC. "(…) It can detect changes in complex data conditions, as defined by the user. Rules can include a series of conditions and actions attached to an event (…)" [ORAC 05b]. The actions executed by the rules can be predefined (e.g. email or text message) as well as extended actions written in a public API.

8.1.2 User Roles in the Event Stream of Oracle BAM

After the previous section (para 8.1.1) fed into the components of Oracle BAM, this section describes, based on the user roles, how the components interact with each other.

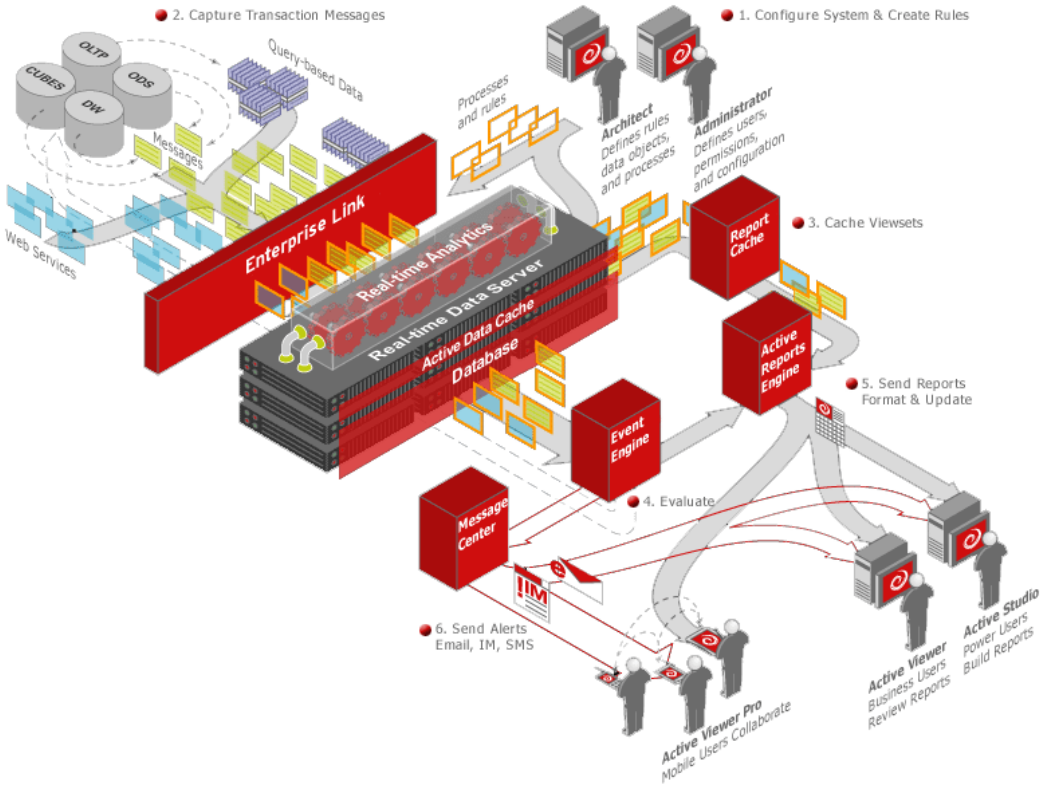


Figure 82: User roles in the event stream of Oracle BAM [ORAC 05a]

Before starting with the explanation of the user roles, information about how BAM fetches user information is needed is explained.

BAM is installed on a Windows operating system within an IIS. BAM's user administration is based on the domain, group user structure of windows. This means, that a BAM user needs to be an actual user on a Windows domain. He can only log on to the BAM system using his domain while being logged onto Windows. BAM matches the credential information to give each user the specific rights its role is assigned with. The following portion describes the standard user roles and shows that each role has its own application in BAM.

Administrator:

The administrator is the most powerful user role and has access to all features. This role cannot be deleted or modified. Users belonging to this user group define users, permissions and the server configuration.

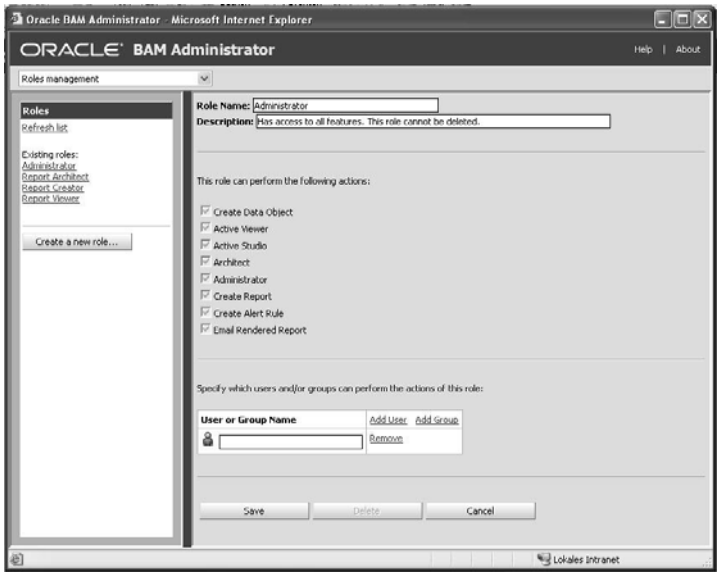


Figure 83: Administrator application – user roles management

The administrator is allowed to access the administrator application as well as all applications below. The application provides a lightweight browser application that enables him to execute the above described administration tasks.

Architect:

Architects have been granted the permission to create data objects and reports. They are not allowed to change system and user settings. Fig. 82 shows that this group defines rules, data objects and processes. An architect has the challenging task in the development of a BAM application to pull the data from the message sources and store them in data objects. He has to deal with two different applications, the BAM Architect application (fig. 84) within the Oracle BAM web application to create message sources and data objects as well as the Enterprise Link Design Studio (fig. 88) to model the data flow between the source and the destination.

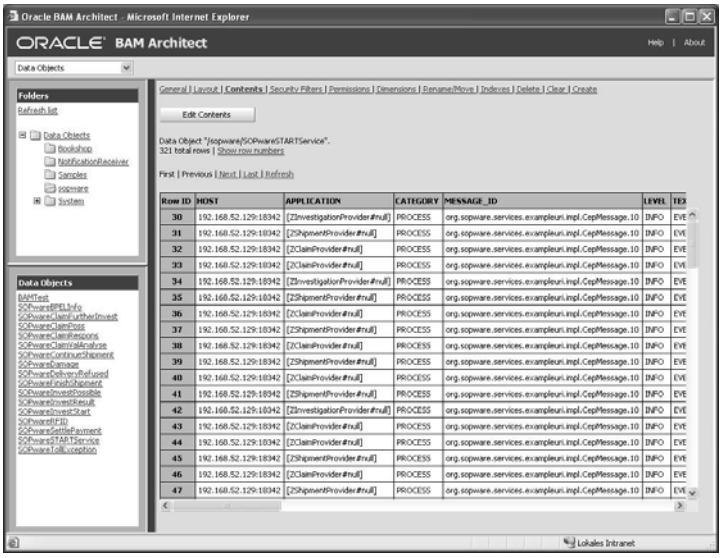


Figure 84: Architect application – data object management

Active Studio Designer

Users that have the report creator role assigned are allowed to create reports from existing data objects. They are able to design dashboards on top of the data structure from the administrator as well as creating alert functionalities.

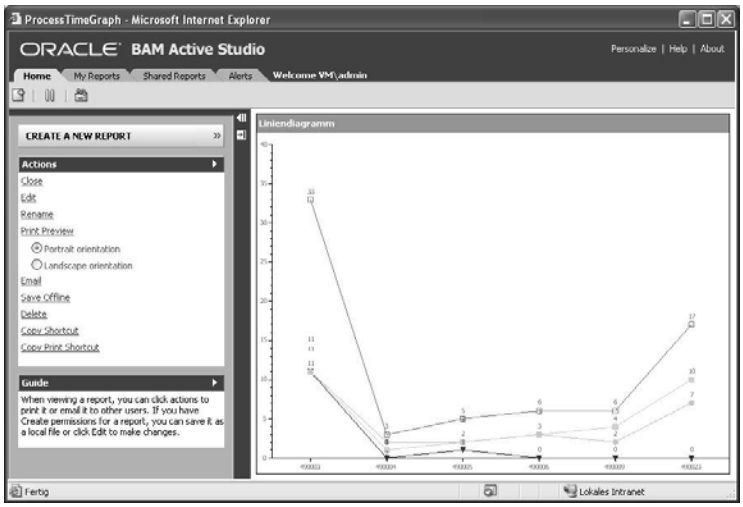


Figure 85: Active studio application – dashboard design

To design the reports and dashboards, BAM offers the Active Studio (fig. 85). This application enables the user to create the reports based on the data of the data objects. Reports can be combined by building dashboards.

Within this application, it is also possible to create the alert rules. If any KPI does not fulfill the requirements of the predefined values, the rule will be executed and the managers informed which are responsible for the alert.

Dashboard User / Alert Receiver

Dashboard users and alert receivers are allowed to monitor the reports created by the Active Studio Designer. Limiting the right to view an analysis is also included in the Active viewer application as the ability to personalize the view.

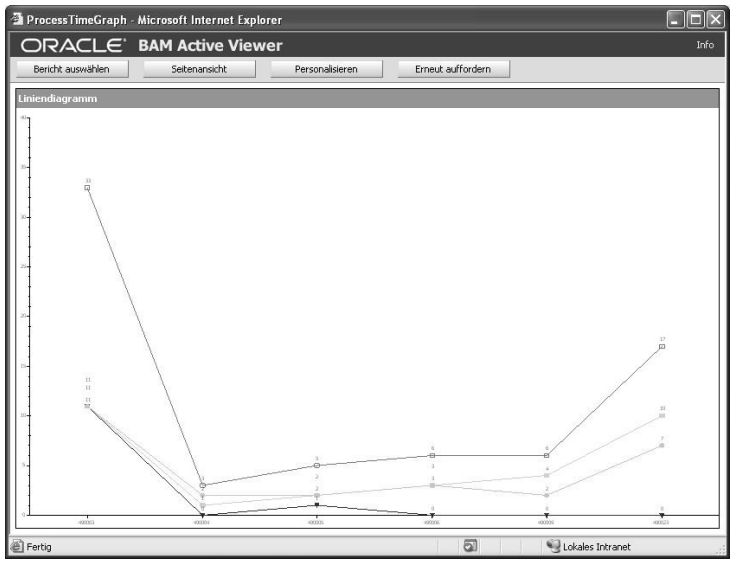


Figure 86: Active viewer - dashboard application

8.2 Oracle BAM Implementation

After the previous sections introduced the architecture and the role management of Oracle BAM this section describes the implementation based on the sample business process.

8.2.1 Event Streams and Persistence within Oracle BAM

This paragraph focuses on how the incoming streams the incoming streams are processed in Oracle BAM, for presenting the information in a dashboard.

Fig. 87 gives an impression how the processing of the event streams works. Enterprise message sources pick the incoming event streams and the data is stored in a database within data objects. To bring the data from the message sources to the data objects, there are enterprise link plans in between. The plans pull the data streams from the sources to the sinks.

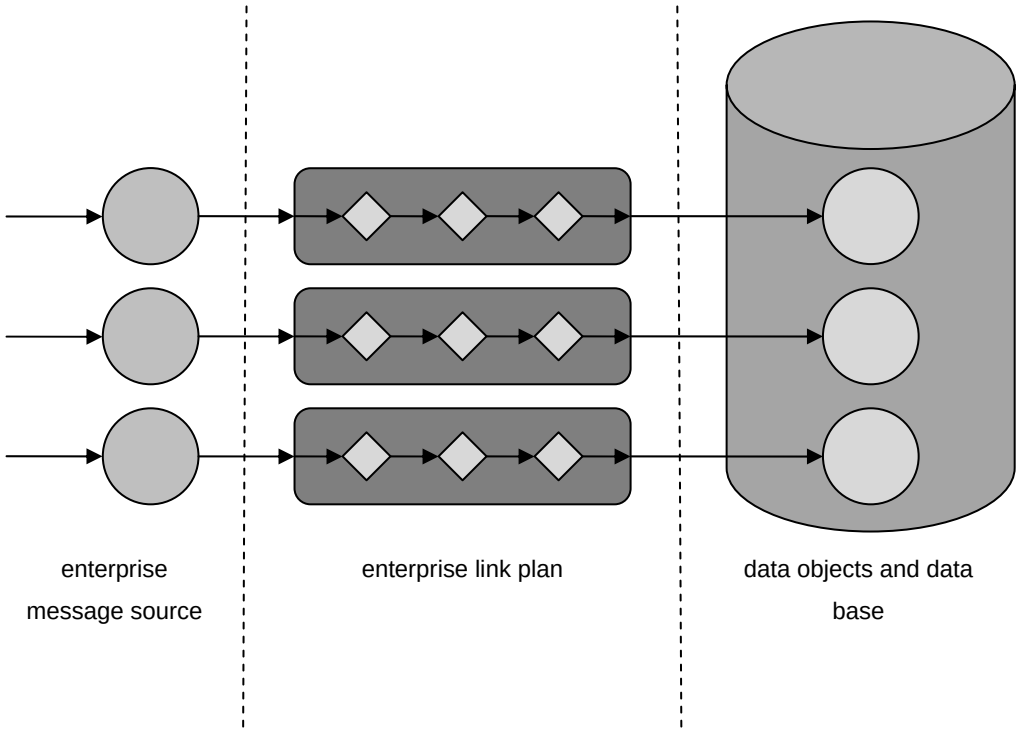


Figure 87: Event streams and persistence in Oracle BAM

Enterprise Message Source:

Based on the implementation decision of para. 6.3.1, BAM needs to pick the event streams from JMS topics per event type. To pick the streams up, BAM offers an adapter which can be used for JMS called enterprise message source. An enterprise message source contains basically two portions. On the one hand, there is a configuration necessary to get the message from the JMS provider. On the other hand, the message needs to be pushed in a certain template so that BAM can understand the content of the event. As an example of an enterprise message source, the following tables show the configuration as well as the message template of the "`STARTService`" message source.

InitialContextFactory	com.evermind.server.rmi.RMIInitialContextFactory
JNDI Service Provider URL	opmn:ormi://localhost:6003:home
TopicConnectionFactory Name	jms/TopicConnectionFactory
Topic Name	jms/sopware/STARTService
JMS Message Type	TextMessage
Durable Subscriber Name	STARTService
Client ID	STARTService

Table 30: Example Configuration for JMS Connection to OC4J JMS Provider

The configuration contains all credentials that are necessary to connect to the JMS provider retrieving the event as a JMS message from a topic.

The following table is an excerpt of an enterprise message source's data portion. It first shows the XSLT transformation which takes place when the event arrives at the enterprise message source and afterwards the template in which the event is going to be translated.

As an example: An incoming event is going to be transformed into a one layer XML structure, which means there is only one sub node under the root node. The root node in the example is "`msg`". The transformation puts all other attributes below this element. An enterprise message source than maps the fitting content of the XML tags to the corresponding attribute in the data flow (e.g. the content tagged in the XML file within "`host`" is mapped to the data flow attribute with the name "`HOST`", the XML tagged content of "`app`" is mapped to the data flow attribute with name "`APPLICATION`").

After this configuration is done, Oracle BAM is able to understand incoming JMS messages and convert them in the Oracle BAM data flow.

<table>
<tr><th>Name</th><th>Flow name</th><th>Type</th><th>Max size</th><th colspan="3">Formatting</th></tr>
<tr><td rowspan="2">DATA</td><td rowspan="2">DATA</td><td rowspan="2">String</td><td rowspan="2">2000</td><td colspan="3">XML Formatting</td></tr>
<tr><td colspan="3">XSLTransformation</td></tr>
<tr><td colspan="4" rowspan="8"></td><td colspan="3"><?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
…</td></tr>
<tr><td>Path</td><td colspan="2">/msg</td></tr>
<tr><td>Column values are contained in</td><td colspan="2">Tags</td></tr>
<tr><td>Tag name</td><td>Dataflow name</td><td>Max size</td></tr>
<tr><td>host</td><td>HOST</td><td>25</td></tr>
<tr><td>app</td><td>APPLICATION</td><td>50</td></tr>
<tr><td>cat</td><td>CATEGORY</td><td>25</td></tr>
<tr><td>messageid</td><td>MESSAGE_ID</td><td>100</td></tr>
</table>

Table 31: Excerpt of a XSLT transformation and a XML – data flow mapping

Enterprise Link Plan:

To transfer the messages now from the message sources to their destinations - the data objects – several plans need to be created. Plans are multi step processes reflecting the data flow [ORAC 05c]. The tool to create plans is the Enterprise Link Design Studio. Within this, the design studio plans are created, tested and deployed. Plans contain multiple steps to manage a data flow. Because of the drawback of using a .NET framework to a pull data from a Java based application, it is not recommended to create complex plans even it is possible.

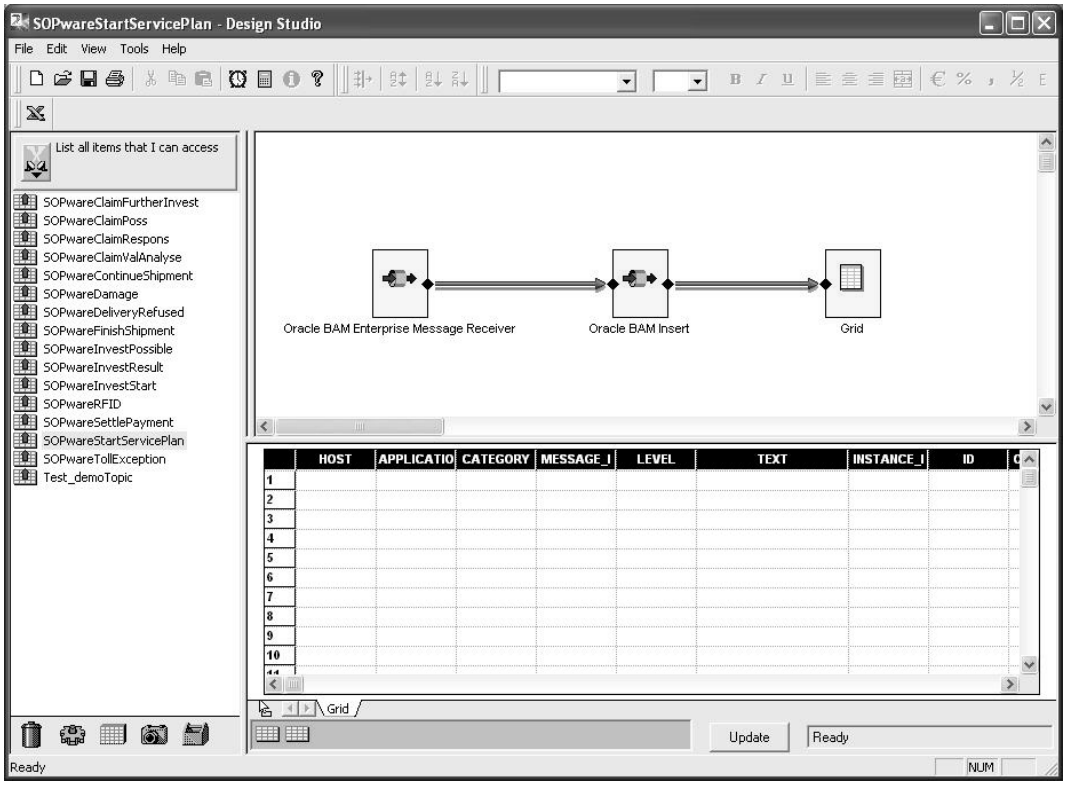


Figure 88: Oracle BAM Enterprise Link Design Studio

Fig. 88 shows Oracle Enterprise Link Design Studio which enables users to create plans. In the example implementation, the decision came up to create, for each enterprise message receiver, a single plan because it is possible to manage plans within the administration interface of BAM. This means plans can be started and stopped independently and removed from the system without having downtime on a different plan.

Another point the figure shows is that the data flows from the enterprise message receiver directly to an Oracle BAM Insert. This means the data is pushed 1 to 1 from the message source to the data object; No transformation in the data flow takes place. For testing purposes, there is an additional component in the plan called grid; the grid enables the Design Studio to test plans without having them deployed.

After a plan is tested and published to public, it is managed and monitored by the BAM Administration application (see fig. 83). This application enables administrators to start and stop the plans as well as to monitor them.

Data Objects:

"(…) Data objects contain the information that displays in reports created in Active Studio (…)" [ORAC 06]. Data objects are created within the BAM Architect application. After the events are processed through the data flow, they are stored in data objects. To be able to store the data in the data objects, the objects must have the same attributes as the event flow that arrives at the endpoints. In the example implementation, according to the fig. 87 and 88 this means, that the data objects need the same attributes the enterprise message sources push on the data flow.

Field name	Field ID	Field type	Max length
HOST	_HOST	string	25
APPLICATION	_APPLICATION	string	50
CATEGORY	_CATEGORY	string	25
MESSAGE_ID	_MESSAGE_ID	string	100
LEVEL	_LEVEL	string	25
TEXT	_TEXT	string	1530
ID	_ID	string	10
OPERATION	_OPERATION	string	50
STARTUP_TIME	_STARTUP_TIME	string	35
PROCESS_ID	_PROCESS_ID	string	5
PROCESS_URL	_PROCESS_URL	string	100
PROCESS_OWNER	_PROCESS_OWNER	string	5
PROCESS_VERSION	_PROCESS_VERSION	string	5
TIME_STAMP_PROCESS	_TIME_STAMP_PROCESS	string	30
BPEL_TIME	_BPEL_TIME	string	100
INSTANCE_ID	_INSTANCE_ID	integer	-

Table 32: Example of data object layout

The example layout of data objects shows the structure of the database table as well as the corresponding field names to the data flow name. Comparing table 30 and 31 points out that the message sources put an attribute with the name "`HOST`" on the data flow and without any transformation work the data object needs to pick the attribute up again in a field name called "`HOST`" to be able to store it in a database field with the id "`_HOST`".

The proof that the data objects with their data flow names match the database tables is shown in the following code snippet.

```
1 CREATE TABLE  "_SOPwareSTARTService"
2    (  "_HOST" VARCHAR2(25 CHAR),
3  "_APPLICATION" VARCHAR2(50 CHAR),
4  "_CATEGORY" VARCHAR2(25 CHAR),
5  "_MESSAGE_ID" VARCHAR2(100 CHAR),
6  "_LEVEL" VARCHAR2(25 CHAR),
7  "_TEXT" VARCHAR2(1530 CHAR),
8  "_ID" VARCHAR2(10 CHAR),
9  "_OPERATION" VARCHAR2(50 CHAR),
10 "_STARTUP_TIME" VARCHAR2(35 CHAR),
11 "_PROCESS_ID" VARCHAR2(5 CHAR),
12 "_PROCESS_URL" VARCHAR2(100 CHAR),
13 "_PROCESS_OWNER" VARCHAR2(5 CHAR),
14 "_PROCESS_VERSION" VARCHAR2(5 CHAR),
15 "_TIME_STAMP_PROCESS" VARCHAR2(30 CHAR),
16 "SysIterID" NUMBER(19,0) NOT NULL ENABLE,
17 "SysIterTrID" NUMBER(19,0) NOT NULL ENABLE,
18 "SysIterTotalsChild" NUMBER(10,0)
19      DEFAULT 0 NOT NULL ENABLE,
20 "_BPEL_TIME" VARCHAR2(100 CHAR) NOT NULL ENABLE,
21 "_INSTANCE_ID" NUMBER(10,0) DEFAULT 0 NOT NULL ENABLE,
22  PRIMARY KEY ("SysIterID") ENABLE
23   )
24 /
```

Listing 36: Data object extracted from the DB as DDL

The DDL captured directly from the database shows a corresponding table to the "SOPwareSTARTService" data object with the table name "_SOPwareSTART-Service". The same behavior occurs with the data flow names; e.g. data flow name "HOST" is captured in the database as "_HOST" column name.

8.2.2 Presentation of the Data within BAM Dashboards

Based on the data objects created in the paragraph before, it is possible to create reports, published in a BAM dashboard. According to the information published from the example business process to the BAM side, it is possible to create technical as well as business oriented reports. The example report shown in fig. 89 provides information about the instances of the business process described in chap. 3. It also includes a drill down functionality to get detailed information about a process instance for analyzing deviation. Depending on the expected information which has to be provided, the complexity of creating reports can become extremely high. Because of this, just the creation of a basic "Bar Char" is shown, to provide a short overview about developing reports in Oracle BAM. Detailed information about report design can be seen in the Oracle User guide [ORAC 06c].

8.2.2.1 Monitoring of the Business Process

This paragraph concerns the technical information of the business process. The report in fig. 89 provides information about the process time of running instances.

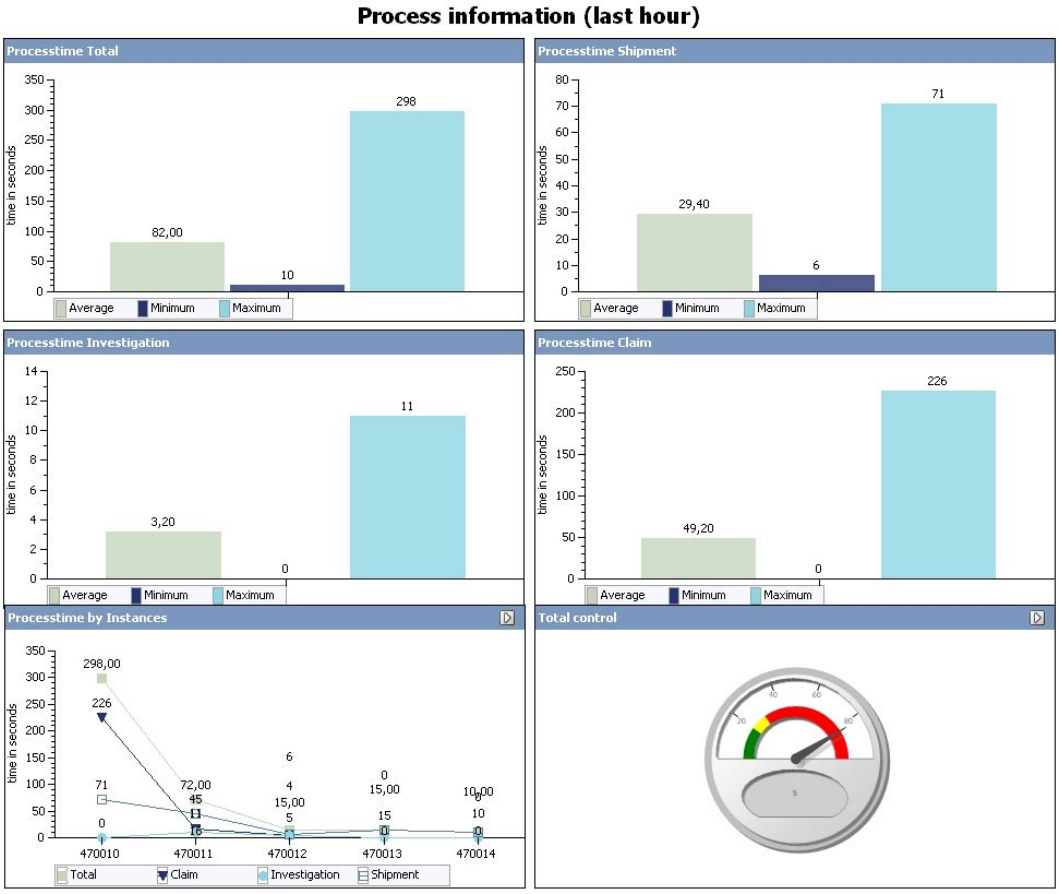


Figure 89: Process report

For each part of the process (shipment, investigation and claim) the required time needed for simulation (minimum, maximum average), based on the test cases in para. 3.2.8, as well as the total processing time is shown in a bar chart. The line chart provides information for each instance and also the ability to drill down. Drill down means the possibility to move from summary information to the detailed data that created it [TECH 08]. In this case, the user has the ability to get detailed information for one process instance, by clicking on it. That enables detailed analyzing of deviations and allows reacting on errors. Drill down can be performed with data from different data objects and applications, e.g. the “Total Time” for a process instance. Usually it is 15 seconds, but fig. 89 shows that the process instance 470010 needed 298 seconds for processing. For identifying the error source or the operation which is responsible for that, the drill down

functionality can be used. Fig. 90 shows the drill down report for the process instance. This view allows the user to take a look on the details of the instance. In this case, a detailed list of operations which were called and their execution time is presented. That perspective makes it possible to find out, which operation was responsible for the bad execution time. Based on the result of the drilldown it has to be decided, if actions are required to solve the problem or if it was only a sporadic problem.

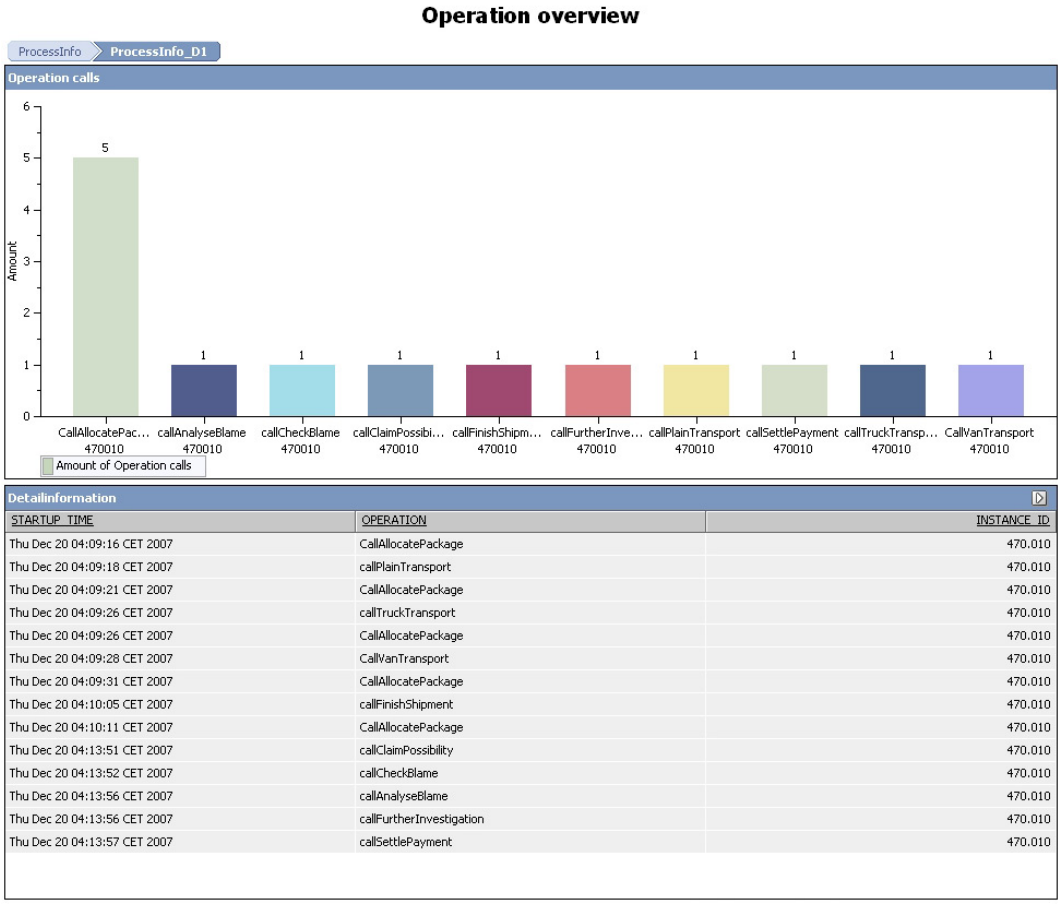


Figure 90: Drill down process report

The sixth chart in fig. 89, named “Total Control” shows if the execution time of the currently running instances is within a defined threshold. The “Range Gauge”-chart, used for this, calculates the average execution time of the instances and displays the current status. Green shows, all is OK, orange is for warning and red for a possible error which has to be solved. It also allows a drill down functionality to view the values. In the example shown in fig. 89, the business process 470010 is responsible for the red indicator and the bad average time.

As mentioned before, BAM allows creation of multitude charts, which can present information in versatile way. So it has to be decided for every use case how the information has to be presented in a BAM dashboard to provide an ordinary understanding to the users. After it is known what information has to be presented, the necessary data has to be extracted out of the processes or applications and sent to BAM. Over here it can be used for generating reports and presenting the data to users to work with.

8.2.2.2 Creation of a Bar Chart

This paragraph describes the development of a simple BAM report. Based on the event “SettlePayment” (para. 3.2.6.4.5), the amount of compensation which was paid and the amount the customer expected should be presented to analyze the deviation. This paragraph just describes the basic steps. All additional settings, filter-options and design changes are waived at this point.

The development of the report has to be done in the BAM Active Studio, as described in para. 8.1.2. It has to be accessed via the BAM Start Page (http://localhost/OracleBAM/). The function to create a new report (fig. 91) can be accessed directly on the startup screen.

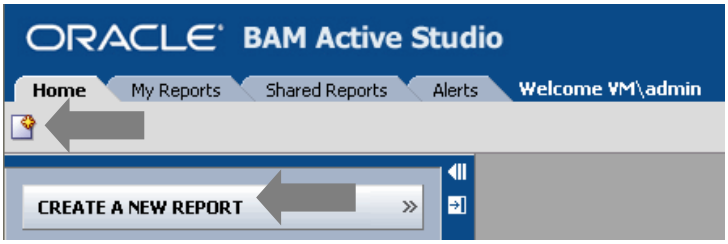


Figure 91: Create new report

After that, the layout of the report has to be chosen. Oracle BAM offers a multitude of different layout designs. That allows the creation of a report with several charts in it, as shown in fig. 92. For this example just one chart area is necessary, therefore the simple layout is chosen.

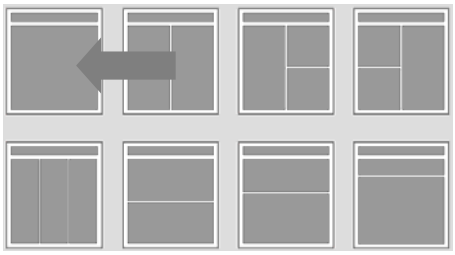


Figure 92: Report layouts

As mentioned, Oracle BAM offers a wide spread of different chart types (fig. 93), e.g. lists, bar charts, line charts, area charts, combo charts, pie charts, range gauges, 3D charts, matrix charts, excel exports, etc. In this case, the simple bar chart is sufficient and has to be selected by clicking on its icon.

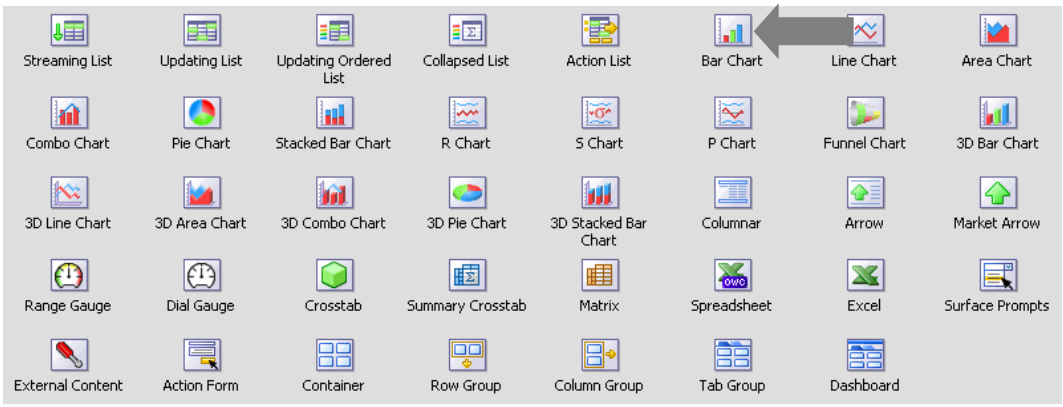


Figure 93: BAM chart types

After selecting the chart type, a wizard guides the user through the next steps. First the data object or the data objects on which the report has to be based on must be selected. For this example the “SettlePayment” data object is selected. The creation of a data object is described in para. 8.2.1. The next step is to create a statement which selects the values out of the data object. The statement includes the fields, group functions and summery functions and has to be created with the user interface of BAM shown in fig. 94. Internally BAM uses a SQL statement, because the data object is similar to a data base table as described in para. 8.2.1. The example has to show the values for every instance of the business process where compensation was paid. Therefore the field “`Instance_ID`”, representing a business process, is the group value. The field for expected payment of the customer is “`Amount`”. The field for the actual payment value which was detected during investigation is the “`Supposed_Amount`”. Because of the group value

"`Instance_ID`", both fields have to be selected and will be shown in the report for each process instance

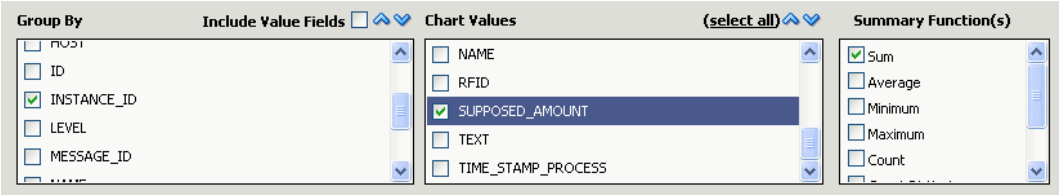


Figure 94: BAM statement generation

After committing these, additional filters or data objects can be added to the chart but that is not necessary for this example. It is also possible to change the colors of the chart, add legends, etc.

To provide Dashboard Users (para. 8.2.1) the possibility to access the report it has to be saved as "Shared Report". That enables users with lower permissions to access the example report created in the BAM Active Viewer as shown in fig. 95.

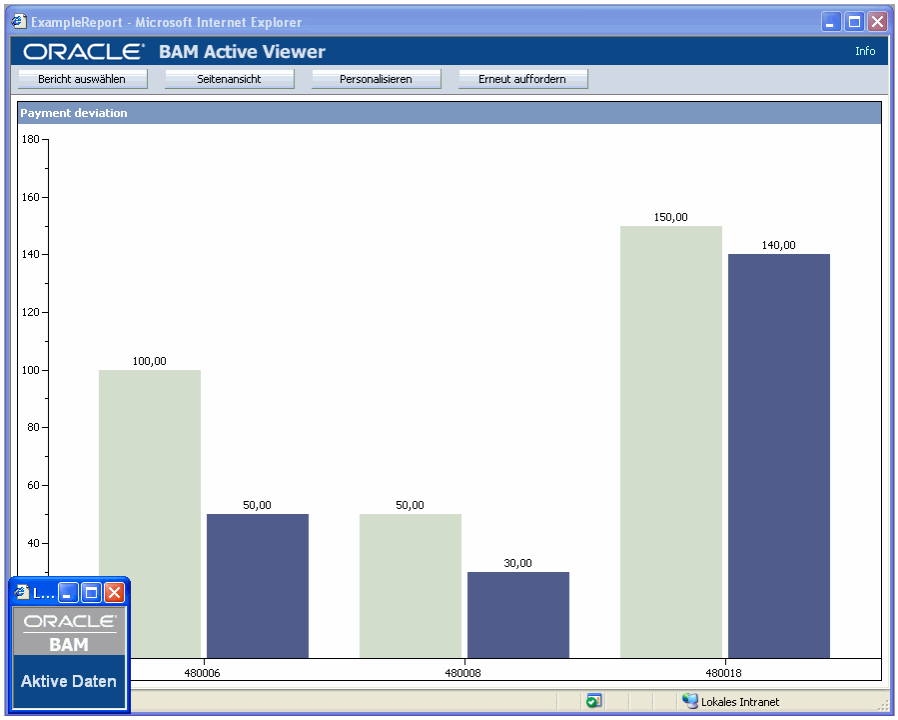


Figure 95: Example report

9 Complex Event Processing

The previous chapters dealt with the basics of ESP. They explained how an event is generated (para. 4.2) and processed (para. 6.3) in a multi component system environment up to the point of displaying, described in para 8.2.1. and para. 8.2.2.

To distinguish this section from the previous, the introduction provides an overview of what a complex event is in contrast to a basic event. The introduction also discusses the difference between database management systems (DBMS) and CEP and their common base.

9.1 Introduction into CEP

"Traditional database management systems (DBMSs) expect all data to be managed within some form of persistent *data sets*. (…) By nature, a stored data set is appropriate when significant portions of the data are queried again and again, and updates are small and/or relatively infrequent. In contrast a data stream is appropriate when the data is changing constantly (often exclusive through insertions of new elements), and it is either unnecessary or impractical to operate on large portions of the data multiple times." [BABU 01; p.1].

As a consequence, CEP offers a solution opportunity to query data streams, also called event streams. CEP queries real-time data from basic events to create complex events. Complex events are the result of the queries on the basic events. Fig.96 pictures the situation that has been described before in para. 3.3.2. The assumption of the figure is that a single event stream is sent over the system and is queried through CEP to get a result out of the streams. The query result is the complex event.

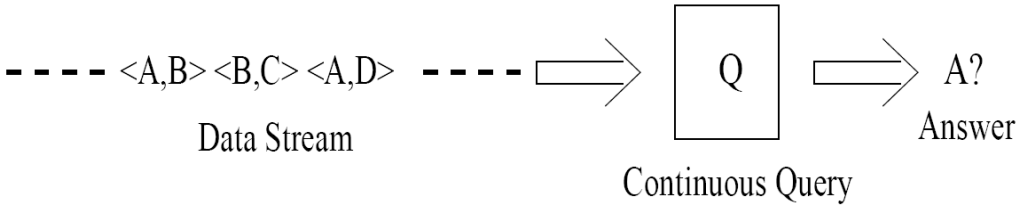


Figure 96: A continuous query *Q* over a single data stream [BABU 01; p. 4]

Supposing that there is not only a single stream, as in the fig.96, but many event streams as in the example implementation, the scenario looks like fig. 97. Many events come into the system (fig. 97, see also fig. 42) at different times and need to be queried to get a complex event as a result. The complex event is a combination between the basic events and is the conclusion from their information.

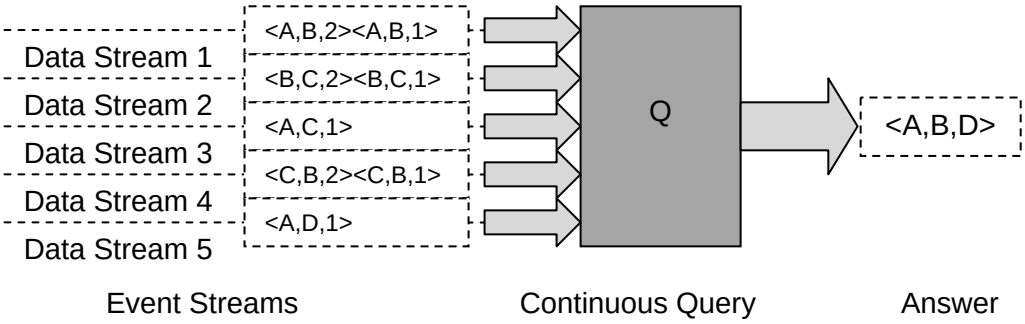


Figure 97: Multiple incoming event streams

Compared to a DBMS, CEP has a constraint on the time events need to be kept in the system providing the ability to create a complex event. A CEP engine has the functionality to get only one event a time. To create queries over two events, the first event needs to be kept in memory till the query needed events are also in the system. In CEP, this memory is considered as a sliding window [BABO 02].

9.2 Querying Streams with CQL

To be able to query a stream, stream data needs to be represented as relations within a CEP engine. To achieve this relational view, the STREAM project of Stanford University introduces some semantics.

9.2.1 Abstract Semantics of CQL

The semantics of CQL is based on three operations:

- *stream-to-relation*: operator that produces a relation from a stream
- *relation-to-relation*: operator that produces a relation from a single or multiple relations
- *relation-to-stream*: operator that produces a stream from a relation [ARAS 05]

To understand how the operators interact with each other, fig.98 gives an overview on the relationship between them.

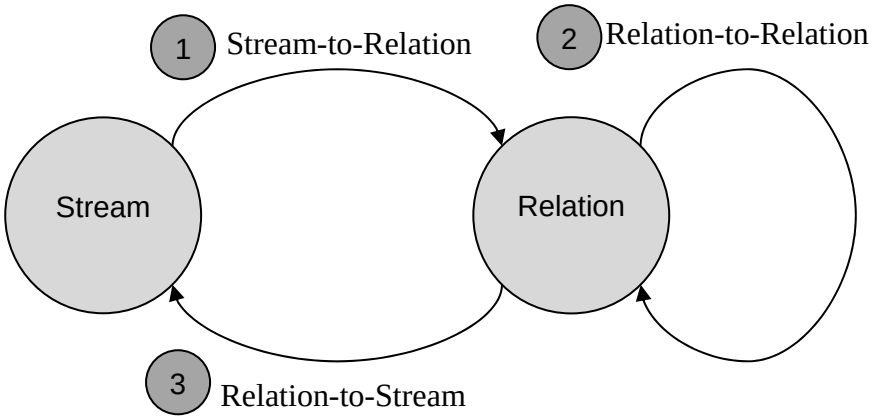


Figure 98: Abstract semantics of operators and classes (derived from [ARAS 05, p.7])

1. The *stream-to-relation* operator picks an incoming stream and transforms it into a relation.

2. The *relation-to-relation* operator takes one or more relations as input and creates a new relation.

3. The *relation-to-stream* operator takes a relation and transforms it back into a stream

Fig. 98 describes in simple words the way that all incoming event streams are transformed into relations because it is only possible to query relations, not streams. After relations have been queried and a new result relation has been created, the result has to be transformed back into an event stream.

Each input stream contains attributes. An input stream is mathematically considered as a tuple. A tuple is defined as a finite sequence of objects of a specific type. It is possible that the same object is included more than once. In the following, there is a formal definition of the n-tuple [MATA 00]:

$$n = 0 : () := \{\}$$
$$n = 1 : (x) := \{(x), \{x\}\}$$
$$n > 1 : (x_1, ...x_n) := \{(x_1, ...x_n), \{x_n\}\}$$

An example for a tuple used in the implementation example is a sample for an incoming XML event as well as for an insert statement through the JDBC interface of a CEP engine.

9.2.2 Sliding Window Consideration on the Example of a SELECT Statement

If just the semantic is considered in reality, it is only possible to query incoming events that enter the system at the same timestamp. Because of common computation limits on e.g. network limitations, it is not possible to ensure that the events needed for a SELECT statement arrive at the same time stamp in the system. After they have entered the system, been transformed into a relation and have not yet been queried, the information is lost. To achieve a system that is able to query not only events that enter at the same time stamp, it is necessary to introduce buffers.

Buffers enable continuous queries to open up a time frame. That means queries can be run up to a certain point of time to find pattern matches resulting in new event streams. The new event streams contain the complex event which is based on the result of several basic events.

Definition:

S(1,2): Input Event Streams

q(1,2): Queue of incoming events

W(1,2): Window buffer

q(4,5): Queue of events waiting for the join

q(6): Queue of events waiting to be selected

q(7): Queue of events with the selected attributes

S(3): Output event stream

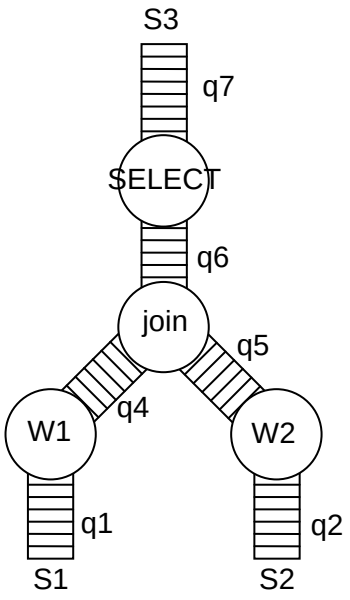


Figure 99: Query plan for simple SELECT

Fig. 99 shows a query plan within a CEP engine for a simple SELECT statement. Two input streams (S1 and S2) are buffered in the windows (W1 and W2) to provide the date, necessary for the join operation. The join brings the two relations of the streams together and the select creates a new complex event as an output stream (S3). The event queues between the nodes (q1 to q7) picture the event stream.

9.2.3 Issues on Querying Plans

The previous example introduced a simple query on two streams. Thinking about a real life scenario, a single query will be very unlikely queries having their own buffer in the memory is very inefficient. To improve the performance of the application, window sharing is an option to resolve this issue.

In our example in fig.99, it means if an additional query is necessary in the system with a new output stream S4 based only on the input stream S1, the time window W1 is reused assuming that the query needs the data over the same amount of time. [ARAS 03]

Other proposals about how to resolve memory issues as well as reducing time during the query plan execution are discussed in the research paper “Query Processing, Resource Management and Approximation in a Data stream Management System” by Motwani et al. [MOTW 03]. The paper provides some algorithmic ideas how to improve the performance of CEP.

9.3 Introduction into Oracles CQL

The continuous query language (CQL) is derived from the structured query language (SQL). The main difference between them is that CQL queries continuously incoming stream data while SQL queries historic, finite, persistent data.

The following example is based on Oracle CQL and provides an introduction into the structure of CQL. The examples are based on the experience while working with CEP, because there is no API available, yet.

9.3.1 Stream Creation

This paragraph introduces the creation of a stream. As already mentioned in paragraph 9.2.1, a stream is translated in CEP into a relation. As a consequence of this, the creation of a stream is similar to the creation of a database table.

The main difference between the creation of a database table and a stream is that a stream needs not only the information about the attributes (tuples), but also information about the source of the stream. Because of this, the creation of a stream is divided into two steps. First the creation of the stream relation, including all attributes, and secondly the declaration of the stream source.

- Create stream

```
1 CREATE STREAM basic_stream(TIME_STAMP integer, FIRST_NAME
   char(25), LAST_NAME char(30), PASSWORD char(10));
```

Listing 37: Create stream statement - example

- Alter stream

```
1 ALTER STREAM basic_stream ADD SOURCE PUSH;
```

Listing 38: Alter stream add source statement - example

9.3.2 Query Creation

According to the SQL query definition of Mike Chapple, “Queries are the primary mechanism for retrieving information from a database and consist of questions presented to the database in a predefined format” [CHAP 08]. CQL offers similar structured expressions to retrieve information from streams.

In contrast to SQL, CQL needs to also define the sliding time window as well as the output stream (para. 9.2.2) for the query. To define this information, CQL is extended for a parameter in the create query statement and needs an alter query statement that points to the output stream destination.

- Create query

```
1 CREATE QUERY q_basic_event AS SELECT * FROM basic_stream
   [now];
```

Listing 39: Create query statement -example

The create statement example shows a simple query with the sliding window parameter [now] at the end. This parameter defines the time frame the memory buffer stores the incoming events for queries.

- Alter query

```
1 ALTER QUERY q_basic_event ADD DESTINATION
   “<EndPointReference><Address>file:///home/cep/test.txt</Add
   ress></EndPointReference>";
```

Listing 40: Alter query statement - example

The alter query add destination statement defines the destination the output stream takes. In the listing, the output stream goes to a test file. There are also other destinations possible e.g. JMS destinations.

```
1 ALTER QUERY q_basic_event START;
```

Listing 41: Alter query start - example

A specialty compared to SQL is also the start expression of a CQL query. A CQL query needs to be started in order to enable the continuous querying mechanism. A SQL query is always fired only once against a database, a CQL query needs to be running all the time because of the possibility of incoming events at all times.

- CQL execution

At the end of the CQL setup expressions (e.g. creation of streams and queries); the CQL engine needs to be started. The “ALTER SYSTEM RUN” statement starts a thread in the CEP engine which is responsible for the execution of the operations and plans.

```
1 ALTER SYSTEM RUN DURATION=0;
```

Listing 42: The run statement executes all statements and algorithms

The duration parameter in this statement tells CEP that it has to stay up and running during the test. It is only a testing parameter and will not be used in a productive environment.

- Insert query

Streams vary from their source types in many cases. Examples for streams are XML formatted JMS streams, network protocols as well as direct insertions into a CQL stream via JDBC. The following example introduces a stream insertion on the previously created "`basic_stream`".

```
1 INSERT INTO basic_stream VALUES(1000000000, 1, “Shailendra”,
2    “Mishra”, “fmwadmin”);
```

Listing 43: Insert query - example

In this example, there is a tuple inserted with the values "1" for the client time stamp, as well as the first name, last name and the password. The first attribute of the tuple has to be the time stamp. This timestamp belongs to the time domain and is important to enable the CEP algorithms to query the data. CEP is assuming that incoming events are ordered by time. All events need to belong to a certain time domain, which gives them an increasing time stamp. If events arrive at CEP with a different order, the algorithm is not able to execute the queries and fails.

9.4 Goals and Possible Fields of Application

To take a look on the possible fields of the application, it is necessary to notice a difference between traditional tracking of business activities and the approach CEP offers for creating a predictive business application.

Traditional message tracking as the NR (para.6), in a direct combination with BAM, handles deals with the topic of collecting the events that are currently in the system to provide real time information for monitoring purposes. Real-time information is useful to react faster to business activities since these events have already occurred in the system and there is no way to prevent these activities.

In contrast to this, CEP provides the ability to predict what will happen in the future of the business by applying patterns. Patterns are schemas based on the research of the past business process activities. A pattern is found when the research discovered a certain schema of events that always lead to a failure of the business process. This pattern can be applied to CEP to make it possible to predict the result by selecting the events of the pattern. If all of the events of the pattern occur within the time window, CEP is able to fire a new complex event that alerts

the staffer in advance about a process failure in the future. This provides him the ability to react proactively e.g. on a lost package.

An example for these patterns is in our case similar to the business process patterns dealing with delivery latency or infeasible delivery. The pattern is described in the next paragraph.

9.5 Integration of Oracle CEP in the Existing Architecture

The integration into the existing architecture works for the simulation with the test cases (para. 3.2.8) through the NR (para.6). The NR fires events to CEP, where the information is correlated to send out result streams. The result streams are picked up by BAM to display the events and react to the information.

The following section describes the use cases which are based on external influences in a pattered structure. Introducing first into the context of the example implementation enables a better understanding of the external influences on the business process. External influences are in the case of the implementation traffic reports as well as weather forecasts.

Vehicles are moving through certain areas to deliver packages from a starting point to the destination (para. 3.1.3). During the tour they are using transportation infrastructure which is also used by other vehicles. Because it is not predictable how many vehicles are attending the traffic, congestions occur. Congestion is reported by traffic reports like TMC. While capturing this traffic information and keeping track about the state where the actual vehicle with the packages are, it is possible to predict if a vehicle is going to enter an area with traffic congestion in the next step. This enables a staffer steering the delivery process to warn the truck driver and suggest an alternative route resolving delivery latencies.

The same argumentation as for traffic congestion can be applied when the weather is also considered as an external process influence. Bad weather conditions put an influence on the delivery process. Worst case scenarios are areas of flooding as well as earthquakes and other unforeseen weather scenarios where the delivery of packages is impossible. A common example for latencies is thunderstorm warnings causing delays at airports.

To be able to match the following abstract description about external process influences, see also para. 3.2.6.5, which describes concrete events that are designed in the example implementation.

9.5.1 External Process Influence Pattern

For the show case (para. 3.2.8.3), a pattern structure describes the idea behind the usage of CEP in the context simulation with the business process. The structure is derived from the “Loss Pattern” which is one of the results of the CEP class in 2007 at the University of Applied Sciences, Regensburg presented at the 5th expert meeting on BPM/BAM/CEP/SOA and EDA in Regensburg by Christian Silberbauer, et al. [SILB 07]

9.5.1.1 Introduction

The external process influence pattern describes how to detect external influences on a logistics related delivery process. The processes need to reveal information about their state as well as other events needed to be captured to provide the information about external process influences. External process influences occur continuously during the process execution. The external events are aggregated with the process events, what enables a forecast, if the process execution succeeds or if any failures occur. The forecast is based on KPI’s which has been defined in advance. The external process influence pattern allows forecasting the process success as well as reacting to process failing predictions in real-time.

9.5.1.2 Example

A logistics related delivery process is a complex process containing many steps until it is completed. For example, a package has to be taken over several routes via several means of transportation till it arrives at the destination. Within these routes, there are many influences which cannot be considered at the process design but have influences on the success of the delivery process. Such influences (para. 3.2.6.5) are e.g.:

- Traffic congestion on the planned transportation route
- Accident of the transporting vehicle
- Bad weather conditions within an area the planned route passes

Not all issues are considered in the example, but these are the major influences on a delivery process.

9.5.1.3 Context

Delivery processes are executed and traced by sending out events at certain processing points. Other information enters the system via several event streams. During when the process is executed, there are various reasons why the process execution does not complete.

9.5.1.4 Problem

The interest of a business analyst is related to the execution of the business process. The main questions in case of a failed process are:

- At which step does the process fail?
- What is the reason for the business process failure?

9.5.1.5 Solution

The CEP platform collects events fired by the business process (process events). It also gathers events coming from external sources (external events). So there are basically two events needed before being able to predict a business process failure.

Process Event:	External Event:
• Key identifying an order • Affected item identifier • Location of the business process • Physical area where the event happens	• Key identifying an order • Physical area where the event happens • Degree of failure probability

Table 33: Events of the "External Process Influence Pattern"

The forecast of a loss is based on the area (*areaCode*) match of a process event, on the next processing step, with the area of an external event as well as a matching key that identifies the order (*timeStamp*).

9.5.1.6 Structure

The solution is already introduced (para. 3.2.6) into the two types of events that are necessary to aggregate the needed information. In the following, these events are ordered according to their abstraction level:

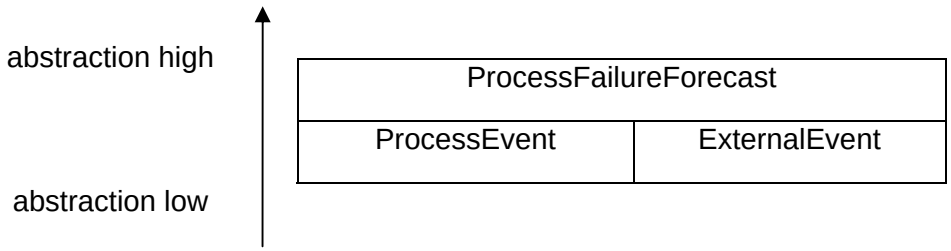


Figure 100: "External Process Influence Pattern" event abstraction layer

On the lowest abstraction level, there are the event types *ProcessEvent* and *ExternalEvent*. The considered moment of the information is always the moment when the event arrives at the system. This means e.g. that the information about the location within the process is relative to the time as well as the physical location of an item.

Events of the type *ProcessEvent* need to contain the following information:

- *timeStamp*: key identifying the order of the processed events,
- *itemId*: key that identifies a processed item,
- *processLoc*: location identifying the position in the business process and
- *areaCode*: identifier locating the item is physically in the process.

The type *ExternalEvents* must provide the following fields:

- *timeStamp*: key identifying the order of the processed events,
- *areaCode*: identifier locating the area of the external information and
- *probability*: factor that allows forecasting the process failure.

The type *ProcessFailureForecast* is the event which is generated by the correlation of the *ProcessEvent* and the *ExternalEvent*. It is further processed to arrive in a BAM view or an alerting system.

To retrieve the needed information, it is necessary to reveal the following fields:

- *timeStamp*: key identifying the exact time when the event is correlated,
- *itemId*: key that identifies the processed item,
- *processLoc*: location identifying the position in the business process now,
- *areaCode*: identifying where the item will be physically located next and
- *probability*: factor if the item passes in the next step through the area

9.5.1.7 Dynamics

Each process step generates several *ProcessEvents* on the one side while in the mean time there are *ExternalEvents* continuously coming in the system.

Once a *ProcessEvent* enters the system, the system matches the process information with the data of the *ExternalEvents* to generate the *ProcessFailureForecast* event.

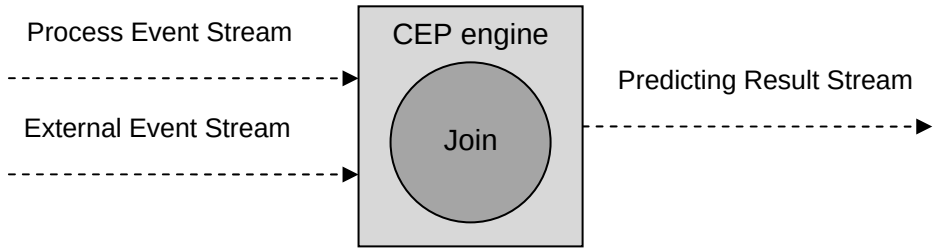


Figure 101: Dynamics of the "External Process Influence Pattern"

The *ProcessFailureForecast* event contains the information about how likely the process will fail in the next step of the execution. The forecast needs to be done based on metrics that are defined by KPI's.

9.5.1.8 Implementation

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

- *Collecting the fired events of the type ProcessEvent*: The *ProcessEvents* can be fired either directly out of the process or hard coded out of the services.

- *Collecting events of the type ExternalEvent*: The continuous collection of the *ExternalEvents* occurs independently of the business process instance. An example for an external event flow is TMC which radio stations send out as a service in addition to their traffic reports.

- *Correlating the events in a continuous query*: The correlation of the *ProcessEvent* and the *ExternalEvent* enriches the process information with valuable forecasting information.

- *Realize the complex event type ProcessFailureForecast*: The result of the correlation is the *ProcessFailureForecast* as a complex event type.

9.5.1.9 Example Resolved

The example shows the implementation in Oracle CQL code. To get to the point where the actual correlation is applied, several steps need to be accomplished.

- Collecting the fired events of the type *ProcessEvent* via the CEP input stream definition:

```
1 CREATE STREAM ProcessEvent
2 (
3    timeStamp BIGINT,
4    itemId BIGINT,
5    processLoc char(50),
6    areaCode INT
7 );
8
9 ALTER STREAM ProcessEvent ADD SOURCE PUSH;
```

Listing 44: ProcessEvent stream creation

- Collecting the events of the type *ExternalEvent* as stream input:

```
1 CREATE STREAM ExternalEvent
2 (
3    timeStamp BIGINT,
4    areaCode INT,
5    probability FLOAT
6 );
7
8 ALTER STREAM ExternalEvent ADD SOURCE PUSH;
```

Listing 45: ExternalEvent stream creation

- Correlating the events in a continuous query:

```
1 CREATE QUERY ProcessFailureForecast AS
2 SELECT
3    P.itemId, P.processLoc, P.areaCode, E.probability
4 FROM
5    ProcessEvent [PARTITION BY areaCode ROWS 1] AS P,
6    ExternalEvent [PARTITION BY areaCode ROWS $ctArea] AS E
7 WHERE P.areaCode != $ctArea
8 AND (P.areaCode + 1) = E.areaCode
```

Listing 46: Event correlation in continuous query

The query selects tuple values from both streams. The queried fields are *itemId*, *processLoc* and *areaCode* from the *ProcessEvent* type and the *probability* from the *ExternalEvent* type. The query is indirectly based on the *timeStamp* attribute which orders the incoming streams.

The *ExternalEvent* type is valid for several areas. To resolve the event stream containing information about multiple areas, incoming streams are partitioned by the area code. An item can be on a specific time just in one area. Because of this, the queried row is equal to 1. The *ExternalEvent* stream contains several areas. The variable *$ctArea* is a placeholder for the maximum number of areas in the logistic system.

The join happens within the WHERE clause of the statement and checks for the state of the next area by comparing the next area for the item (*P.areaCode + 1*) with the matching *areaCode* of the *ExternalEvent*.

- Realize the complex event type *ProcessFailureForecast*:

```
1 ALTER QUERY ProcessFailureForecast
2 ADD DESTINATION
3 "<EndPointReference>
4    <Address>
5       jms:jms/TopicConnectionFactory:
6       jms/publishingTopic</Address>
7 </EndPointReference>";
```

Listing 47: Realization of the ProcessFailureForecast

To realize the complex event *ProcessFailureForecast* it is necessary to publish the query result to an endpoint. In the example, the endpoint is represented as a JMS destination.

9.5.1.10 Variants

Variants on the “External Process Influence Pattern” depend on the incoming event streams.

Possible variants are e.g. comprehension of external weather information on the delivery process as wells as including the human factor of the driver.

9.5.1.11 Known Uses

The development of the “External Process Influence Pattern” happened during the development of the projects implementation for Deutsche Post and Oracle in this thesis. The implementation dealt with the topic of capturing process data and external data to provide the business a real-time overview of the actual state as well as predicting future business activities.

9.5.1.12 Consequences

The External Process Influence pattern provides benefits in the following areas:

It is possible to predict future process activities by correlating the actual process data with the information of the external process influences.

It enables the business to take appropriate measurements in advance on the predicted activities to avoid losses based on failed business processes.

10 Summary

Event processing enables businesses not only to monitor real-time data but also to introduce predictive business, which enables reacting to possible opportunities or errors before they actually impact the business process.

To reach this goal, there are several event processing products on the market. Within the project, a continuous solution has been developed by combining the goals of a SOA with the advantages of CEP. For this, we used the combination of Oracle BPEL, CEP and BAM with the Sopera ESB. That prototypically implementation allowed us to create an environment for event processing based on the latest technology to evaluate the advantages of SOA plus CEP.

In the example implementation, the Web services are developed with the open source SOA Framework and orchestrated with Oracle BPEL. The Sopera services are capable of sending process information directly to BAM and CEP. Sopera offers a service, called notification receiver, which is able to collect all events that are directly fired from the services as well as technical status information about the services. CEP provides a powerful event processing engine, engineered after the research project STREAM of Stanford University. It contains a flexible way of querying stream data by using a continuous query language (CQL) derived from SQL. CEP queries event streams and generates results used within BAM to be displayed within dashboards. The data is also used to create alerts for predictive business. BAM also offers its own less flexible event engine to collect event streams for displaying and alerting purposes.

Currently, it is able to detect possible errors within a business process by using the CEP technology and alert responsible persons by using e.g. BAM. That is just the first step. In the future, business processes have to react automatically to such errors. But compared to the technology used in today’s businesses, which just monitors activities that happened in the past, CEP accomplishes a great step in the right direction.

11 Acknowledgment

At this point the authors would like to thank every involved party for the excellent support.

First of all, thanks to Oracle and Sopera for providing all software components and the excellent support during the project.

Thanks go particularly to Rainer von Ammon who offered us, with his great business contacts all over the world, the chance to work at Oracle Headquarters in a project with the global players of SOA – Oracle and Deutsche Post.

Thanks go also to Dieter Gawlick through whom we got in touch with Oracle Headquarters. He also brought in many interesting research ideas in the context of event processing.

Explicit thanks go to Shailendra Mishra - Director of CEP at Oracle - and his team, who directed the development of the CEP patterns and supported us on the CEP side technically. Shailendra's good contacts within Oracle helped us to resolve all issues with Oracles products quickly.

Special thanks also go to Bernd Trops, who helped setting up the project when he was working at Oracle. During the project, he left Oracle and started working at Sopera, but was still a mentor for the authors.

Explicit thanks also go to Gerald Preissler and Dietmar Wolz, who always actively listened to all questions regarding the Sopera products. Both supported the authors technically as well as with great ideas regarding event processing during the weekly telephone conferences.

We want to thank Oracle Germany and Deutsche Post AG for sponsoring this project.

12 List of Literature

[ABER 06] AberdeenGroup: The Busines Activity and Monitoring Benchmark Report

August 2006

[ADBO 07] A. Adi, D. Botzer, O. Etzion: Semantic event model and its implication on situation detection

http://www.complexevents.com/media/articles/ECIS2000.pdf

Request: January 24th 2007

[ARAS 03] A. Arasu, B. Babock, S. Babu, J. Cieslewitz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom: STREAM: The Stanford Stream Data Manager

IEEE Data Engineering Bulletin

http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=2004-20&format=pdf&compression=&name=2004-20.pdf

2003, March

[ARAS 05] A. Arasu, S. Babu, J. Widom: The CQL Continous Query Language: Semantic Foundations and Query Execution

VLDB Journal

http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=2003-67&format=pdf&compression=&name=2003-67.pdf

2005, Stanford University

[BABO 02] B. Babock, S. Babu, M. Datar, R. Motwani, J. Widom: Models and Issues in Data Stream Systems

Invited Paper in Proc. Of PODS 2002

http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=2002-19&format=pdf&compression=&name=2002-19.pdf

2002 June

[BABU 01] S. Babu, J. Widom: Continous Queries over Data Streams

In SIGMOD Record

http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=2001-9&format=pdf&compression=&name=2001-9.pdf

September 2001, Stanford University

[BIEN 07] A. Bien: Java EE 5 Architekturen Java Patterns und Idiome,

ISBN: 9783939084242

2007, entwickler.press

[BRAY 06] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau: Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation 16th August 2006, edited in place 29 September 2006

Copyright 2006 W3C (MIT, ERCIM, Keio)

http://www.w3.org/TR/REC-xml/

Request: January 17th 2008

[BUSC 96] F. Bushmann, R Meunier, H. Rohnert, P. Sommerlad, M. Stal: Pattern-Oriented Software Architecture A System of Patterns,

ISBN: 0471958697

1996, John Wiley & Sons Ltd.

[CHAP 08] M. Chapple: Query Definition

http://databases.about.com/cs/administration/g/query.htm

Request: February 3rd, 2008

[CLAR 99] J. Clark: XSL Tranformations (XSLT), W3C Recommendation 16 Noveber 1999,

Copyright 1999 W3C (MIT, INRIA, Keio)

http://www.w3.org/TR/xslt

Request: January 18th 2008

[COME 04] D. E. Comer, R. E. Droms: Computer Networks and Internets

ISBN: 013143351

2004, Prentice Hall

[CORR 02] J. Correia, N. Schroder: BAM: A Composite Market View, Research Brief

Gartner Dataquest, Marketplace code: SOFT-WW-DP-0067

April 1st, 2002

[DEKR 76] F. DeRemer and H. H. Kron, "Programming-in-the-Large versus Programming-in-the-Small," IEEE Transactions on Software Engineering, Vol. SE-2, No. 2, p. 80-86

June 1976

[DOST 05] W. Dostal, M. Jeckle, I. Melzer, B. Zengler: Service-orientierte Architekturen mit Web Services

ISBN 3-8274 – 1457 - 1

2005, Spektrum Akademischer Verlag

[GAMM 95] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides: Design Patterns Elements of Reusable Object-Oriented Software,

ISBN: 0201633612

2000, Addison-Wesley

[GERR 06] Gerrard Consulting: The Terms

http://www.gerrardconsulting.com/bs7925_1/terms.html

Requests: December 25th 2007

[GREV 04] R. Grevink: Von Legacy-Anwendungen zur Service-Orientierten Architektur, WRQ Software GmbH

http://itverlag.uspmarcom.de/Integration/documents/5WRQ.pdf

February 19th 2004

Request: January 11th 2008

[HERR 07] W. Herrmann : Computer Woche (SOA Vision): SOA 2.0 zu schnell für die Anwender?

http://www.computerwoche.de/zone/soavision/578012/

Request: February 8th 2008

[IBMC 08] IBM FileNet Business Activity Monitor (BAM)

ftp://ftp.software.ibm.com/software/data/ECM/Bro/IBM_ECM_BAM_FED_DS.pdf

Request: January 21st 2008

[IBRA 00] M. Ibrahim, J. Küng, N. Revell: Database and Expert Systems Applications, 11th International Conference, DEXA 2000 London, UK

ISSN: 03029743

2000, Springer

[JECK 04] M. Jeckle, C. Rupp: UML 2 glasklar

ISBN 3-446-22575-7

2004, Carl Hanser Verlag

[KENT 82] W. Kent: A Simple Guide to Five Normal Forms in Relational Database Theory

http://www.bkent.net/Doc/simple5.htm

Request: January 14th 2008

[KOCH 05] H. Kochar: Business Activity Monitoring and Business Intelligence

http://www.ebizq.net/topics/bam/features/6596.html

Request: January 17th 2008

[LiTa 08] Ling Liu, M. Tamer: Encyclopedia of Database Systems

http://refworks.springer.com/mrw/index.php?id=1217

Will be Updated in 2008

[LUCK 07] D. Luckham: Complex Event Processing – What's the Difference Between ESP and CEP,

http://complexevents.com/?p=103

Request: December 17th 2007

[MATA 00] R. A. Mata-Toledo, P. K. Cushman: Fundamentals of Relational Databases

ISBN: 0-07-136188-X

2000, The McGraw-Hill Companies Inc.

[MATJ 06] Matjaz B. Juric, B. Mathew, P. Sarang: Business Process Execution Language for Web Services (Second Edition)

ISBN: 1-904811-81-7

January 2006, Packt Publishing

[MOTW 03] R. Motwani, J. Widom, A. Arasu, B. Babock, S. Babu, M. Datar, G. Manku, C. Olston, J. Roesenstein, R. Varma: Query Processing, Resouce Management, and Approximation in a Data Stream Management System

In Proc. CIDR 2003

http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=2002-41&format=pdf&compression=&name=2002-41.pdf

January, 2003

[NATI 03] Y. V. Natis: Service-Oriented Architecture Scenario

Gartner Research Note AV-19-6751

2003

[OASI 04] OASIS: UDDI Version 3.0.2, UDDI Spec Technical Committee Draft

http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

October 19th 2004

[OASI 06] OASIS: Reference Model for Service Oriented Architecture 1.0, Committee Specification

http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

August 2nd 2006

[OASI 07a] OASIS: Web Service Business Process Execution Language 2.0,

http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

April 11th 2007

[OASI 07b] F. Ryan: OASIS: WS-BPEL 2.0, Web Service Business Process Exectuion Language, Technical Instruction Presentation

http://www.oasis-open.org/committees/download.php/23068/WS-BPEL%20Technical%20Overview%20for%20Developers%20and%20Architects%20-%20Part%201%20(Frank%20Ryan).pdf

Request: January 5th 2008

[ORAC 01] Oracle: Oracle9i Lite Developer's Guide for AQ Lite Release 5.0, Part Number A90251-01

http://download.oracle.com/docs/html/A90251_01/aqintr.htm

Request: December 16th 2007

[ORAC 05a] Oracle, H. Kochar, K. Clugage: Oracle Business Activity Monitoring, An Oracle White Paper, June 2005

http://www.oracle.com/technology/products/integration/bam/pdf/bam_whitepaper.pdf

Request: January 25th 2008

[ORAC 05b] Oracle: BAM Enterprise Link, Getting Started Guide 10g Release 2 (10.1.2)

Part No. B19357-01

September 2005

[ORAC 05c] Oracle: BAM Enterprise Link Design Studio User's Guide 10g Release 2 (10.1.2)

Part No. B19355-01

September 2005

[ORAC 06] Oracle: Oracle Business Activity Monitoring Architects User's Guide 10g (10.1.3.1.0)

Part No. B28992-01

October 2006

[ORAC 06a] Oracle: SOA Suite Developers Guide 10.g (10.1.3.1.0)

Part No. B19355-01

September 2006

[ORAC 06b] Oracle: SOA Suite – Oracle BPEL Process Manager 06, Whitepaper

http://bpel.us.oracle.com/docs/Oracle-SOA-Suite-Process-Orchestration.pdf

July 2006

[ORAC 06c] Oracle: Business Activity Monitoring Active Studio User's Guide 10.g

Part No. B28990-01

http://st-doc.us.oracle.com/review/IPDOC/bam/101310/b28990.pdf

October 2006

[ORAC 06d] Oracle: Oracle SOA Suite – Oracle Business Activity Monitoring,

White Paper,

July 2006

[ORAC 07] Oracle: Tech Note: Oracle BAM 10.1.3 configuration for BPEL 10.1.2 using JMS sensors

2007

[ORAC 08] Oracle: Oracle Integration Business Activity Monitoring

http://www.oracle.com/technology/products/integration/bam/pdf/oracle-bam-datasheet.pdf

Request: January 21st 2008

[PELT 03] C. Peltz: Web Services Orchestration and Choreography

Computer, vol. 36, no. 10, pp. 46-52,

October 2003

[REGU 07] B. Regunath: Myth of ESB, Picture of an ESB

http://regumindtrail.wordpress.com/2007/09/14/myth-of-the-esb/

Request: September 14th 2007

[REGU 06] B. Regunath: Software Development

http://regumindtrail.wordpress.com/2006/12/21/softwaredev/

Request: September 9th 2007

[RIED 08] S. Ried: ESB Open Source Challenge, CITT Experts meeting, Dr. Stefan Ried **(**Forrester Research**)**

http://www.stefan-ried.de/wp-content/uploads/2008/01/opensourceesbs-citt-2008-01-14.pdf

Request: January 14th 2008

[RUEB 07] U. Rübesamen: Web Services im Takt, IRDIX AG

http://www.ordix.de/ORDIXNews/2_2005/bpel.html

Request: December 12th 2007

[SAP 07a] SAP: Web Services Human Task, Version 1.0, Active Endpoints Inc., Adobe Systems Inc., BEA Systems Inc., International Business Machines Corporation, Oracle Inc., and SAP AG

https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/a0c9ce4c-ee02-2a10-4b96-cb205464aa02

June 2007

[SAP 07b] SAP: WS-BPEL Extension for People Version 1.0, Active Endpoints Inc., Adobe Systems Inc., BEA Systems Inc., International Business Machines Corporation, Oracle Inc., and SAP AG

https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/30c6f5b5-ef02-2a10-c8b5-cc1147f4d58c

June 2007

[SILB 07] C. Silberbauer, R. v. Ammon, H.-M. Brandl, D. Guschakowski, T. Greiner, R. Kintscher et.al.: Proposal for a Reference Model for Event Patterns Taking the Example „Loss Pattern“

http://www.citt-online.com/downloads/ReferenceModelsEventPatterns_with_Appendix_v3.pdf

June 2007

[SOPE 07a] SOPERA GmbH: ToolSuite User's Guide, Copyright © Deutsche Post AG

August 2007

[SOPE 07b] Sopera GmbH: Policies Reference Guide, Copyright © Deutsche Post AG

July 2007

[SOPE 08] SOPERA GmbH Website

http://sopera.de/119/

Request: January 12th 2008

[SUNM 02] Sun Microsystems, M. Hapner, R. Burridge, R. Sharma, J. Fialli, K. Stout: Java Message Service, Version 1.1,

http://java.sun.com/products/jms/docs.html

April 12th 2002

[SUNM 03] Sun Microsystems: Java™ 2 Platform Std. Ed. v1.4.2

http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html

2003

[SYST 08] Systar: Business Bridge

http://www.systar.com/products/businessbridge.asp

Request: January 20th 2008

[TECH 08] TechEncyclopedia: drill down

http://www.techweb.com/encyclopedia/defineterm.jhtml?term=drill+down

Request: Feb 08th 2008

[UEBE 08] S. Ueberhorst: Computerwoche, Report: Absage an SOA plus EDA

http://www.computerwoche.de/produkte_technik/software/1854954/index4.html

Request: February 8th 2008

[W3C 01] W3C: Web Services Description Language (WSDL) 1.1

http://www.w3.org/TR/wsdl

March 15th 2001

[W3C 04] W3C: W3C Working Group Note, Web Service Architecture

http://www.w3.org/TR/ws-arch/wsa.pdf

February 11th 2004

[W3C 07] W3C: W3C Recommendation: SOAP Version 1.2 Part 1 Messaging Framework

http://www.w3.org/TR/soap12-part1/

April 27th 2007

[WIKI 07] Wikipedia Encyclopedia: Definition Test Case

http://en.wikipedia.org/wiki/Test_case

Request: December 16th 2007

[WIKI 08] Wikipedia Encyclopedia: Definition Loose coupling

http://en.wikipedia.org/wiki/Loose_coupling

Request: January 15th 2008

13 List of Figures

14 List of Tables

15 List of Listings

16 List of Abbreviations

ADC:	Active Data Cache
API:	Application Programming Interface
AS:	Application Server
BAM:	Business Activity Monitoring
BPEL:	Business Process Execution Language
BPM:	Business Process Monitoring
CEP:	Complex Event Processing
CORBA:	Common Object Request Broker Architecture
CQL:	Continuous Query Language
DB:	Database
DDL:	Data Definition Language
DSB:	Distributed Service Bus
EDA:	Event Driven Architecture
EPC:	Event-driven Process Chain
EPL:	Event Programming Language
ESB:	Enterprise Service Bus
ESP:	Event Stream Processing
HTTP	Hypertext Transfer Protocol
HTTPs:	Hypertext Transfer Protocol over Secure Socket Layer
IDE:	Integrated Development Environment
IT:	Information Technology
JEE/J2EE:	Java Enterprise Edition
JMS:	Java Messaging Service
JSE/J2SE:	Java Standard Edition
KPI:	Key Performance Indicator
MOM:	Message Oriented Middleware
NR:	Notification Receiver

OASIS:	Organization for the Advancement of Structured Information Standards
OC4J:	Oracle Container for Java
OOP:	Object Oriented Programming
PAPI:	Participant Programming Application Interface
POM:	Project Object Model
QoS:	Quality of Service
RFID:	Radio-Frequency Identification
TMC:	Traffic Message Channel
SBB:	Service Back Bone
SMTP:	Simple Mail Transfer Protocol
SOA:	Service Oriented Architecture
SOAP:	(Former) Simple Object Access Protocol
SOPERA:	Open Source SOA Framework
UDDI:	Universal Description, Discovery and Integration
URI:	Uniform Resource Identifier
W3C:	World Wide Web Consortium
WSDL:	Web Service Description Language
WSFL:	Web Services Flow Language
XML:	Extended Markup Language
XPath:	XML Path Language
XSD:	XML Schema Definition
XSLT:	Extensible Stylesheet Language Transformation