



AGH

University of Science and Technology

Faculty of Electrical Engineering, Automatics,
Computer Science and Electronics

Department of Computer Science

Kraków, Poland

Master thesis

ESB application for effective synchronization of large volume measurements data

Przemysław Wyszkowski

Major: Computer science

Speciality: Computer networks and distributed systems

Thesis supervisor:

Register No.: 203792

Dominik Radziszowski, Ph.D. Eng.

Kraków, Genève, Dresden 2011

Oświadczenie autora

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

.....
(Podpis autora)

Contents

Streszczenie	4
Zusammenfassung	5
Abstract	6
Acknowledgements	7
1. Introduction	8
1.1. The Large Hadron Collider	8
1.2. Scope	10
1.2.1. Thesis aim	10
1.2.2. Thesis structure	11
2. Background information	12
2.1. Massive data processing in High Energy Physics experiments	12
2.1.1. Measurement data life-cycle in HEP	12
2.1.2. The TOTEM experiment	15
2.2. Used technologies overview	16
2.2.1. SOA - ESB - OSGi	16
2.2.2. MOM - JMS - ActiveMQ	21
2.2.3. EIP - Camel	24
2.2.4. IoC - DI - Spring	25
2.2.5. ORM - JPA - Hibernate	27
2.3. Clarification summary	28
2.4. Selected frameworks	29
2.4.1. Apache ServiceMix 4.x	30
2.4.2. FUSE ESB 4	31
2.5. Related works	31
2.6. Chapter summary	32
3. System model	33
3.1. Current state of art analysis	33
3.1.1. Current processing model	33
3.1.2. Existing state evaluation	35
3.2. The TOTEM Offline Database Project	35
3.2.1. Data sources identification	37

3.2.2.	Examples of the data routes	40
3.3.	Problem statement	43
3.3.1.	Requirements	43
3.3.2.	Use-cases definition	44
3.3.3.	Solution proposal - DBPop	45
3.4.	Architecture	46
3.4.1.	Basic view	46
3.4.2.	Technology-agnostic approach	46
3.4.3.	Architecture using ESB approach	48
3.4.4.	Alternative ways to achieve solution	50
3.5.	Chapter summary	50
4.	Implementation issues	52
4.1.	Architecture using ServiceMix 4.x	52
4.1.1.	Basic concepts	52
4.1.2.	Architecture	53
4.1.3.	Working mechanism of DBPop	54
4.2.	DBPop components	55
4.2.1.	LHC API Poller	56
4.2.2.	Database Binding	58
4.2.3.	Processing Logic	60
4.2.4.	Other data accessing modules	62
4.2.5.	JMS queues	63
4.2.6.	Camel routes	66
4.3.	TOTEM Offline Database	67
4.3.1.	Schema	67
4.3.2.	Tables description	68
4.3.3.	Mappings	72
4.4.	Development environment	72
4.4.1.	Using Maven	72
4.4.2.	Working with IDE	73
4.4.3.	Working remotely	73
4.5.	User interface - DBPop CLI	73
4.5.1.	Implementation	73
4.5.2.	Functionality	74
4.5.3.	Usage	74
4.6.	Chapter summary	75
5.	Tests	77
5.1.	Tests environment	77
5.1.1.	Hardware	78
5.1.2.	Software	80
5.1.3.	Tests scenario	82
5.2.	Efficiency	83
5.2.1.	Direct methods invocation (scenario 1.a)	84
5.2.2.	External message broker (scenario 1.b)	85
5.2.3.	Internal memory message broker (scenario 1.c)	86
5.3.	Scalability	86
5.3.1.	Distributed DBPop (scenario 2.a)	87

5.3.2. Application clusterization (scenarios 2.b, 2.c)	88
5.4. Tests results	89
5.5. Other properties evaluation	91
5.5.1. Fault-tolerancy	91
5.5.2. Extendability	91
5.6. Chapter summary	92
Summary	93
Glossary	95
Glossary	95
Acronyms	97
Acronyms	97
List of Figures	101
List of Tables	103

Streszczenie

Wykorzystanie szyny ESB do efektywnej synchronizacji danych pomiarowych dużej objętości

Eksperyment TOTEM w CERN bada całkowity przekrój czynny, rozpraszanie elastyczne i dysocjację dyfrakcyjną protonów ulegających kolizjom w Wielkim Zderzaczu Hadronów. Aby badanie tych zjawisk było możliwe, konieczne jest przetwarzanie ogromnej ilości danych pochodzących z różnych źródeł: detektorów TOTEM'u, detektorów CMS'u, aparatury pomiarowej zainstalowanej w tunelu Wielkiego Zderzacza Hadronów oraz wielu innych, pomniejszych systemów zewnętrznych. Do opracowywania wyników potrzebne są również dane pośrednie, opracowywane na podstawie surowych informacji pochodzących z detektorów.

Dla wydajnej i wygodnej pracy naukowców niezbędnym jest zapewnienie centralnego punktu, w którym wszystkie potrzebne dane (tak w stanie surowym jak i przetworzonym) będą przechowywane. Niniejsza praca ma na celu prezentację wykorzystania koncepcji szyny integracyjnej Enterprise Service Bus w budowie infrastruktury programowej do efektywnego przesyłu danych pomiarowych dużej objętości. Zaprezentowane zostały technologie i mechanizmy realizujące koncepcję szyny integracyjnej, model systemu przesyłu danych w oparciu o wykorzystanie kolejek komunikatów, jego implementacja na potrzeby eksperymentu TOTEM oraz badania wydajności i skalowalności rozwiązania.

Słowa kluczowe

przesył danych, Service-Oriented Architecture, OSGi, Enterprise Service Bus, Message-Oriented Middleware, Enterprise Integration Patterns, CERN, TOTEM

Zusammenfassung

Enterprise Service Bus-Anwendung für die effektive Synchronisation der umfangreichen Messdaten

Das TOTEM Experiment in CERN untersucht den Wirkungsquerschnitt, die elastische Streuung und diffraktive Protonen-Dissoziation, die in dem Großen Hadronen-Speicherring zusammenstoßen. Um solche Forschung zu ermöglichen muss man größte Datenmengen verarbeiten, die aus verschiedene Quellen kommen: die TOTEM-Detektoren, die CMS-Detektoren und Messeinrichtungen, die in dem Großen Hadronen-Speicherring Tunnel lokalisiert sind, sowie von mehreren anderen äußeren Systemen stammen. Um die Ergebnisse zu bearbeiten braucht man auch indirekte Daten, die von Primär-Informationsdetektoren ausgehen.

Um effektive und bequeme Arbeitsabläufe der Wissenschaftler zu gewährleisten, benötigt man einen zentralen Speicherort, an der alle notwendigen Daten (primäre und bearbeitete) gespeichert werden. Diese Arbeit betrachtet die Nutzung des Integrationskonzeptes Enterprise Service Bus für eine Softwareinfrastruktur zum Transfer umfangreicher Messdaten. Es wird die Technologie und Funktionsweise des Integrations-Konzeptes sowie des Transfersystemmodells präsentiert, welches auf Message-Oriented Middleware basiert. Dieses Modell wird anhand des TOTEM-Experiments implementiert und dessen Effektivität und Skalierbarkeit evaluiert.

Schlüsselwörter

data transfer, Service-Oriented Architecture, OSGi, Enterprise Service Bus, Message-Oriented Middleware, Enterprise Integration Patterns, CERN, TOTEM

Abstract

ESB application for effective synchronization of large volume measurements data

The TOTEM experiment at CERN aims at measurement of total cross section, elastic scattering and diffractive processes of colliding protons in the Large Hadron Collider. In order for the research to be possible, it is necessary to process huge amounts of data coming from variety of sources: TOTEM detectors, CMS detectors, measurement devices around the Large Hadron Collider tunnel and many other external systems. Preparing final results involves also calculating plenty of intermediate figures, which also need to be stored.

In order for the work of the scientist to be effective and convenient it is crucial to provide central point for the data storage, where all raw and intermediate figures will be stored. This thesis aims at presenting the usage of Enterprise Service Bus concept in building software infrastructure for transferring large volume of measurements data. Topics discussed here include technologies and mechanisms realizing the concept of integration bus, model of data transferring system based on Message-Oriented Middleware, system implementation for the TOTEM experiment as well as efficiency and scalability tests of the solution.

Keywords

data transfer, Service-Oriented Architecture, OSGi, Enterprise Service Bus, Message-Oriented Middleware, Enterprise Integration Patterns, CERN, TOTEM

Acknowledgements

Here is the proper place to acknowledge all those people, who, with their involvement and support are contributors to this work.

First of all, I would like to thank my thesis supervisor - Dominik Radziszowski for all the knowledge, patience and good advices given throughout the time of this thesis creation. Many thanks to Leszek Grzanka for this thesis reading and support over the recent year. I would like to thank Hubert Niewiadomski, Valentina Avati, Adrian Fiergolski and friends from CERN for enormous help while my stay in Genève. There were also plenty of people at CERN who gave me a lot of useful hints and advices. Thank you for your help and collaboration.

I would like to send many thanks to Matthias Wauer and Professor Alexander Schill for fantastic collaboration while my stay in Dresden. I appreciate the works of Rafał Czarny, who has designed the L^AT_EXstyle used in this manuscript, thanks for all the tips you gave me. Let me also thank my friends who were accompanying me in my efforts, especially Maciej Matoszko and Tomasz Lewicki for good words and all the great times we had during this thesis creation.

I would like to express my deep gratitude to my family, my parents Krystyna and Stanisław, my brother Grzegorz and my grandparents Janina and Władysław, for the spiritual support of my effort. Finally, completion of this thesis would not have been possible without the support of the most wonderful woman in the world - Edyta Krzyżak. Thank you for the energy and strength you gave me, I appreciate your patience and understanding.

Kraków, September 2011
Przemysław Wyszkowski

Chapter 1

Introduction

1.1. The Large Hadron Collider

Our understanding of the Universe is incomplete [1]. Although we live in 21st century there are still basic questions regarding the world around us which lack answers. The motivation to construct the *Large Hadron Collider (LHC)* at *European Organization for Nuclear Research (CERN)* comes from the fundamental questions in Particle Physics. These questions [2] include:

What is mass?

What is the origin of mass? Why do tiny particles weigh the amount they do? Why do some particles have no mass at all?

What is 96% of the universe made of?

Material world we see every day is only 4% of the whole universe. Dark matter and dark energy are believed to make up the rest of the universe still to be discovered

What was matter like within the first second of the Universe's life?

It is unknown, how the matter was formed just after the Big Bang. Is it possible to reconstruct the contemporary conditions?

Why is there no more antimatter?

At the birth of the Universe, equal amounts of matter and antimatter should have been produced in the Big Bang. Why does Nature appear to have this bias for matter over antimatter?

The *Standard Model* provides description of how particles interact with each other but the existence of some particle supposed by the model was not yet confirmed. The most famous particle still to be discovered is the *Higgs Boson* (also called “*The God particle*”), which is supposed to give mass to other elementary particles. Final criteria of every physics rule verification is always an experiment. In order for those questions to be answered high energy and a high *luminosity*¹ collider needed to be built.

The *LHC* is at present the biggest particle accelerator in the world [3]. It is installed in a circular tunnel with 26.7 km circumference at the depth of 50 to 175 m

1. *luminosity* - is the particles collision rate in the accelerator; more in the glossary

underground. LHC is a place of the greatest extremes on earth: largest accelerator, highest energy of particles, lowest operating temperature, highest pressure of helium to cool the machine and of course the highest data production pace. Particles, prior to being injected to the main *LHC* pipes, are first prepared in a chain of smaller accelerators which increase their energy levels.

Protons, or heavy ions start their run in linear accelerator Linac2 which accelerates them to the energy of 50 MeV. Beam feeds the *Proton Synchrotron Booster (PSB)* and with the energy level of 1.4 GeV enters the *Proton Synchrotron (PS)*, where it is accelerated to 26 GeV. At this stage particles are already moving with the speed around 99,9% of the velocity of light. It is here, where the point of transition is reached, when the energy added to the protons by the pulsating electric field cannot be translated to the increase of the speed. Instead, the added energy manifests itself as increasing mass of the protons and at current stage they are 25 times heavier than at rest. *Super Proton Synchrotron (SPS)* is used to boost the energy of the particles from 26 GeV up to 450 GeV. Finally, particles are distributed into LHC pipes both in a clockwise and anticlockwise direction, where the energy of 3.5 TeV is reached by each beam resulting in 7 TeV when colliding. Nominal energy of the single beam, still to be achieved, will be 7 TeV resulting in 14 TeV collisions energy. Works related to proper technical upgrade in order to meet these requirements are foreseen in the nearest future.

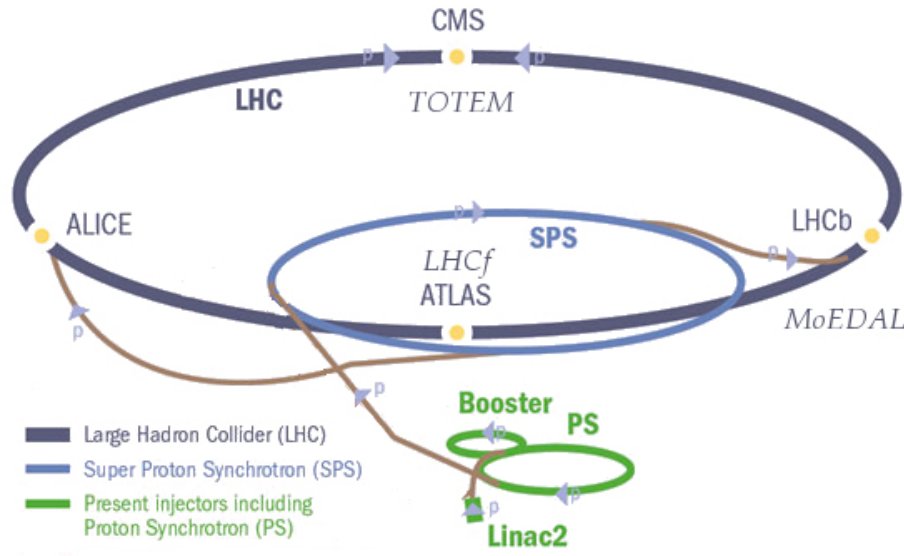


Figure 1.1. LHC filling accelerator chain and experiments' location. Figure based on [4]

The beam will circulate inside the pipes for several hours producing collisions in four interaction points, where experiments' detectors are located. A simplified schema of accelerator chain and experiments' locations are presented in Figure 1.1. There are seven major particle physics experiments located around the *LHC* accelerator [5]:

1. **ALICE (A Large Ion Collider Experiment)**

To study the physics of strongly interacting matter at extreme energy densities, where the formation of a new phase of matter, the quark-gluon plasma, is expected

2. **ATLAS (A large Toroidal LHC ApparatuS)**
The ATLAS detector will search for new discoveries (such as the Higgs) in the head-on collisions of protons
3. **CMS (Compact Muon Solenoid)**
The CMS detector will explore the fundamental nature of matter and the basic forces that shape our universe.
4. **LHCb (Large Hadron Collider beauty)**
It undertakes precision studies of the decays of particles that contain heavy flavours of quarks (charm and beauty)
5. **LHCf (Large Hadron Collider forward)**
Measurement of the energy and numbers of neutral pions produced by the collider
6. **TOTEM (TOTAl Elastic and diffractive cross section Measurement)**
It is dedicated to the measurement of total cross section, elastic scattering and diffractive processes at the *LHC*
7. **MoEDAL (Monopole and Exotics Detector At the LHC)**
Will search for the massive stable or pseudo-stable particles, such as magnetic monopoles or dyons, produced at the *LHC*

1.2. Scope

Unlike in other fields of physics research, *High Energy Physics* (*HEP*) involves work of many computer scientists as an essential part of the experiment. Role of computer scientists in the process is to ease the effort of getting right data and provide resources and methods for the data to be processed. Plenty of tools need to be developed in order for the aim to be achieved. Author believes that the implementation work behind this thesis will help to speed up the research performed in the *TOTAl Elastic Scattering and Diffraction Measurement* (*TOTEM*) Experiment at *CERN*.

1.2.1. Thesis aim

This thesis describes the newest technologies and architectural concepts related to the topic of applications integration. The central concept of this thesis is an *Enterprise Service Bus* (*ESB*), which is used not only to integrate various systems, but also to transfer large amounts of data. Author is trying to prove the following:

- appliance of *ESB* for easy integration of heterogenous systems
- ease of extending and efficient development process
- appliance of component, loosely coupled architecture in building robust, scalable and reliable data transfer system
- test the behaviour of asynchronous *Message-Oriented Middleware* (*MOM*) sub-systems in data delivery

This thesis aim can be formulated as follows:

It is possible to use the ESB facilities to construct effective and scalable integration solution for synchronizing large volumes of measurements data

Effectiveness does not have to stand for velocity of data transfer only. Effectiveness will be considered in several aspects, including:

- speed
- scalability
- applying load balancing policies
- fault tolerancy
- extensibility ease

Chapter 5 of this thesis reveals how the chosen *ESB* implementation deals with these statements. It is also shown how the newest trends of component technologies find appliance in the field of *HEP* software support. Implementation of the *TOTEM DataBase POPulation System (DBPop)* is used in the *TOTEM* experiment at *CERN* as the data transferring system for measurements data. The text of this thesis might be the starting point for a computer scientist to get involved into the *TOTEM* data processing model. Proposed architectural and technological approach will be explained in the further chapters of this thesis.

1.2.2. Thesis structure

The structure of this thesis is as follows. Chapter 2 gives an overview of the environment in which the built system is being deployed. At first, the methods and organization of data processing in *TOTEM* experiment is presented. Next, technologies used for solution implementation are given an overview. There are a few projects addressing similar goals which are discussed in last section of Chapter 2. Chapter 3 is dedicated to the system model description. Requirements for the system as well as the existing state analysis are explained. Then, architectural concepts and technology application are discussed in the system design. In Chapter 4 the main design and implementation issues are analyzed. Description leverages practical ways of realizing the *ESB* concepts. Chapter 5 evaluates the system behaviour with reports from performed tests. Each of the succeeding chapters is followed by a conclusion summarizing its content. Last chapter is giving summary of the thesis.

Background information

This chapter's aim is to introduce the reader into concepts, vocabulary and approaches used in further chapters. The chapter is divided into three parts. First section discusses data gathering and processing subsystems in the *High Energy Physics* (*HEP*) experiments like the *TOTEM* experiment. It introduces basic terms and concepts crucial to analyze the environmental conditions for developed solution. Second section focuses on presenting the most important concepts and technologies that are fundamental for implementation of the *TOTEM DBPop* system. At the end, related projects dealing with data transfer and systems integration are given an overview.

2.1. Massive data processing in High Energy Physics experiments

HEP experiments are the largest producers of data that have ever been encountered in the history of human beings [6]. *CERN* is not an exception, it possesses the largest accelerator ever built and with the highest *luminosity* it delivers the largest data stream. In order to understand how it is generated, an introduction to the data acquisition process is required.

2.1.1. Measurement data life-cycle in HEP

For a computer scientist it is crucial to understand what are the phases in the data flow within the experiments of *High Energy Physics* (*HEP*) :

- How is a single measurement value generated?
- How is it represented and transmitted?
- Where is it stored?
- Where and how is it processed?
- How is it accessed?

Division of HEP systems

The basic division of the data processing procedure in *HEP* splits systems into two main groups:

- **Online processing**

Involves purely hardware-based *Data acquisition (DAQ)* processes like *Trigger*¹ and data transferring. Transition from raw data to meaningful physics values is performed by complex pattern recognition or clustering algorithms. These systems demand hard real-time systems which are implemented usually in *Application Specific Integrated Circuit (ASIC)* and *Field Programmable Gate Array (FPGA)* technologies.

- **Offline processing**

Processing is based on static data acquired in previous phase. Software processes (sometimes also hardware accelerated) for event reconstruction, data analysis, modelling and simulations are used to generate readable results. Presenting, concluding and evaluating experiment results is possible. No need to work in real-time.

A more detailed view on these systems is performed in the following subsections.

Online systems

Collisions in the accelerator tunnel are registered by detectors. These are composed of dedicated electronical devices capable to capture high rate collision events. Detectors plates generate raw electrical impulses representing events, which took place while collision. An example of the ATLAS detector will be used to present the *DAQ* process in more details[7].

The detector generates unmanageably large amounts of raw data, about 25 MB per event (raw, zero suppression reduces this to 1.6 MB) times 23 events per beam crossing, times 40 million beam crossings per second in the center of the detector, for a total of 23 TB/s of raw data. Because of huge data throughput it would be impossible to store all of the measurements coming from the detectors. Actually in most cases it is not needed - majority of it represents values which belong to the background of the process that are useless for the research purpose. The process called *Trigger* is dedicated to filter out the considered useless signals in order to reduce the data stream to a manageable rate[8].

The *Trigger* system uses simple information to identify, in real time, the most interesting events to retain for detailed analysis. There are three trigger levels, the first based in electronics on the detector and the other two primarily run on a large computer cluster near the detector. After the first-level trigger, about 100,000 events per second have been selected. After the third-level trigger, a few hundred events remain to be stored for further analysis. This amount of data will require over 100 MB of disk space per second - at least a petabyte each year.

Filtered data is immediately stored in the computer centre at *CERN*, so called Tier-0 which is also distributed among the data centers in Tier-1. This process assures that the data is backed-up and none of the valueable measurements gets lost. The data stored in Tier-1 Centers is redistributed on demand to about 130 computing centers around the globe for end-user access. Tier architecture of data processing at the *LHC* is presented in Figure 2.1 Whole process involves great amount of computing power, storage and network infrastructure, which is very often dedicated only for this purpose.

1. *Trigger* - is the process of filtering non-essential data; more in the glossary

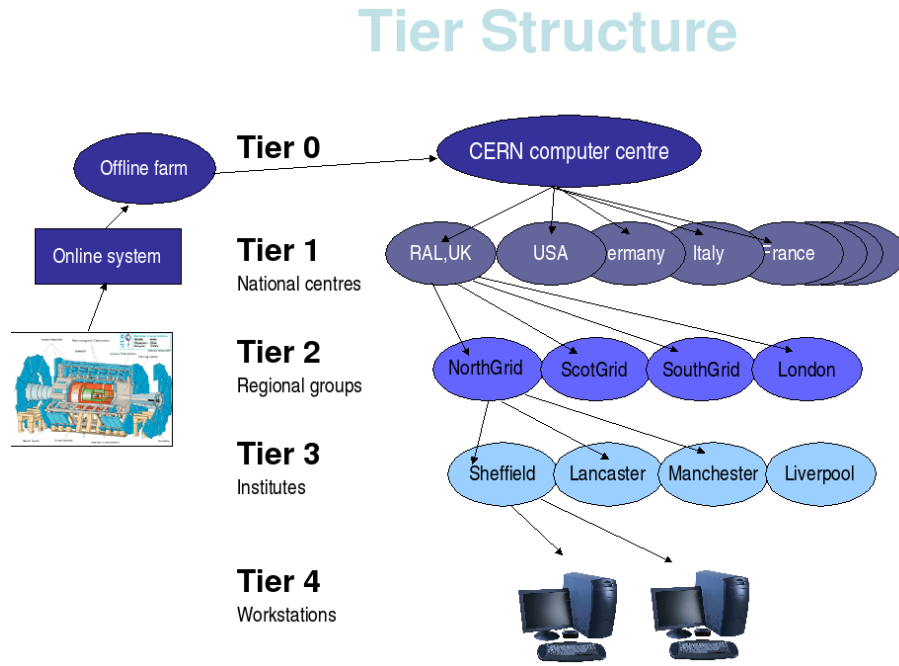


Figure 2.1. Tier architecture in data processing at the LHC. Figure from [9].

Offline processing

After the data appears in the data-stores it is exposed to the usage of scientists performing offline part of the life-cycle. Offline processing involves activities like, among others:

- event reconstruction
- pattern recognition
- simulations
- 3d graphics
- data mining

In order for the scientists to perform their tasks it is necessary for the data to be accessible. Researchers use the computing *Grid*² to submit their jobs as well as other means of computing and storing power to work and get the results. *CERN* uses Oracle facilities for the data management. It provides dedicated, centralized services which relieve all of the LHC experiments of maintaining separate infrastructure for databases. Huge amounts of measurements data are stored in a representant of *Hierarchical Storage Management (HSM)* concept - *CERN Advanced STORage manager (CASTOR)*. To ease the data distribution among all the scientists at *CERN* an *Andrew File System (AFS)* network file system is used for the data exchange.

2. *Grid* - is a genre of distributed computing infrastructure; more in the glossary

2.1.2. The TOTEM experiment

The *TOTAL Elastic Scattering and Diffraction Measurement (TOTEM)* [10] [11] experiment will measure the total proton-proton cross section and will study the elastic scattering and diffractive dissociation at the *LHC*. More specifically, *TOTEM* focuses on [12]:

- the total cross-section with an absolute error of 1 mb^3 by using the luminosity independent method. This requires the simultaneous measurement of the elastic proton-proton scattering down to the four-momentum transfer of $-t \approx 10^3 \text{ GeV}^2$ and of the inelastic proton-proton interaction rate with an adequate acceptance in the forward region;
- elastic proton scattering over a wide range in momentum transfer up to $-t \approx 10 \text{ GeV}^2$;
- diffractive dissociation, including single, double and central diffraction topologies using the forward inelastic detectors in combination with one of the large *LHC* detectors.

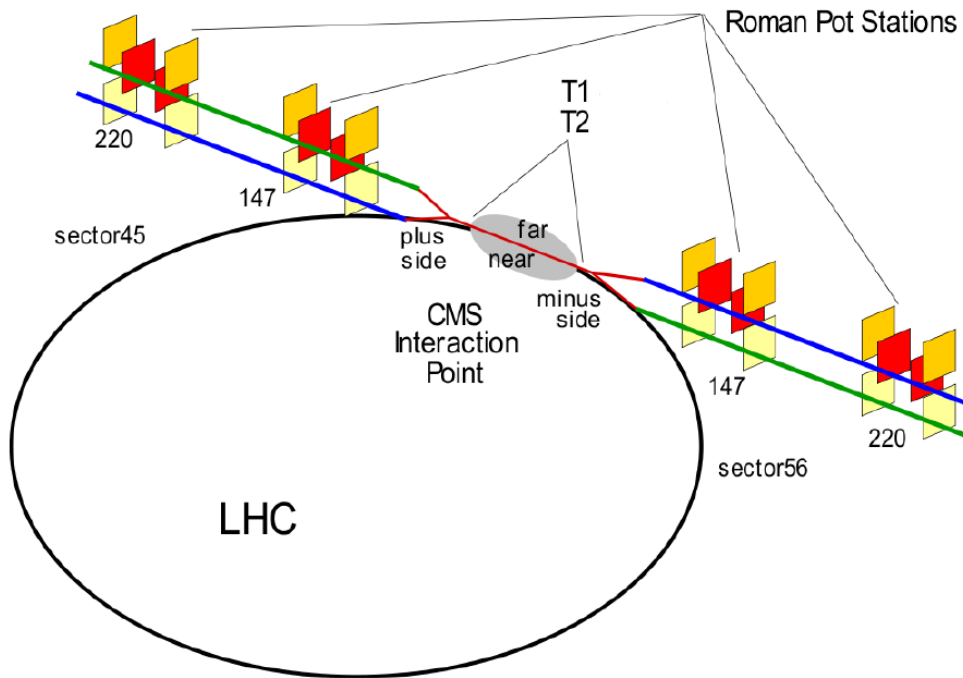


Figure 2.2. TOTEM detectors location at the LHC. Figure based on [12]

The *TOTEM* collaboration comprises about 80 physicists from 11 universities and laboratories in 8 countries. *TOTEM* is one of the smallest experiment at the *LHC* located near a bigger one - the *Compact Muon Solenoid (CMS)*. From the technical point of view it means that the data, trigger and commands format should be compatible with the *CMS* detector. *TOTEM* experiment uses its own detectors

3. mili barn, barn is a unit used in nuclear physics for expressing cross sectional area of nuclei. $1b = 10^{-28} \text{ m}^2$ (size of an uranium nucleus)

located in the neighbourhood of the *CMS* detector in Interaction Point 5. T1 and T2 detectors are situated 9 and 13,5m from the interaction point whereas Roman Pot detectors are located 147 and 220m from the center of the *CMS* detector. Positions of the detectors in relation to *CMS* interaction point and *LHC* tunnel are presented in Figure 2.2.

Sophisticated *DAQ* systems in close neighbourhood to the tunnel generate raw data stream stored on *CASTOR*. As *TOTEM* experiment is inseparably bound to the main *LHC* pipes and *CMS* interaction point, it relies heavily on the data coming from external systems. *TOTEM* uses environmental data coming from the LHC Logging Databases, like beam position, beam loss monitor, beam quality monitor or luminosity measurements. In addition there are several intermediate measurements which are calculated based on the raw data by means of the Offline Software. These are all ingredients needed to produce reliable physical results. Details regarding the current state of art of data processing within the *TOTEM* collaboration are discussed in Section 3.1.1 All of the data sources are given a specific insight in Section 3.2.1.

2.2. Used technologies overview

This section's aim is to provide a general overview of concepts and technologies used in the system implementation. Each subsection walks through the terms starting from the abstract concepts, through specifications, ending with concrete implementations.

2.2.1. SOA - ESB - OSGi

The terminology around *Service-Oriented Architecture* (*SOA*) is broad and very often misunderstood, an explanation for the main terms is needed in order to proceed. To have a clear view of what acutally *SOA*, *ESB*, *OSGi* and *ServiceMix* are, it is crucial to understand the relationship between these terms. Each of the terms will be discussed in following subsections in extent enabling to understand the main ideas behind them.

Service-Oriented Architecture

Service-Oriented Architecture (*SOA*) is an emerging approach to have software resources in an enterprise available and discoverable on network as well defined services[13]. Each service would achieve a predefined business objective and perform discrete units of work. The services are independent and do not depend on the context or state of the other services. Typically business operations running in an *SOA* comprise a number of invocations of these different components, often in an event-driven or asynchronous fashion that reflects the underlying business needs. Services work within distributed systems architecture.

The notion of *SOA* has been constantly evolving over the recent years and there is no single, official way of describing this concept. Nonetheless, a set of characteristics emerged which every solution meant to be built in *SOA* paradigm should follow [14]:

1. **Standardized Service Contracts**

Each of the services has to be recognizable for its functions and responsibilities with a clear, standards-based interface

2. **Service Loose Coupling**

A service should not be dependant on other services to perform its functionalities

3. **Service Abstraction**

Emphasizes the need to hide as much of the underlying details of a service as possible

4. **Service Reusability**

Principle known from *Object-Orientation (OO)* on the level of services

5. **Service Autonomy**

A service should care for its own environment without the need to escalate it on others

6. **Service Statelessness**

Management of excessive state information can compromise the availability of a service and undermine its scalability potential. Services should remain stateful only when required

7. **Service Discoverability**

The enterprise environment should be able to dynamically locate, identify and bind to any needed service

8. **Service Composability**

In order to solve bigger problems services have to be able to cooperate and be assembled into compositions realizing problem solution

SOA uses the find-bind-execute paradigm as shown in Figure 2.3. In this paradigm, service providers register their service in a public registry. This registry is used by consumers to find services that match certain criteria. If the registry has such a service, it provides the consumer with a contract and an endpoint address for that service [15].

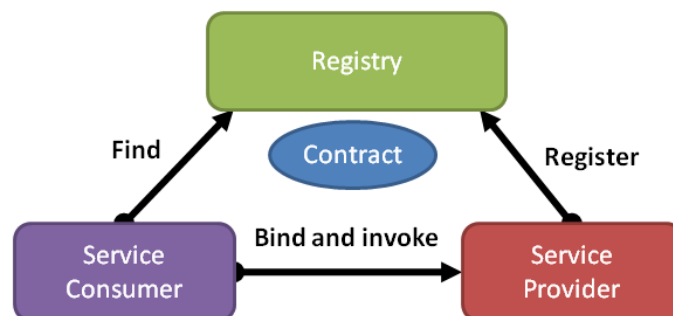


Figure 2.3. Find-bind-execute paradigm as the basic way of interactions between services in SOA. Figure based on [15].

It is crucial to understand what *SOA* really stands for, and what *SOA* is not. Table 2.1 pinpoints the most crucial aspects of this comparison.

SOA is nowadays the key principle for building reliable, extendable and dynamic enterprise solutions and is the subject of research for most big enterprises on the

SOA is not	SOA is a set of
a technology	guidelines
a product	acceptable and recommended practices
a protocol	architectural patterns
a standard	I

Table 2.1. What SOA is and what SOA is not

Information Technology (IT) market. At the beginning of the new millenium Gartner, the world-leading company in IT colsulting, predicted [16]:

By 2008, SOA will be a prevailing software engineering practice, ending the 40-year domination of monolithic software architecture (0.7 probability)

The recent years' boom around service-oriented technologies like Web Services and *ESBs* revealed the prophecy was true.

Enterprise Service Bus

In the days of high diversification of technologies, protocols, frameworks and design concepts there is a high demand on integration providers. To build an *SOA* conforming solution, a highly distributable communications and integration backbone is required[13]. This functionality is provided by the *Enterprise Service Bus (ESB)* that is an integration plarform which is obliged to support a wide variety of communications patterns over multiple transport protocols. An *ESB* acts as a shared messaging layer for connecting applications and other services throughout an enterprise computing infrastructure. Originally defined by analysts at Gartner, *ESB* is increasingly seen as a core component in a service-oriented infrastructure. It is important to understand, that *ESB* is one of many potential ways for realizing *SOA* solution.

Enterprise Service Bus introduces the concept of an integration bus, which connects services together. A conceptual view of such "device" is presented in Figure 2.4. *ESB* is the contradiction to the *Hub-and-spoke*⁴ topology of service communication.

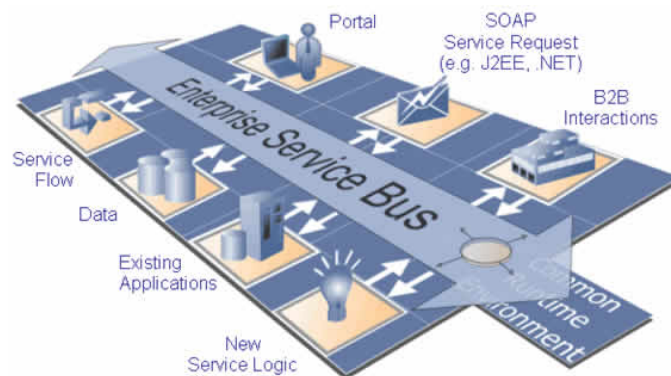


Figure 2.4. An abstract, graphical view of the Enterprise Service Bus acting as mediation middleware connecting Services together. Figure from [17].

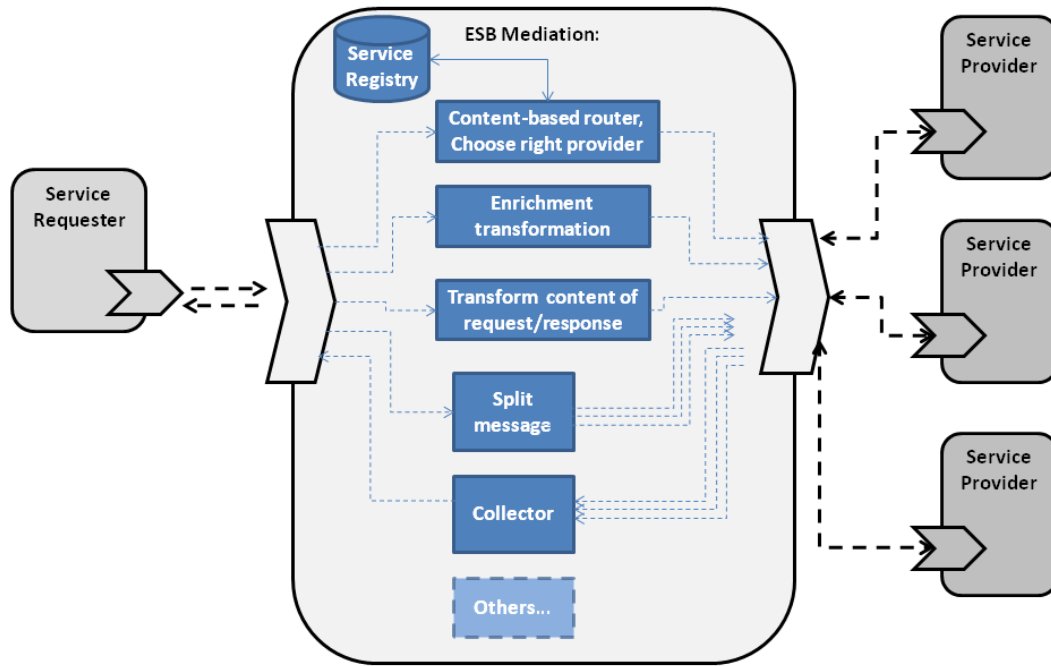


Figure 2.5. Mediation between Services by means of ESB. Figure based on [18].

Instead of a centralized point of communication, *ESB* provides the bus abstraction built on distributed network of messaging infrastructure. Figure 2.5 shows the logical relationship between service requesters, service providers, and the *ESB*. Service requesters and providers interact by exchanging messages. The *ESB*, acting as a logical intermediary in the interactions, provides loosely coupled interconnectivity between the requester of a function and the provider of that function. Its role as a logical intermediary allows the *ESB* to intercept messages and process them as they flow between a service requester and service provider. This processing is called mediation.

As previously mentioned, there are no strict requirements nor specifications describing precisely what an *ESB* should provide. Nonetheless, it is agreed that an *ESB* should follow a set of principles in order to provide the following values[19]:

- **Service virtualization**

Refers to the ability of the *ESB* to virtualize the following during service interactions:

- *Protocol and pattern*

Interacting participants need not use the same communication protocol or interaction pattern. For example, a requester may require interaction through some inherently synchronous protocol, but the service provider may require interaction using an inherently one-way protocol, using two correlated interactions. The *ESB* provides the conversion needed to mask the protocol and pattern switch.

4. Hub-and-spoke - is a way of connecting entities (services or other resources) with a central point of exchange each entity is connected to

- *Interface*

Service requesters and providers need not agree on the interface for an interaction. For example, the requester may use one form of message to retrieve customer information, and the provider may use another form. The *ESB* provides the transformation needed to reconcile the differences.

- *Identity*

A participant in an interaction need not know the identity (for example, the address) of other participants in the interaction. For example, service requesters need not be aware that a request could be serviced by any of several potential providers at different physical locations. The actual provider is known only to the *ESB*, and in fact, may change without impact to the requester. The *ESB* provides the routing needed to hide identity.

- **Aspect-oriented connectivity**

This principle in its idea is related to the concept of *Aspect-Oriented Programming* (*AOP*). *ESB* should provide means to realize *Cross-cutting concerns*⁵ without affecting the services implementation. These aspects may include logging, accounting, auditing, management, monitoring, security etc.

Just like *Service-Oriented Architecture* (*SOA*), *ESB* is only an abstract concept which determines a set of guidelines and wishes to be fulfilled by the implementing technologies. Till now, there is no single standard agreed on how an *ESB* should be implemented. Nonetheless from the descriptions above some conclusion regarding the *ESB* implementation can be drawn:

- Need for a runtime environment for the deployed services
- Runtime environment has to be responsible for fulfilling cross-cutting concerns, in order for the services to be maximally cohesive and loosely-coupled to each other
- Need for some reliable mean of communication between services
- Need for the ability to dynamically join services together in order to compose working application from available services
- Requirement to hide the technical details of communication protocols
- Requirement to obey open standards for services communication, like Web Services technology

Currently there is a wide range of products realizing the *ESB* concepts. These are both proprietary products and open-source projects using different approaches to realize the requirements listed above. The way of implementing these requirements varies in many aspects: runtime environment, communication backbone, integration and routing facilities etc. Ideally, an *ESB* solution should use a standardized container model, such as *Java Enterprise Edition* (*JEE*), *Java Business Integration* (*JB**I*), or *Open Services Gateway initiative* (*OSGi*), for managing deployed services.

In the next sections technologies used within the project, which is the basis of this thesis, will be discussed.

5. *Cross-cutting concerns* - are aspects not related to common functionalities; more in the glossary

OSGi

Open Services Gateway initiative (*OSGi*) is a technology specification created by the OSGi Alliance⁶. It is describing the Dynamic Module System for Java[21] - a framework that supports implementation of component-based, service-oriented applications in Java. The framework manages the life-cycle of modules (called *bundles* in *OSGi*) and provides means to publish and search for *services*. Moreover, it supports the dynamic deployment model in which modules can be installed, started, stopped, updated and uninstalled without requiring a reboot. It allows to build applications which consist of many independent, loosely-coupled components communicating with each other using their interfaces. Nowadays, *OSGi* is used in many application domains, including mobile phones, embedded devices, and application servers. *OSGi* introduces few basic concepts:

- **Bundle** is a collection of Java classes packed together in one *Java ARchive* (*JAR*) file. Each *bundle* specifies which Java packages it exports (allows other *bundles* to use them) and what packages it imports from other *bundles* to function. *Bundles* are deployed within a container which is the runtime of *OSGi*. All together the *bundles* form the environment for applications which may run within the framework. *Bundles* are a static way of reuse, because they provide class definitions to be later instantiated as *Services*.
- **Feature** is a collection of bundles. *Feature* is used to group bundles in order to be deployed together.
- Container is the runtime environment for *Bundles* and *Services*. *Container* is a single *Java Virtual Machine* (*JVM*) instance which provides mechanisms for dynamic component manipulation. *Container* integrates many applications in one place and allows them to share and reuse resources (classes, services) which are provided within it.
- **Service** is an instantiated Object which runs within the container and is registered in the Service Registry. Every application running within the container can publish its *services* but also look for *services* dealing with required tasks and use them in its activities. *Services* are a dynamic way of reuse, because they represent Objects, which are instantiated Java classes provided by *Bundles*. Good examples of services include:
 - persistence service, which takes care of storing data into the database;
 - management service, which exposes the functionality of *Java Management eXtensions* (*JMX*) for all bundles within the container.

OSGi was chosen as the runtime environment for services in the open-source *ESB* project - Apache ServiceMix from version 4.x.

2.2.2. MOM - JMS - ActiveMQ

For many years the synchronous models of communication between distributed applications have been dominating. Canonical examples include *Remote Procedure Call* (*RPC*) or the object-oriented Java version: *Remote Method Invocation* (*RMI*). This

6. OSGi Alliance is a consortium of IT companies to lead the development of OSGi technology[20]

way of application communication has many advantages when building monolithic applications. It is however not suitable in highly distributed, service-oriented application building, where we need reliable, dynamic and fault-tolerant way of communication.

Message-Oriented Middleware

Message-Oriented Middleware (MOM) is software infrastructure focused on sending and receiving messages between distributed systems. *MOM* allows application modules to be distributed over heterogeneous platforms, and reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces [22].

MOM is a software that resides in both portions of client/server architecture and typically supports asynchronous calls between the client and server applications. *MOM* reduces the involvement of application developers with the complexity of the master-slave nature of the client/server mechanism. *MOM* changes the way we look on the software development. From standard, synchronous invocations, we switch to focus on efficiency, asynchronicity, and independence.

Java Message Service

One of the realization of the *MOM* concept in the Java world is the *Java Message Service (JMS)*⁷ specification which is a part of *JEE* specification created by Sun Microsystems Inc.⁸

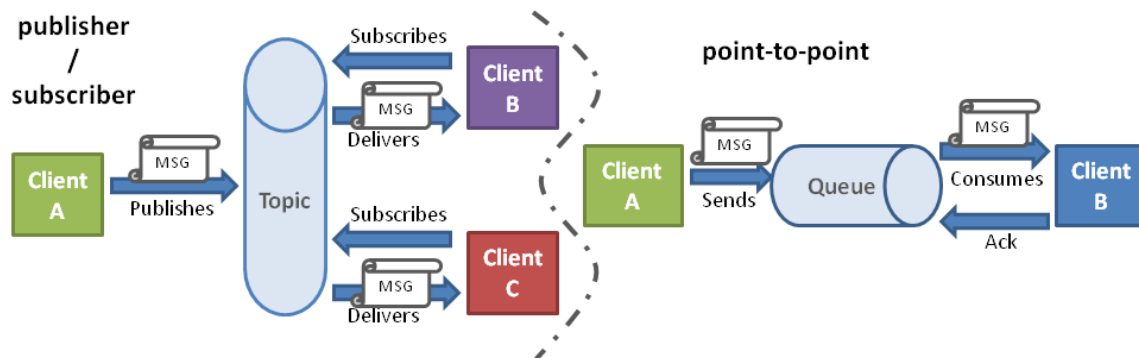


Figure 2.6. JMS publisher/subscriber and point-to-point exchange models. Figure based on [24].

It supports two messaging models: point-to-point (P2P) and publish/subscribe (pub/sub). P2P messaging is built around the concept of message **queues** where each *queue* forms a virtual communication channel. Each incoming **message** is addressed to one certain consumer for consumption, which resembles the one-to-one connection. In contrast, pub/sub messaging is built around the concept of **topics**. Each *message* may be delivered to multiple consumers which are interested in a specific *topic*, which maps to one-to-many connection. *Queues* and *topics* are referred to in *JMS* jargon as

7. Java Message Service is defined in JSR 914[23]

8. Sun Microsystems was bought by Oracle Corporation on January 27th, 2010

destinations.[25] These two exchange models are presented in Figure 2.6. The *JMS* application is composed of several parts:

- **JMS provider** is a messaging system that implements the JMS interfaces and provides administrative and control features. An implementation of the *JEE* platform at release 1.3 includes a JMS provider.
- **JMS clients** are programs or components, written in the Java programming language, that produce and consume messages.
- **Messages** are the objects that communicate information between JMS clients.
- **Administered objects** are preconfigured *JMS* objects created by an administrator for the use of clients. The two kinds of administered objects are *destinations* and *connection factories*
- **Native clients** are programs that use a messaging product's native client *Application Programming Interface (API)* instead of the JMS API. An application first created before the *JMS API* became available and subsequently modified is likely to include both JMS and native clients.

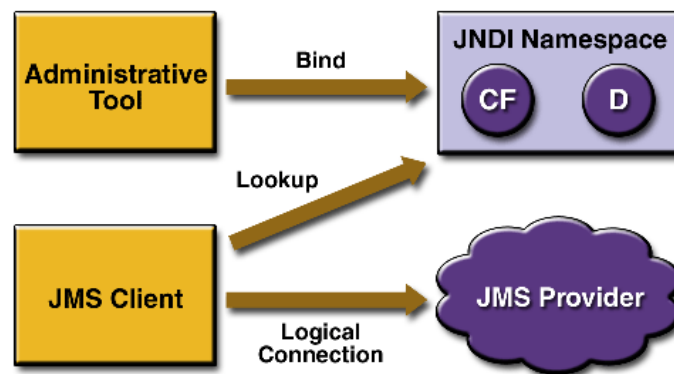


Figure 2.7. Elements of the JMS architecture and relationships between them. Figure from [24].

Figure 2.7 shows relations between the elements of the specification. Administrative tools bind destinations and connection factories into *Java Naming and Directory Interface (JNDI)* namespace. A *JMS* client looks up the administered objects and establishes a logical connection to the *JMS* provider.

Apache ActiveMQ

One of the implementations of the *JMS* specification is the Apache ActiveMQ⁹ project. Besides full implementation of the *JMS* specification, it provides many advanced features like clustering, multiple message stores, and ability to use any database as a JMS persistence provider besides VM, cache, and journal persistency. Apart from Java, ActiveMQ can be also used from other popular programming languages (.NET, C/C++, Delphi, Perl, Python, PHP, Ruby) together with connecting to many protocols and platforms [26]. These include several standard wire-level protocols, plus

9. <http://activemq.apache.org>

specially designed protocol called OpenWire, for fast, binary communication. These features make ActiveMQ a perfect tool for performing integration point across applications built using different technologies.

ActiveMQ was chosen as the messaging layer in the open-source *ESB* implementation - Apache ServiceMix 4.x. It is also used within this thesis work as the data transfer infrastructure. ActiveMQ proves to be one of the most advanced *JMS*-compliant implementations achieving high results in benchmarks[25].

2.2.3. EIP - Camel

There are plenty of problems software developers struggle every day and instead of finding an original solution every time (which might not function very well) it is better to follow the guidelines of acknowledged software engineering practices - the *design patterns*.

Enterprise Integration Patterns

In the world of object-oriented programming there are plenty of scenarios in which applying certain solution simplifies the architecture and helps avoiding further pitfalls. The book *Design Patterns: Elements of Reusable Object-Oriented Software* is considered a “bible” of design patterns for *OO* paradigm [27].

Similarly, when it comes to software integration there are repeating, common scenarios where certain solutions can be applied. That’s why a set of patterns was gathered which are a kind of standardized integration language. *Enterprise Integration Patterns (EIP)* provides a consistent vocabulary and visual notation to describe large-scale integration solutions across many implementation technologies. It also explores in detail the advantages and limitations of asynchronous messaging architectures[28]. *EIPs* are becoming, by its comprehensiveness and popularity, a common language for software integration.

Apache Camel

Apache Camel¹⁰ is an open-source integration framework based on known *EIPs* defined in the book *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*[29]. Camel allows to configure routing and mediation by using Java or Scala *Domain Specific Language (DSL)* or by Spring based *eXtensible Markup Language (XML)* file. Camel is virtualizing different sources and protocols by the concept of an endpoint, which is described by an *Uniform Resource Identifier (URI)* in similar way to the addresses in the Internet. It supports variety of protocols by over 100 supported URIs including HTTP, CXF, File, FTP, IRC, iBATIS, JDBC, JMS, RMI and many more. Using Camel, application integration seems like building program out of blocks¹¹.

Table 2.2 presents three ways of defining a sample integration solution using Apache Camel:

- Java DSL

10. <http://camel.apache.org/>

11. There is already a graphical interface for building Camel Routes - FUSE IDE for Camel, which enables to use drag-and-drop technique

Java DSL	Spring XML DSL
<pre> from("jms:queue:potentiallyReadyRequests") .choice() .when(header("isReady").isEqualTo("true")) .to("jpa:myRequestsDatabase") .otherwise() .to("bean:requestProcessing?method=process") .to("jms:queue:potentiallyReadyRequests"); </pre>	<pre> <route> <from uri="jms:queue:potentiallyReadyRequests"/> <choice> <when> <xpath>\$isReady = 'true'</xpath> <to uri="jpa:myRequestsDatabase"/> </when> <otherwise> <to uri="bean:requestProcessing?method=process"/> <to uri="jms:queue:potentiallyReadyRequests"/> </otherwise> </choice> </route> </pre>

Table 2.2. An example route in notion of Enterprise Integration Patterns

- Spring XML DSL
- graphical representation using *EIP* notation

This route realizes the concept of a Content-based Router pattern, which is visualized in the diagram. Code fragment shows an example of a route, which listens for messages on a *JMS* queue and sends them either for further processing by a Java class (bean endpoint) or to be stored in the database (via *Java Persistence API (JPA)* component).

Declarative way of defining the routing among certain components promotes high cohesion and loose coupling of the components. Components should be specialized pieces of software and do not have to be aware of other components. Camel, as the integration framework will tie all parts together creating a new piece of software from working blocks. Apache Camel integrates seamlessly with other standards-based projects like ActiveMQ or Apache CXF. Camel is discussed further in Section 4.2.6, where implementation issues are analyzed.

2.2.4. IoC - DI - Spring

One of the revolutionary concepts in *Object-Oriented Programming (OOP)* was the introduction of *Inversion of Control (IoC)* pattern and its *Dependency Injection (DI)* variant. It was the conceptual beginning of the container-based applications. Containers providing IoC functionality relieve the programmer from the duty to perform repeating jobs and lowers complexity of the code. These concepts are widespread

in the *IT* industry and have proven to aid the development process, reusability and *Plain Old Java Object (POJO)* style coding.

Inversion of Control

Inversion of Control (IoC) is a design principle where reusable generic code controls the execution of problem-specific code. The responsibility to do certain jobs (providing database connection, transaction management etc.) is taken out from the component, which allows it to be focused on performing its desired functionality.

IoC is a key part of what makes a framework different to a library. A library is essentially a set of functions that you can call, usually organized into classes. Each call does some work and returns control to the client [30]. The framework realizing *IoC* on the other hand, gets instructed which aspects in the life-cycle of the component should it control and relieves the programmer from coding it. Such decoupling should result in more cohesive component design, and mitigate the dependencies overhead on the components.

Dependency Injection

Dependency Injection (DI) is one of the ways of realizing the *IoC* concept. *DI* is a design pattern consisting in removing direct dependencies among components in favour for an external entity, which takes care of it. Instead of hard-coding dependencies¹² we rather configure the relations of the components externally. Then the container is performing the initialization for us, with respect to the principle: "A class should not configure itself but should be configured from outside". It is useful to use *DI* in order for the code to be cleaner and vendor-independent. *DI* is also helpful while performing object-oriented tests. It promotes *POJO* style coding where dependent component can be substituted to mock object which is usable while testing.

Spring framework

An example of concrete implementation of the above concepts is Spring framework¹³. Spring as its core uses the *DI* container which leverages *POJO* programming, ease of extension, loose-coupling and configuration-over-coding. Spring container allows to inject required objects into other objects seamlessly, without the code being aware of the object configuration. This results in a design in which the Java classes are not hard-coupled and better stick to the *POJO* programming model. The injection in Spring is either done via setter injection or via construction injection. These classes which are managed by Spring must conform to the JavaBean standard. In the context of Spring, classes are also referred to as beans or as spring beans. The Spring core container handles the configuration, generally based on annotations or on an *XML* file (*XMLBeanFactory*) and manages the selected Java classes via the *BeanFactory*. The container is responsible to solve all the objects' dependencies and make it available to the code which is being instrumented.

12. like for example - initialization of the database driver and it's configuration within the code

13. <http://www.springsource.org/>

2.2.5. ORM - JPA - Hibernate

Relational databases are nowadays extensively used as the facility to perform data storage. Having incontrovertible advantages relational way of organising data does not suit the object-oriented way of thinking. It is the reason why there is a need to somehow map the entities in the database to objects corresponding them in a working program.

Object-Relational Mapping

Object-Relational Mapping (ORM) is a concept which enables the most popular object-oriented programming languages to easily and seamlessly communicate with the relational database. *ORM* lets us look at the *Entity Relationship Diagram (ERD)* diagram from the perspective of objects, which by their relations to each other, form the dependency graph. Compared to traditional techniques of exchange between an object-oriented language and a relational database, *ORM* often reduces the amount of code that needs to be written and simplifies working with the data storage. *ORM* can also provide means of realising cross-cutting concepts like logging, accounting, security etc.

Java Persistence API

Java Persistence API (JPA) is a standard *API* agreed as the JSR 220¹⁴ in May, 2006 as a part of the *Enterprise Java Beans (EJB)* 3.0 specification. *JPA* is the *ORM* standard for Java programming language. From the point of view of the programmer it provides the capability of managing object entities and interact with the database by means of the *EntityManager* object. The way in which the objects and their relations map to the entities in the database are defined by means of Java Annotations or *XML* documents. The *EntityManager* object, which is the core of the *JPA* specifications, among the functionalities to manipulate the data (inserting, updating, deleting etc.) provides the *JPA Query Language*, which is similar to *Structured Query Language (SQL)*.

JPA hides the concrete provider of the *ORM* service by the unified *API*. It removes the need to stick to vendor-specific concepts which increases interoperability, compatibility and simplifies code understanding.

Hibernate

JBoss Hibernate¹⁵ is one of the products implementing the *JPA* specification and can serve as the provider in the *JPA* model. Hibernate's primary feature is mapping from Java classes to database tables (and from Java data types to *SQL* data types). Hibernate provides data query and retrieval facilities, also it functions as a cache for query execution optimization. Hibernate generates the *SQL* calls and attempts to relieve the developer from manual result set handling and object conversion and keep the application portable to all supported *SQL* databases with little performance overhead.

14. There is already version 2.0 of the standard agreed on December, 2009 under JSR 317

15. <http://www.hibernate.org/>

2.3. Clarification summary

As the previous sections are overloaded with concepts, definitions, abbreviations and specification names it might be difficult to have a right picture of them. Figure 2.8 presents the concepts stack in order to clarify the relationships between all those entities.

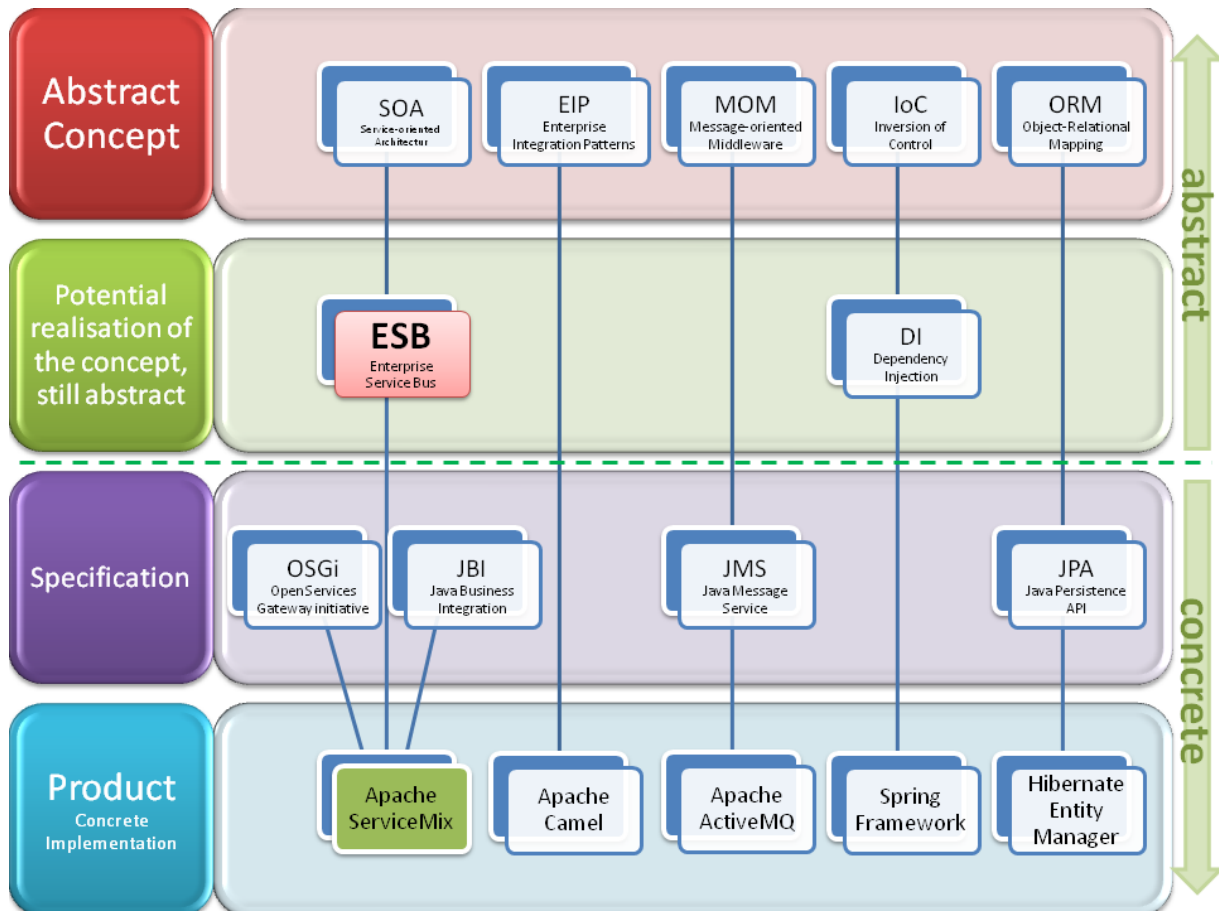


Figure 2.8. Graph of the main concepts of the thesis presenting relations between them.

At the top, we have the most abstract concept of *Service-Oriented Architecture* (SOA). *Enterprise Service Bus* (ESB) is an infrastructure which enables building applications conforming the SOA paradigm. There is no such thing like the ESB specification - it is still an abstract concept. OSGi is a specification for Java Dynamic Modules, which might be considered the deployment environment for the services within the ESB. Finally, Apache ServiceMix is a concrete piece of software which is the full implementation of ESB concept on the basis of OSGi specification.

Similarly, *Enterprise Integration Patterns* (EIP) are conceptual entities describing integration scenarios. Apache Camel is the concrete implementation of these patterns, ready to be used in software solutions.

MOM describes the idea of asynchronous communication by means of messaging. JMS is a specification which realizes the MOM concepts in the Java world. Apache

ActiveMQ is an exemplary representant of the *JMS* specification implementation, which adds also plenty of other features.

The *IoC* concepts describe a wish in which application deployment and execution is controled from outside of the program. *DI* gives a possibility to relieve the programmer from configuring object dependencies inside the application code, which in fact is an example for realizing the *IoC* concept. At the bottom we have a concrete product - Spring Framework which, among others, realises the *IoC* and *DI* principles.

Finally, *ORM* is an idea of marrying the object-oriented and relational-oriented worlds. *JPA* is the standardised Java *API* fully realizing the *ORM* principles. JBoss Hibernate (with its EntityManager component) can be used as a *JPA* provider, just like other solutions conforming to the *JPA* specifications.

2.4. Selected frameworks

This thesis focuses on the *ESB* usage in systems integration and data transferring scenarios. In order to be able to evaluate this composition, choices regarding used frameworks had to be done. This section describes the concrete frameworks used within this thesis implementation works.

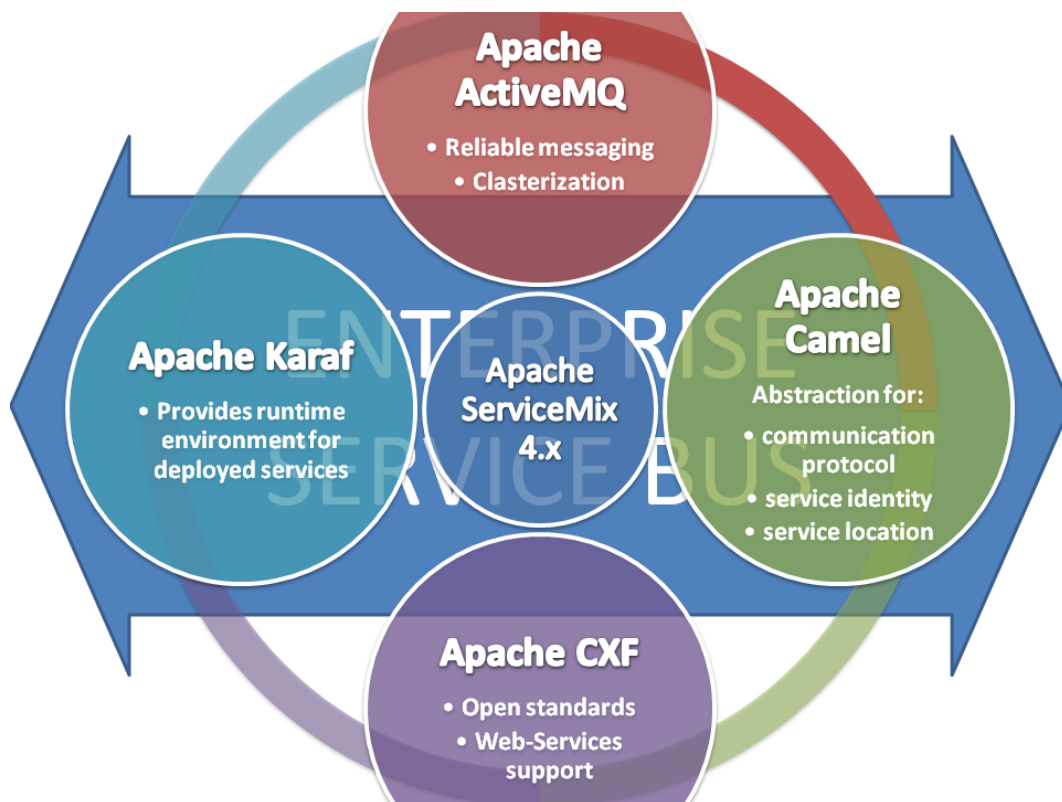


Figure 2.9. Main components of ServiceMix providing ESB facilities.

2.4.1. Apache ServiceMix 4.x

Apache ServiceMix is an open-source *ESB* released under the Apache License. There are two mainstream versions of ServiceMix currently available - 3.x and 4.x. Version 3.x of ServiceMix is based on the *JB*I specification. Version 4.x keeps support for *JB*I, while providing also *OSGi* runtime as its core. In this thesis work ServiceMix 4.3 (4.3.1-fuse-00-00) is being used.

In order to fulfill the *ESB* principles (Section 2.2.1), ServiceMix takes advantages of already presented frameworks to achieve certain goals:

OSGi and JBI used for services deployment and management

Apache Karaf¹⁶ is an OSGi-based runtime that provides a lightweight container into which you can deploy various components and applications. Karaf supports the following *OSGi* containers: Apache Felix¹⁷ and Eclipse Equinox¹⁸. Karaf hosts the *OSGi* runtime while providing useful management features: hot-deployment, logging, dynamic configuration, shell console or remote access. Architecture of Apache Karaf is illustrated in Figure 2.10.

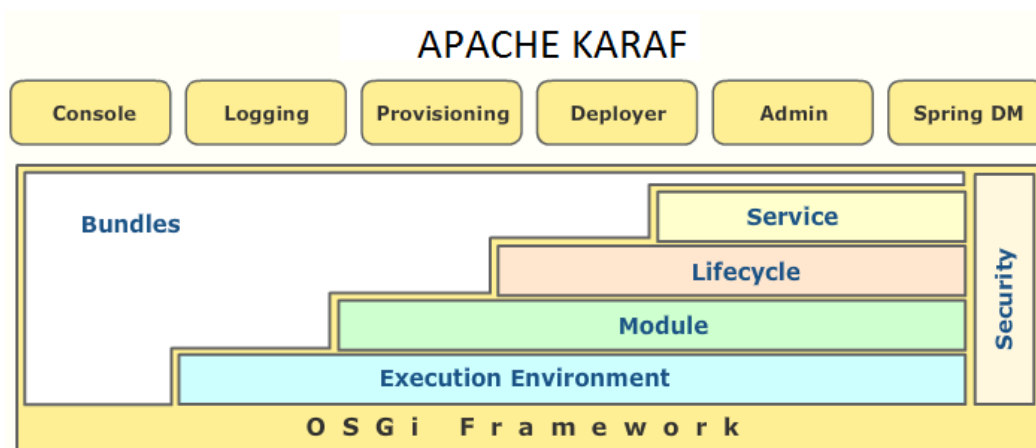


Figure 2.10. Architecture of Apache Karaf. Figure based on [31].

Apache CXF is used as the WebServices hosting framework

Apache CXF¹⁹ is an open-source services framework, which helps to build and develop services using frontend programming *APIs*, like JAX-WS and JAX-RS. CXF provides support for variety of protocols such as SOAP, *XML*, RESTful HTTP or CORBA over many transport protocols, including HTTP or *JMS*.

Spring support

Spring provides the IoC functionalities as well as other utilities, e.g. *JMS* or *JNDI* template, Spring Dynamic Modules, etc.

Apache ActiveMQ provides reliable messaging services

ServiceMix as its core for services communication uses ActiveMQ, which is also the preferred *JMS* provider for deployed applications.

16. <http://karaf.apache.org>

17. <http://karaf.apache.org/>

18. <http://www.eclipse.org/equinox>

19. <http://cxf.apache.org/>

Apache Camel is used for mediation and routing

Camel with its high-level abstraction of endpoints and integration patterns ties the services together to form business flows.

Figure 2.9 depicts the components which working together fulfill the *ESB* guidelines.

2.4.2. FUSE ESB 4

FUSE ESB 4 is the enterprise release of ServiceMix 4.x developed by FuseSource²⁰. FuseSource provides regular release policy, tested features, support and well-organized documentation, staying open-source and free to use. These are sufficient arguments to use FUSE ESB over ServiceMix itself.

2.5. Related works

There are many works which are concerning data transfer in general. Some science experiments or research projects have their own data management solution to meet specific requirements. This section discusses some of the works related to the topic of asynchronous data transfer and integration issues. All of the discussed systems perform data transfer of rather "low-level" data formats, like files. System designed by the author of this thesis is an integration solution, which has to deal with different protocols and data formats used by the data providers. The data is being transformed, filtered and processed to conform to the output format. It differs significantly from basic file transfer. Nonetheless, interesting approaches and techniques used by these systems can be considered for adaptation into future releases of the system discussed in this thesis.

Data Transfer Library

A paper presented at the 19th ACM International Symposium on High Performance Distributed Computing in 2010 [32] introduces a conceptual bulk data transfer framework, for usage in large-scale scientific experiments. Data Transfer Library provides *API* for asynchronous, fast and fault-tolerant large-volume data transfer. DTL is divided into several layers, each of different responsibility. At the top it accepts job issuing from higher layers, then it submits and manages the data transfer. DTL also uses thread and queue mechanisms to implement asynchronous data transfers. Data transfer mechanism is meant to be extensible but only GridFTP protocol is currently supported.

The whole framework is focused around file transferring. It is not suitable in the *TOTEM* experiment scenario where we are facing scenario in which many different sources of data are being integrated into single database population system. Before putting data into the database it is also essential to do the data processing as well. Parallel works on *DBPop* and Data Transfer Library and similar ideas for data transfer organisation prove the interest in using integration and asynchronous messaging concepts in data transfer.

20. <http://fusesource.com>

DataMINX Data Transfer Service

DataMINX Data Transfer Service [33](DTS) is an open-source project which focuses on file-based data transfer. Works begun in 2010 and till now, the prototype can handle simple data transfer jobs. It supports GridFTP, *File Transfer Protocol (FTP)*, and file protocols, it can do file-to-directory, directory-to-directory, and file-to-file transfers. As its core it uses JMS queues infrastructure as well as small integration framework called Spring Integration. It is using Spring Batch as the skeleton for batch jobs management, which might be considered for adoption in future releases of *DBPop* since it might be a more standard-based approach.

While showing many other, useful approaches to the data-transfer topic, DataMINX does not stick to *TOTEM* Experiment scenario. The main aim achieved by the project is the file transfer, while other possible ways of data transfer are not considered.

Other works

Data transfer in general is the most basic purpose of computer networks and computer science in general. Since it is a wide field of research there are many efforts dealing with improving data transfer for variety of purposes. The following paragraph shortly describes some of the main works which are dealing with data transfer. Aspera FASP [34] is a protocol dedicated for data-transferring which mitigates the TCP overlap for better performance. The PhEDEx [35] data transfer management system is used by the *CMS* experiment at *CERN* for file replication, tape migration and routing decisions. Plenty of other works related to data movement are listed in paper [32].

2.6. Chapter summary

This chapter begun with an overview of *HEP* data processings systems. The *Offline* and *Online processing* phases were described as well as the organisation of data storage at *CERN*. Basic information about the *TOTEM* experiment was introduced. Next section provided a short tutorial into the most significant technologies used in the *DBPop* solution. Very important concepts of *SOA*, *ORM*, *EIP* together with their practical implementations were presented. The concept of an *ESB* and its concrete implementation, the Apache ServiceMix project, was discussed in great detail. At the end, projects related to data transferring and integration were given an overview in order to have a more comprehensive look at the domain of the problem.

Chapter 3

System model

This chapter's aim is to describe full motivation for constructing the *TOTEM DataBase POPulation System (DBPop)*. At first, description of the existing state is given, which covers details of the methods used within the *TOTEM* experiment to leverage physics research. Afterwards, problem statement and full requirements analysis is performed. Next section is presenting the *DBPop* system model in great detail. Architecture, data sources, integration policies and working mechanisms of the proposed solution are given detailed clarification.

Sections 3.1 and 3.2 are introducing a lot of physics related and *TOTEM* specific information, not strictly relevant to this thesis topic, however crucial to understand the data flow in the *TOTEM* experiment. Rest of the chapter is discussing the system model in reference to the information provided in physics specific sections. Problem statement and requirements analysis are performed in Sections 3.3 and 3.3.1. Last section considers application architecture from several perspectives.

3.1. Current state of art analysis

Previous chapter introduced the *High Energy Physics (HEP)* data life-cycle and provided general overview of the *TOTAL Elastic Scattering and Diffraction Measurement (TOTEM)* experiment at *European Organization for Nuclear Research (CERN)*. This section discusses *Online processing* phase in the *TOTEM* experiment, which is focused on real-time hardware and primary data placement, as well as the *Offline processing*, which is performing the actual physics research based on the data acquired by the *DAQ* systems. In this section the information provided in Section 2.1 are complemented with current state of art analysis.

3.1.1. Current processing model

Schematic view of the current processing model is presented in Figure 3.1. Raw data coming from the detectors' (called: *Roman Pots*, *T1*, *T2*) chips (*Very Forward ATLAS and TOTEM chip (VFAT)*) is being transmitted via *Gigabit Optical Link (GOL)* to the *TOTEM Counting-room* [12]. This is indicated with letter A in the diagram. *TOTEM Front End Driver (TOTFed)* is equipped with *Optical Receiver (Op-*

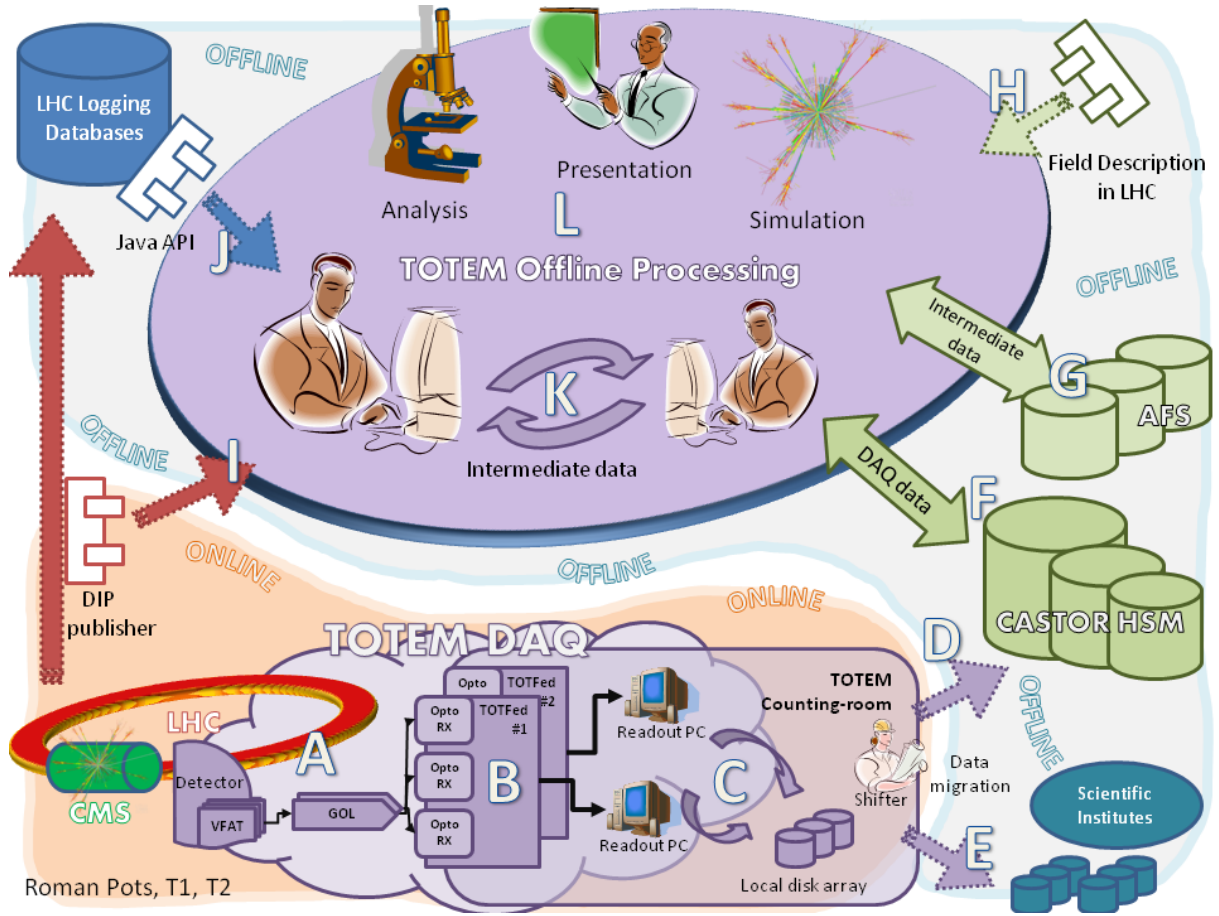


Figure 3.1. Current data processing model in the TOTEM experiment.

toRX) units transmitting the data to the Readout PCs. Letter B shows this phase. After being initially processed the data is stored in raw *VMEA data format* (*VMEA*) on the local disk array, presented with letter C. From one data taking (a so-called *TOTEM Run*¹) a run file is generated, composed of **events** frames. An *event* is basically a set of raw detector measurements in certain moment of time while the data taking. Since a single run aggregates huge amounts of *events* in order for the files not to be too big, they are splitted into several parts. Splitted files consists of about 20 to 50 thousand *events*. These files are put under the process of **reconstruction** in order to transform raw detector state into significant physics data representing *collisions*. This process is triggered by the *shifter*. The reconstruction process stores its output as *ROOT* files, ready to be analyzed by the scientists. When the data gathering is finished, the reasonable data (filtered of corrupted and/or badly taken) is transferred by a shifter to the *HSM - CASTOR*, indicated by letter D in the diagram. This is the point at which the data comes from the restricted TOTEM-only network to the public CERN subnetwork where it is easy to access any data in the common pool. The same data gets also to clusters in *Istituto Nazionale di Fisica Nucleare (INFN)* in

1. *TOTEM Run* is the period of time when the TOTEM detectors are running; more in the glossary

Genova, Italy and other institutes of the *TOTEM* collaboration for backup storage. This is presented with letter E in the diagram.

The main data of interest in the *TOTEM* experiment are those coming from the *DAQ* and intermediate calculations (stored in F and G). These are used in each of the activities performed by the scientists, mainly (L):

- analysis
- simulations
- alignment calculations
- presentation
- data mining

Data coming from the *DAQ* systems are not sufficient to perform all the calculations. There are other, external data sources delivering demanded data, including:

- LHC Logging Databases (letter J)
- *LHC Software Architecture (LSA)* and *Field Description for the LHC (FiDeL)* (letter H)
- *CERN Data Interchange Protocol (DIP)* for accessing real-time LHC data (letter I)
- *TOTEM Offline Software* or *TOTEM Detector Control System (DCS)* (letter K)

These sources are described in Section 3.2.1.

3.1.2. Existing state evaluation

Scientists performing research need to have access to range of necessary data. By now the data exchange is done by using variety of access methods, protocols and sources which are distributed among different sites at *CERN*. For instance, a scientist who needs the positions of the Roman Pots has to get the data from LHC Logging databases and store them locally in his file system. Afterwards, he calculates intermediate values (e.g. *Alignment*) and pushes them on the shared file system - *AFS* to be available for other scientists. Someone who is performing simulations or pattern recognition is using these data in his activities. This is only one of the scenarios performed by the physicists on their daily basis. This kind of cooperation and data sharing is highly ineffective since it requires all the people to have the knowledge of the data access methods, places of storage and relies on many external systems which is error prone.

To improve current processing model an idea of a centralized databank emerged, which will aggregate the necessary data in a common place. Before applying such solution it is crucial to identify all of the data sources and understand the way data flows between all interested sites.

3.2. The TOTEM Offline Database Project

The way scientists work in the *TOTEM* experiment does not conform current trends in the *IT* industry. The *TOTEM* software roadmap considers the fact, that the data processing model has to lean towards using a centralised data source in

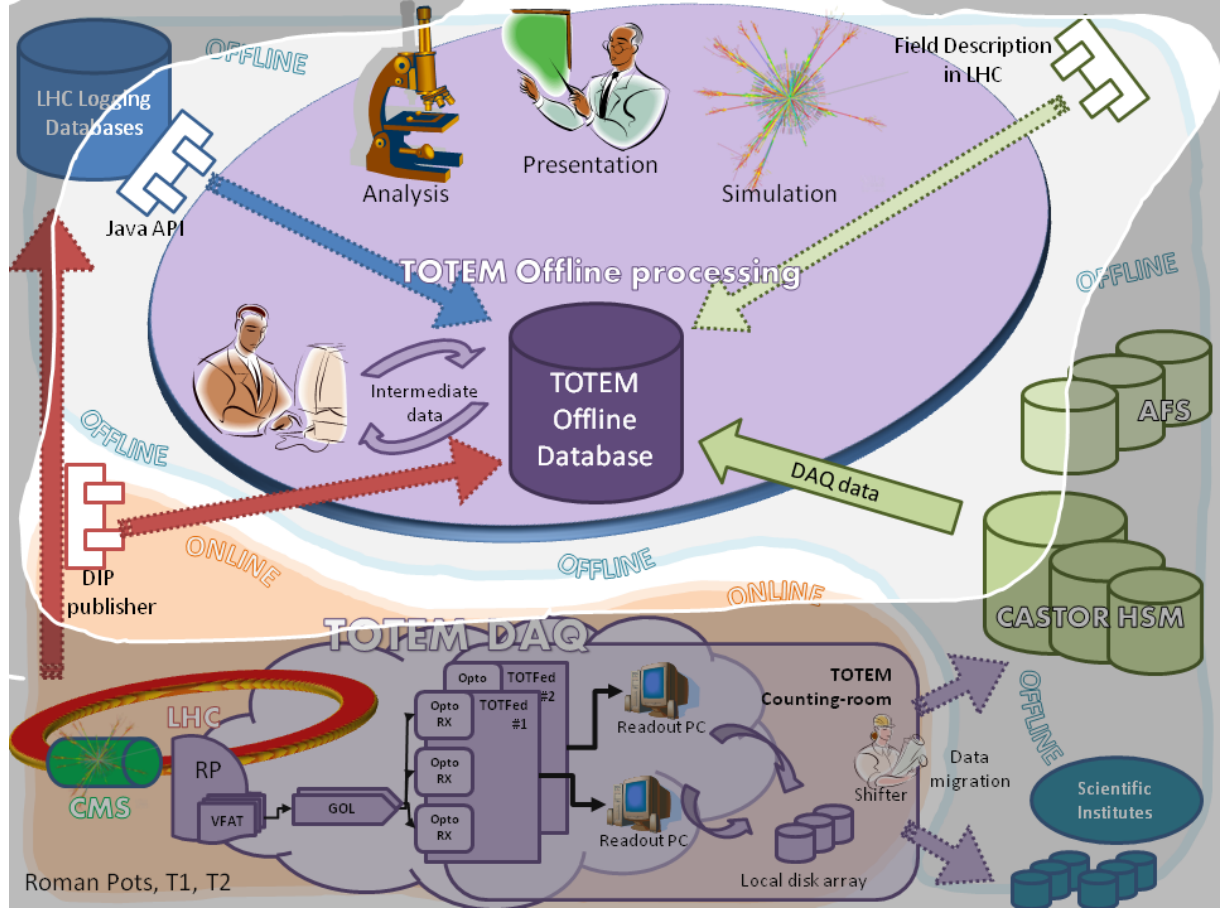


Figure 3.2. Processing model with the central data storage - the TOTEM Offline Database.

order to simplify data gathering for performing analysis. Currently much of the work is performed “by hand”, which mitigates the effort of scientists, who spend their time on accessing different data sources instead of processing the data.

Main aim of introducing the central database is to integrate all the data sources in one place. TOTEM Offline Database will contain all the information needed by *TOTEM* scientists to do the research. This model of processing will have positive impact on work efficiency and will promote separation of concerns: data getting and data using. Everyone will basically turn to the database to get the necessary data without having to know access methods for the variety of sources. Central point of data distribution will cause reduced network traffic saving resources of the external systems. Figure 3.2 presents the potential processing model resulting from the Offline Database introduction. Please note the differences from the model presented in Figure 3.1.

TOTEM Offline Database schema was designed to store any kind of data that describes the environment (experimental apparatus, settings, configurations and data taking conditions) during the reconstruction. There are several data sources and the measurements can be related to any level of the detector’s structure. All the conditions data are identified by the category to which they belong and by their *Interval Of*

Validity (IOV) - the time period for which a datum is valid. Database schema of the TOTEM Offline Database is discussed in Section 4.3. The database is based on Oracle technology, its running conditions are maintained by the Oracle Database Services at CERN, which assures high speed, availability, clusterization and serviceability.

3.2.1. Data sources identification

The actual state of the TOTEM Offline Database depends on several external systems, which are: data sources, data modifiers and data readers. In order to have the best control over what is being written to the database, it is essential to have the intermediate layer which will integrate all the sources and provide single way of modifying contents of the database. The following subsections discuss the identified data manipulators.

DAQ

Data coming from the TOTEM DAQ systems is stored on a disk array in *TOTEM Counting-room* in the file system. *VMEA* is a specialized data format conveying detectors' reads. The shifter is responsible for triggering reconstruction process (done by means of *Compact Muon Solenoid SoftWare (CMSSW)* framework) which takes the *VMEA* files and produces *ROOT* compatible, reconstructed events. These files are being migrated to *CASTOR* as well as to the clusters in TOTEM institutes around the Europe for backup storage. Each of the files is representing a single *TOTEM Run*.

A single *TOTEM Run* is being performed within certain period of time and consists of *events*. The data gathering within a *TOTEM Run* is done while the *LHC* is working and delivering stable beam, which is a desired beam status while an *LHC Fill* (more in the Tests chapter and in the glossary). Data gathered by the *DAQ* is crucial for all analysis performed within the *TOTEM* experiment.

LHC Logging Databases

Main purpose of the Logging Service is the persistence of logged time-series data for the lifetime of the *LHC*. This aim is fulfilled by the Oracle database management system which stores the data in two distinct databases:

- *Measurements DataBase (MDB)* which stores the unfiltered data for 7 days
- *Logging DataBase (LDB)* which stores filtered data for the lifetime of the LHC

Architecture of the system is presented in Figure 3.3.

LHC Logging facilities have been evolving since the year 2001 [36]. Data extraction requirements have evolved significantly from that time. Currently, the main principle of the whole system is to provide large variety of data concerning LHC and experiments around it, hidden behind single access facade. To fulfill these requirements, extraction is done by means of a Java *API* - there is no *SQL* access allowed. Interested sites can build applications exploiting the *API* capabilities or can use a tool called Timber² which is a graphical interface built on top of the *API*. Direct database access must be avoided, it has several disadvantages:

- **not scalable** across all clients, because of:

2. <http://cern.ch/timber> - available only within the CERN network

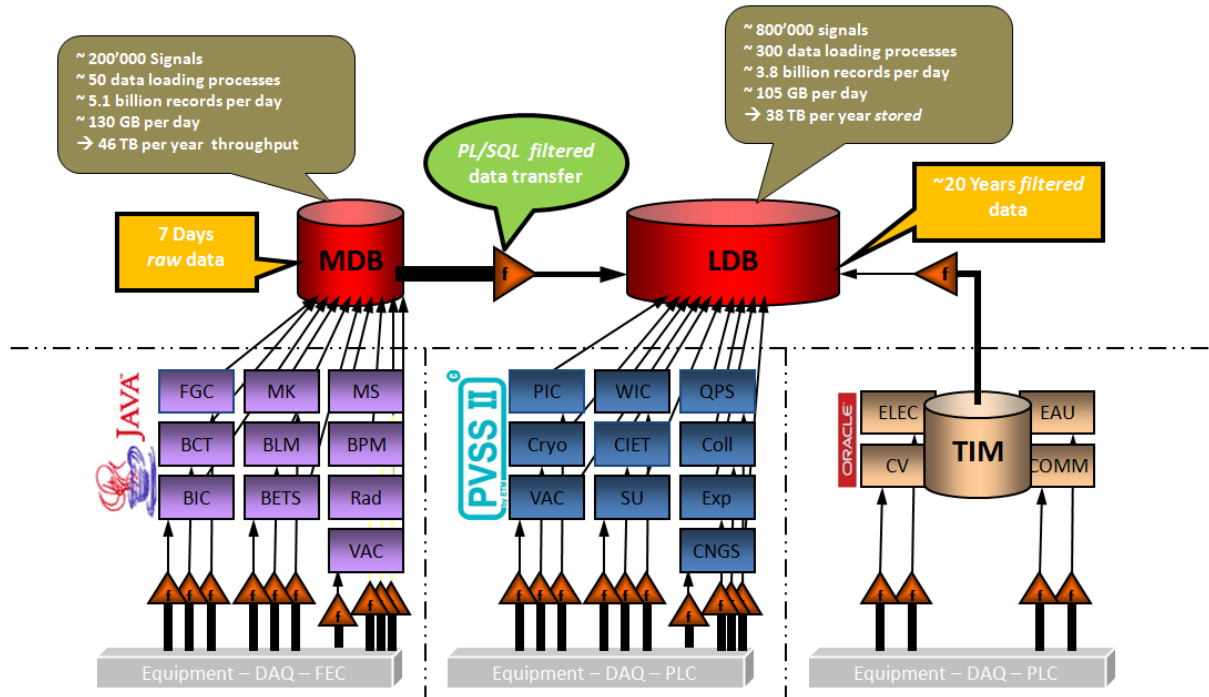


Figure 3.3. LHC Logging Databases data sources aggregation. Figure based on [36].

- number of connections
- security considerations
- volatile infrastructure
- **not secure** - badly written queries or application logic may corrupt the database state or crash the entire service (leaving database in inconsistent state may lead to failures)
- **not performant** - most programming languages provide database access, but only few languages are optimized to work with Oracle in a performant manner

The concept of hiding access to databases behind the Java API, using *Java DataBase Connectivity (JDBC)* under the hood, resolves these problems.

- *JDBC* fulfills the requirements, with respect to Oracle performance, as it supports:
 - connection pooling
 - statement caching
 - bind variables
 - flexible array fetching
- 3-tier architecture has more benefits:
 - resource pooling (connections, statements)
 - database protection and isolation, users don't have to care about:
 - database schema
 - server details and login credentials
 - access to restricted technical network

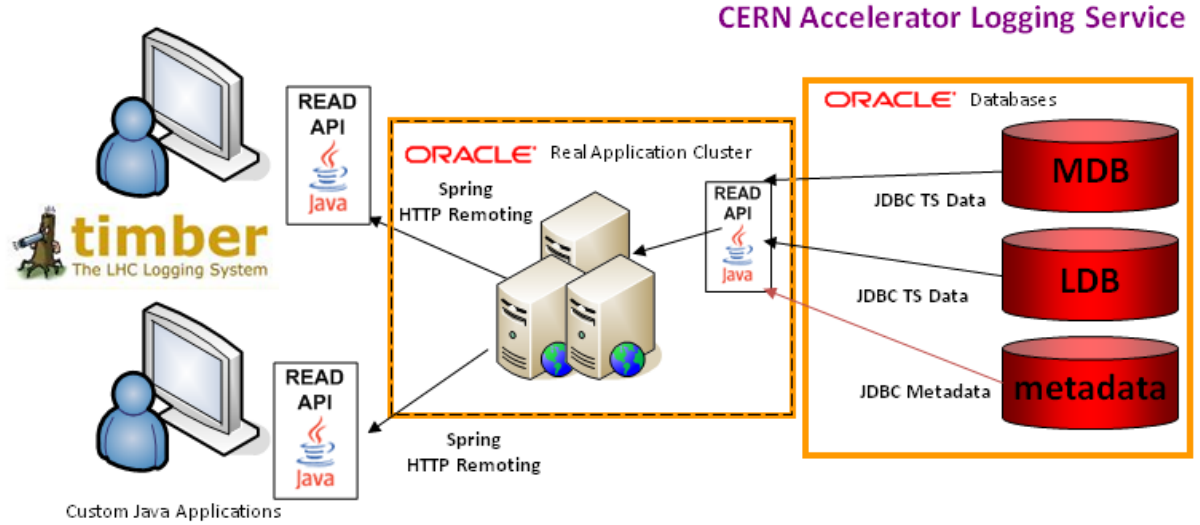


Figure 3.4. LHC Logging Java API database isolation concept. Figure based on [36].

Purpose of the Java API is clearly shown in Figure 3.4. Principles used in the LHC Logging project were very important in construction of the *TOTEM DBPop* system.

TOTEM Offline software

TOTEM Offline Software[37] is a programming framework used to build applications suite for data analysis in the TOTEM experiment. It is built on top of the *Compact Muon Selenoid SoftWare (CMSSW)* which is a highly modularized set of programs being used by the scientists in *CMS* experiment in order to perform data processing. As *TOTEM* experiment cooperates with the *CMS* experiment it was a natural match to use common basis for application development.

TOTEM Offline Software extends the skeleton basis of *CMSSW* with modules performing TOTEM-specific tasks. It can be decomposed in the following domains:

- detector simulation
- geometry and *Alignment*
- reconstruction
- LHC optics parametrisation
- L1 *Trigger*
- data quality monitor and event display
- data management

A more detailed description of the TOTEM Offline Software domains is provided in [37]. Plenty of tools rely on the data, process it and store the intermediate results. An example data manipulation done by means of the Offline Software is the *Alignment* calculations. It consists of numerical searching for minimum of a function. Initial guesses are based on the *Roman Pot (RP)* positions drawn from the LHC Logging Databases and the data coming from the *DAQ*. When successfully calculated, *XML*, *Comma Separated Values (CSV)* or *ROOT* files are delivered by the scientists to other activities in the experiment which rely on the alignments.

Other sources

It is likely that there will be other sources of data to be considered, as the experiment is still in its early state. It has to be taken into account that the needs are constantly evolving and are getting more sophisticated.

3.2.2. Examples of the data routes

The following subsections present a few examples of data routes to be considered in designed system. Analysing different formats and access methods will give a more detailed view on the system purpose and will help to decide about architectural issues.

Conditions data from the LHC

LHC Logging Databases store measurements coming from systems like *Beam Position Monitor (BPM)* or *Beam Loss Monitor (BLM)*. A variety of LHC-related measurements hierarchy is accessible in two huge databases: *Measurements DataBase (MDB)* and *Logging DataBase (LDB)*. To the clients, data is accessible from Oracle Application Server via Java API providing high level functions for data extraction. After the data is acquired it is stored in computer's internal memory as a Java object. Certain data sent together with the object is not essential to be stored in the TOTEM Offline Database. Before putting such measurement into the database it is required to connect it to the certain *IOV* (discussed in section 4.3) and transform all these connections into the format of an *SQL* statement. Diagram showing the data route for discussed scenario is presented in Figure 3.5.

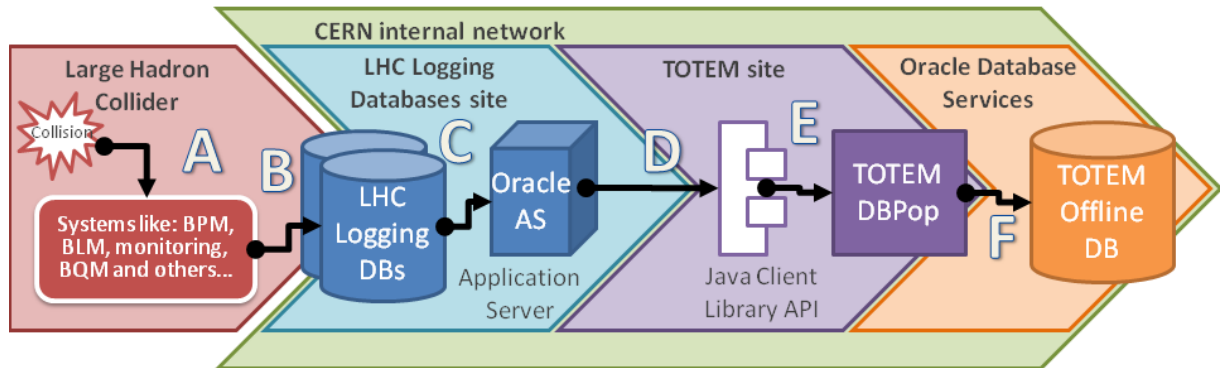


Figure 3.5. Data route of measurements coming from LHC Logging Databases.

We can distinguish different formats into which the data is being transformed:

- A - Raw detectors' readout
- B - SQL insert statements
- C - SQL select statements
- D - Serialized Java objects
- E - Java objects in the application internal memory
- F - SQL statements

The practical way of realising the data transfer from LHC Logging Databases is discussed in Section 4.2.1.

TOTEM DAQ data

In order for the data stored in the TOTEM Offline Database to be cohesive there should be an information about *TOTEM Runs* and specific events stored within them. *DAQ* data has to be parsed to extract this information. A *TOTEM Run* file is too large to be stored completely in the database and a file system is much better for storing it. Nonetheless a pointer to the place in the file system will be stored, which will ease the *DAQ* measurements access.

Information about the *TOTEM Run* are also available in other systems used by the *TOTEM* experiment. A shifter, after every data taking is responsible to collect all the information describing a Run in the RunLog[38] application. This application's database should be interfaced by the TOTEM DBPop application in order to retrieve relevant data.

Discussed data flow is presented graphically in Figure 3.6.

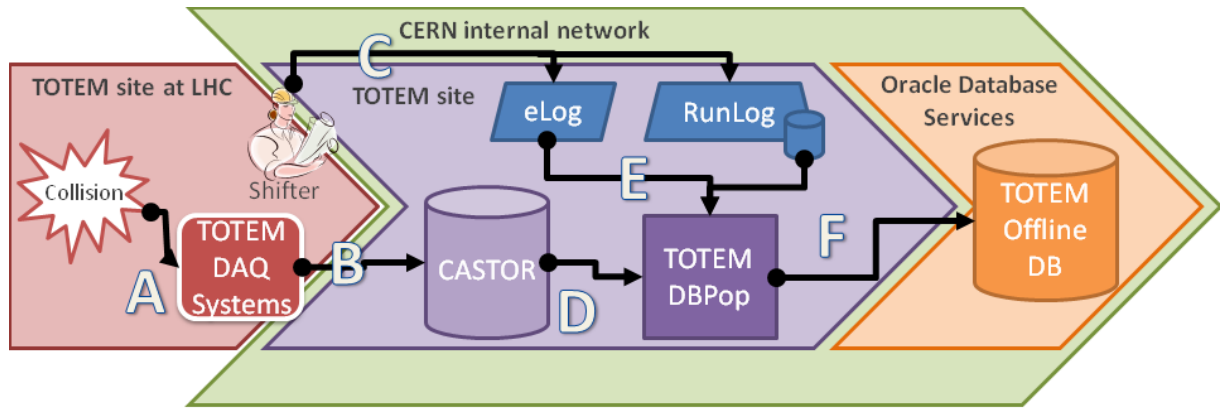


Figure 3.6. Data route of the DAQ-related data to the TOTEM Offline Database.

On the diagram, different formats into which the data is being transformed can be distinguished:

- A - Raw detectors' readout
- B - File transfer
- C - Information gathered via appropriate form by the Shifter
- D - File parsing
- E - SQL access
- F - SQL statements

It is still not precisely decided what kind of measurements will be stored in the database - designed application has to be ready for adjustments to upcoming needs.

TOTEM Offline software intermediate data

Information external to the events recored by detectors are necessary in order to properly process data of a *HEP* experiment. Such non-event data includes *conditions* (measurements during the data taking) and *calibrations* (metadata needed to correctly interpret measurements from a detector). A set of non-event data are stored locally in files in *XML* format. These data should be accessible by the Offline software either

directly (accessing the xml file) or from a persistent store - the TOTEM Offline Database.

Alignment (mentioned also in Section 3.2.1) is an example of an intermediate calculation which will be used by other pieces of software and has to be stored in the database. Alignment is calculated on the basis of Roman Pots positions acquired from the LHC Logging Databases via Timber interface as *XML*, *CSV* or *ROOT* files. After alignment calculation the output may be stored as *ROOT* or *XML* file.

Such file is read by the *DBPop* system and transformed into Java Object representation. Alignments have complex structure, because they describe precise positions of multi-part detectors. In order to resemble the measurement in the database it is crucial to bind the *Alignment* with *IOV* and place it into one of 3 tables designed to store alignments (database schema is discussed in section 4.3). Finally, a Java Object is being transformed into SQL statement to fill the database.

Discussed data flow is presented graphically in Figure 3.7.

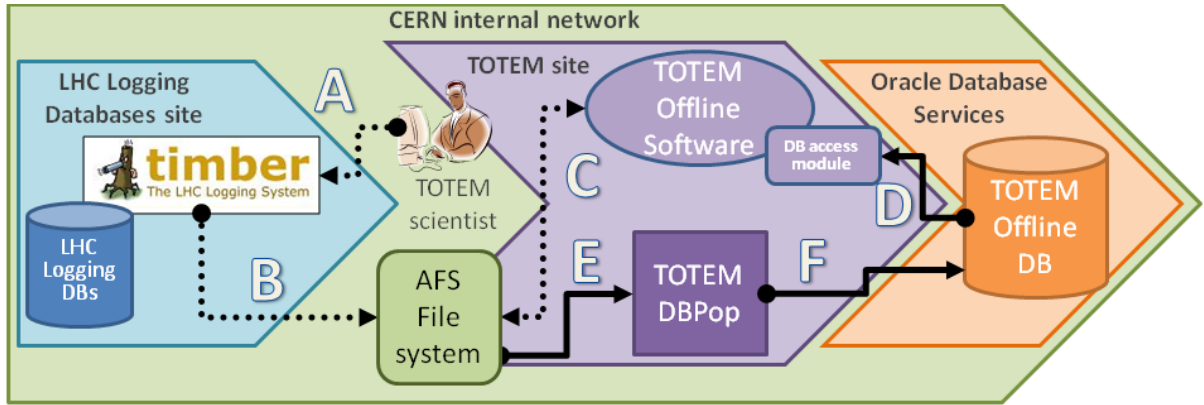


Figure 3.7. Data route for the intermediate calculations done by means of the TOTEM Offline Software.

In the diagram, different formats into which the data is being transformed can be distinguished. Intermittent lines show how the data circulates currently, while solid connections show how would it work after changing the processing model to use the database. Current data flow:

- A - scientist is using Timber *Graphical User Interface (GUI)* to get necessary data
- B - Timber extracts data from LHC Logging Databases and stores it in the file system (XML, CVS, ROOT files)
- C - finally, TOTEM Offline Software uses extracted data and saves the results back on the file system

Data flow using the TOTEM Offline Database:

- D - SQL statements, Offline Software accesses the database to get necessary data
- E - processed results are put to be aligned in the database by *DBPop*
- F - SQL statements

3.3. Problem statement

Offline database is an ongoing project at the TOTEM experiment and, since it still requires a lot of work, is not put into production yet. There are several reasons for that:

- database schema still not verified in production
- cumbersome, manual database filling
- range of different technologies used for database access
- lack of abstraction layer separating the database from improper usage

These reasons prohibit scientists to change the from processing model described in Section 3.1.1 to database-centric model described in section 3.2. Among others, there is a demand for creating a tool which will automate the data gathering mechanisms and aggregate all of them into a single provider. From this description a simple conclusion can be drawn - there is a need for an integration solution dealing with variety of technologies and high throughput of measurements data. When all the requirements were gathered, an idea of the *TOTEM DataBase POPulation System* (*DBPop*) system was introduced. TOTEM DBPop addresses the need of populating the TOTEM Offline Database with essential data.

3.3.1. Requirements

The aim

Main aim of the *DBPop* system is to be an intermediate layer to access the database which will provide:

- getting the demanded data from defined sources
- process it to be in appropriate format
- filter out unrelated information
- transform the data into format suitable for storing in the database

This middleware should ease and automate the process of filling the database with data essential from the TOTEM scientists point of view. Requirements regarding the DBPop system were evolving from a very raw state when the works started. After complete *System analysis* process the lists of requirements could be defined.

Functional requirements

Following list specifies the functional requirements to be met by the *DBPop* system:

- Acquiring general measurements data from LHC Logging Databases using high-level Java API
- Acquiring TOTEM *DAQ* data from a remote file system
- Gathering all intermediate (alignments, calibrations, etc.) data generated while the measurements analysis
- Data filtering of unessential information
- Formatting data to suit the TOTEM Offline Database schema

- Provide user interface to govern the data transferring process
- Provide automatic data transferring facilities

In order to have a clear understanding of what *DBPop* system is it might be useful to pinpoint what *DBPop* is not. *DBPop* is not used for the database monitoring or accessing values stored in it. Its purpose is to aggregate all of the data sources, represent them as a single data manipulator and coordinate the data transfer. Database monitoring is performed by a separate tool - TOTODAM [39]. Providing database abstraction layer for client application is the focus of other ongoing works in the *TOTEM* experiment.

Non-functional requirements

Besides the knowledge of what we actually expect from the system behaviour, it is crucial to define requirements regarding non-functional aspects. These reflect the quality of how the functional requirements will be provided. We demand that the system will:

- provide means of efficient data transfer
- be easy to use by the TOTEM scientists
- be easy to extend to add new data sources or functionalities
- scale with the problem growth, in order not to redesign the solution
- be able to clusterize
- generate reports

The choice of implementation technology has been made - it should be a Java-based solution. Java is a mature platform for building reliable, robust and effective enterprise solutions. Vast range of Java technologies gives the possibility to suit the means to the actual needs. Java was considered from the first moment, since it will seamlessly integrate with one of the sources, the LHC Logging Databases Java API (discussed in Section 3.2.1), and is the right choice when it comes to fast and convenient database manipulation by means of *Object-Relational Mapping (ORM)* solutions. It is better to use *ORM* approach instead of direct, database-specific language since it provides certain advantages, like connection pooling and caching (discussed in more detail in Section 2.2.5). The last argument for choosing Java as the implementation platform is the author's experience in building application in this technology.

3.3.2. Use-cases definition

The diagram in Figure 3.8 presents the use-cases for the *DBPop* solution.

Trigger data-taking interactively

The database population process might be launched by the shifter or system administrator in interactive (blocking, synchronous) mode. This option might be useful to be used in scripts, because it assures the data will be already populated till the end of the system invocation.

Trigger data taking in batch mode

The shifter launches data taking to be run in background. For large chunks of data the batch population mode is more suitable because of non-blocking invocations and full speed, concurrent modules functioning

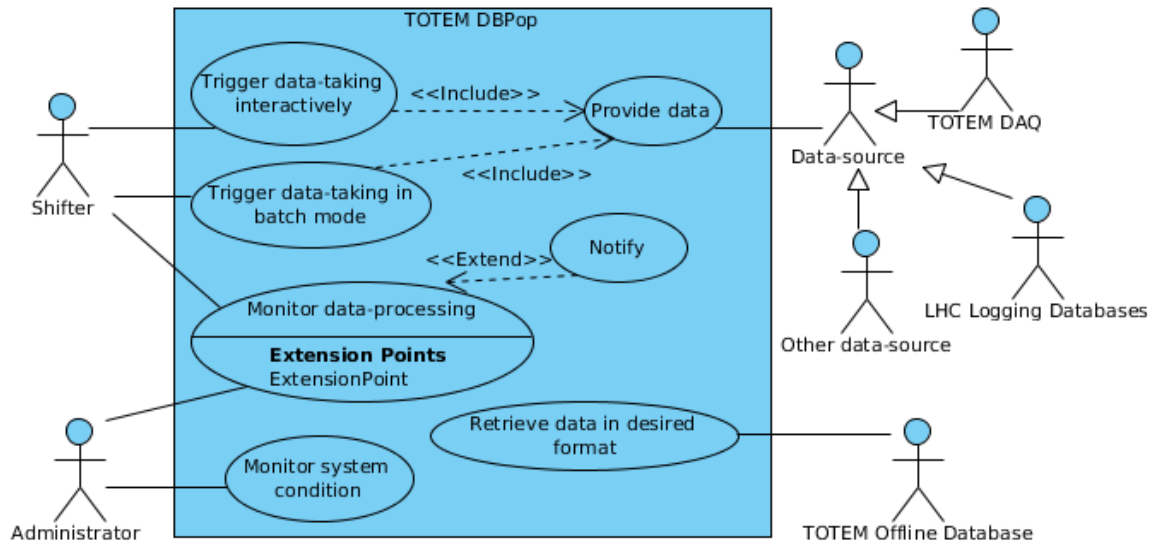


Figure 3.8. Use-cases diagram for the TOTEM DBPop system.

Monitor data processing

System user might ask for the data-transferring process status to monitor the progress. System informs about the data-population jobs' statuses

Monitor system condition

Contents of the system should be easily monitorable. System administrator watches for system performance and potential problems while data taking.

Provide data

Data sources, which are external systems, are used to acquire the desired data. The designed system is providing right means of accessing different sources when the data taking process is triggered. *DBPop*, based on user wishes, chooses right data source to be used.

Retrieve data in desired format

TOTEM Offline Database is an external entity which receives aggregated data from the system.

3.3.3. Solution proposal - DBPop

TOTEM DataBase POPulation System (DBPop) is a tool for performing data transfer from distinct sources to the TOTEM Offline Database. It deals with access methods to different sources, relieving scientists from this task. Its back-end works as a daemon, together with a network of queues which might be distributed across separate JMS brokers. Front-end is a command-line tool which enables user to trigger and monitor the data transmission process.

3.4. Architecture

Architecture divagations begin with an abstract vision of designed system. Afterwards, more detailed insight into the system will be presented by stepping down with the abstraction level. Section ends with a discussion of alternative ways to implement solution.

3.4.1. Basic view

High level of abstraction would define the considered system as a funnel, gathering all the data in one data sink. This concept is presented in the Figure 3.9. What is

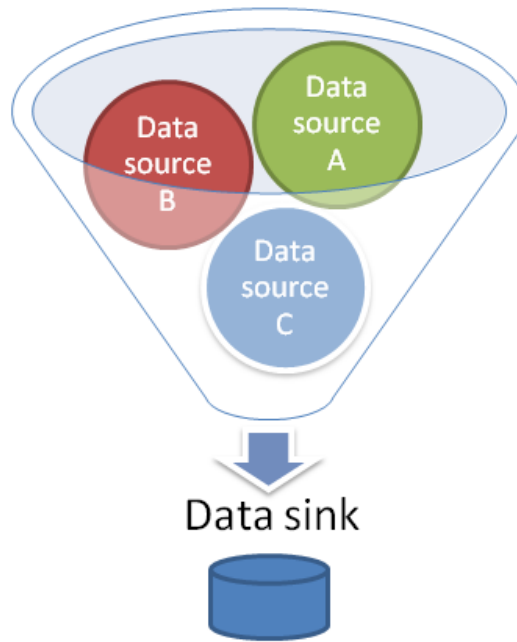


Figure 3.9. Funnel model of the designed system. All of the data comes by means of a single valve into the data sink.

important to achieve is a complete abstraction of what kind of systems are delivering the data behind the scenes. *DBPop* should be the only tool for accessing all of the data sources.

3.4.2. Technology-agnostic approach

An idea of how to accomplish functional and non-functional requirements with respect to the current environmental conditions leads to a first solution proposal. Figure 3.10 shows the proposed architecture in a technology-agnostic way. Here we will walk through the schema and discuss each of the working blocks.

Data source adapters

There is a set of distinct data sources that provide information of interest. Each of the sources might feature different access methods and expose variety of interfaces. Examples of potential sources might be: a file system, Web Service, *JMS* queue, high

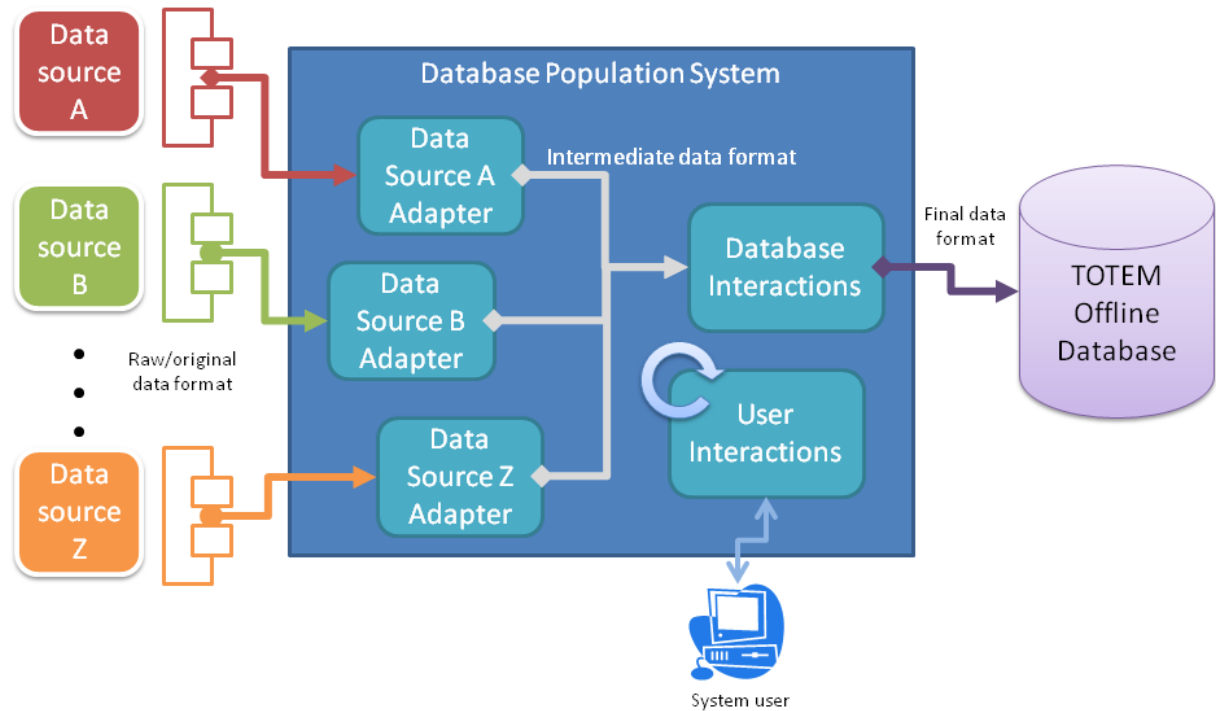


Figure 3.10. High-level view on the designed system with basic division into modules.

level library providing certain API or a database with *SQL* access. For every data source we have to consider the data exchange model. While designing a data-source adapter one has to take into account the following aspects regarding the access method to the data-source:

- does it support publisher/subscriber
- do we have to use polling
- does it support synchronous, blocking invocations?
- does it work in batch mode?
- what is the *Message Exchange Pattern (MEP)* while communicating with it
 - in-only
 - in-out
 - robust in-only

For instance, a file system access does not implement any publish-subscribe policies, since the user does not have the possibility to register interest for some kind of event. In order to acquire data from it the polling method has to be used. JMS queue is the representant of a two-way communication with the possibility to implement the publish-subscribe paradigm. An external system providing an *API* will probably support synchronous, blocking invocation or might use asynchronous callbacks. It is important to understand in what way does the external system provide data.

For each potential data source it is required to have means by which to communicate with it with respect to the data exchange model mentioned earlier. For the right interaction with a data source an adapter has to be built. An adapter is responsible for:

- performing given interactions with the data source
- hiding access protocol complexity from perspective of the system
- performing necessary data conversion to the intermediate shape
- provide easy to use interface for interactions with other parts of the system

These requirements give the general overview of a component fulfilling these tasks. The output of the adapter should be an intermediate format, ready to be processed further by the system.

User interactions

The way adapters work should be controlled (supervised) and monitored by the user. User interactions should provide necessary means to map user wishes on the adapters steering. The status of system functioning should be provided in an integrated manner to the user. The way of transporting data from adapter to the data sink has to be done in effective and reliable manner. User interactions module is the central part of the system, where integration of all the data takes place, in order for it to become the database content.

Database interactions

A database interactions module is responsible for communicating with the actual data sink, which in this case is the database. It is responsible for:

- establishing and effectively maintaining database connection
- hiding access method details and protocol complexity from the user
- perform necessary format adoption
- provide optimization for the data sink access like pooling and resources caching

Nonetheless system should be built in a way allowing to add new data sinks. Adopting intermediate data format to any output format should be an easy task.

3.4.3. Architecture using ESB approach

Using *Enterprise Service Bus (ESB)* as the skeleton of an application (or a set of applications within an enterprise) has several advantages [40]:

Standardization

Without a unifying platform, enterprise gets a divergence of integration methods which leads to inconsistency and higher cost of management and change

Loose coupling

ESB promotes:

- physical decoupling - by making use of message passing mechanisms (e.g. *JMS*) instead of direct network connections (e.g. *HyperText Transfer Protocol (HTTP)*)
- semantic decoupling - use of message transformation so that services are exposed in a system-neutral format

Scalability and reliability

By physical decoupling *ESB* achieves scalability advantages such as high-availability, fault-tolerance and load balancing

Routing and mediation

Message routing supports scalability and fault-tolerance

Complex *MEP* support

An ESB supports complex MEPs such as asynchronous one-way messaging and to multiple subscribers using topic-based messaging

One can easily notice that the considered scenario in the *TOTEM* experiment at *CERN* is actually an integration problem, which can easily be tamed by the *ESB*. The reasons, why an ESB is an adequate solution are:

- variety of technologies, protocols and access methods
- preferred asynchronous nature of database population with respect to performance

Figure 3.11 shows the proposed architecture using Enterprise Service Bus in a vendor-independent way. *DBPop* application is a set of independent modules, which

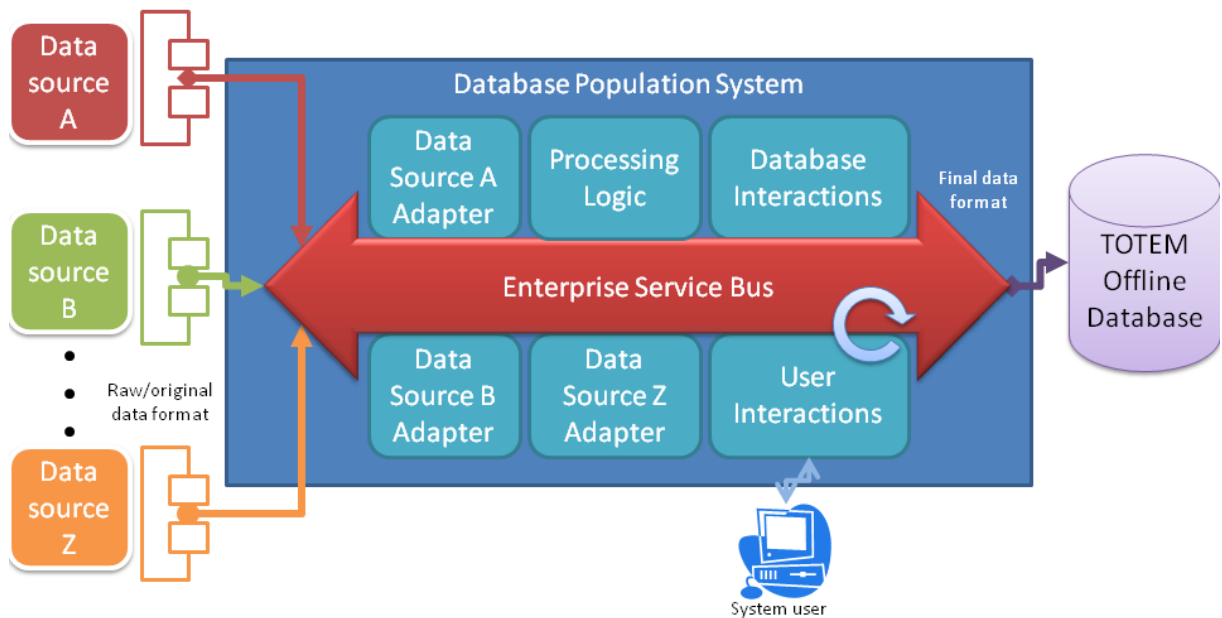


Figure 3.11. System architecture using Enterprise Service Bus as its backbone.

communicate by means of a messaging and routing framework. Each of the components should be able to work separately without the knowledge of the other parts of the system. Coarse-grained, single responsibility modules should conform to the *SOA* principles (Section 2.2.1), which effects in reusability and flexibility. It is expected, that the system will deliver the highest efficiency, since it is composed of independent, concurrently working pieces. The tests described in Chapter 5 will reveal the genuineness of this thesis.

Processing Logic is the brain of the system. It is responsible for orchestrating other modules and managing the application status. Building the system in asynchronous manner simplifies the whole architecture. Modules are simple, easy to maintain and focused on one purpose. All the connection logic is left to be configured via ESB mediation (routing).

The application has to be ready for including additional working blocks. Extensibility of the system will be achieved by adding modules responsible for data collecting from new sources. In this way we achieve openness for extension and closure for modification.

In the next chapter (Section 4.1) it will be shown how the above (vendor independent) vision maps onto the open-source *Enterprise Service Bus (ESB)* - Apache ServiceMix 4.x.

3.4.4. Alternative ways to achieve solution

The *ESB*-approach to resolve the problem is of course only one potential way to implement the system. As the *Enterprise Service Bus (ESB)* conforming paradigm suits very well to the purpose of the whole system it was chosen as the application skeleton. Author admits that using *ESB* should be considered with caution, since it does not always have to be beneficial. Gathered experiences, together with pitfalls are described in Chapter 5.

Monolitical application

Another option would be to build the system as a standalone application, without using any kind of integration middleware. Monolitical, not service-oriented application is not versatile across different data transporting technologies, since access to every different protocol has to be coded in vendor-specific way. Building application this way would cause:

- less extendability
- more error prone
- worse efficiency because of synchronous nature of methods invocation

On the other hand, to implement such solution one does not have to be aware of sophisticated component technologies discussed in previous sections.

Set of applications dealing with data-sources separately

System could also be designed as a set of dedicated programs, that could provide the means of accessing data from distinct sources. One governing main application could be used to orchestrate the others. Such approach would imply code duplication for the database access and problems with maintaining common data format. Proposed architecture is similar to using *ESB*, but leached of all of its advantages.

3.5. Chapter summary

This chapter begun with more detailed insight into the data processing methods in the *TOTEM* experiment at *CERN*. Current state of art was described together with its evaluation. Data sources, data life-cycles together with the data routes were discussed in order to fully understand the deployment scenario. *TOTEM* Offline Database project was introduced as well as the motivation behind the *DBPop* system construction. Next sections described the requirements for constructed system. The system model was analyzed in a top-down manner. The description begun with use-cases definition, followed by basic view on the system architecture through

technology-agnostic vision, ending with presentation of the *ESB*-based architecture. Basic working blocks were given detailed description conforming defined requirements. At the end, alternatives for system implementation were given an overview.

Implementation issues

This chapter's aim is to introduce the reader to the most significant implementation issues which are worth describing in order to fully understand the working mechanisms behind the scenes of the *TOTEM DataBase POPulation System (DBPop)*. At first, *DBPop* architecture using ServiceMix is presented. This section gives an insight into how the facilities of ServiceMix are exploited while building ESB-compliant systems. All of the system ingredients are given a detailed overview, together with sequence diagrams and code samples. Next, the TOTEM Offline Database is discussed. The reader will get familiar with the database schema and the way it was implemented at CERN Oracle Services. Afterwards, the development environment is described, since it takes advantage of many original and innovative approaches. Finally, the usage and working mechanisms of *DBPop Command-Line Interface (CLI)* are explained.

4.1. Architecture using ServiceMix 4.x

Sections in previous chapter were explaining the system architecture going from the most abstract level (Section 3.4.1) through conceptual component-based solution (Section 3.4.2) to the *Enterprise Service Bus (ESB)*-based architecture (Section 3.4.3). In Section 2.4.1 an open-source *ESB* implementation - Apache ServiceMix 4.x (FUSE ESB 4) was presented together with all of its subsystems realising the ESB facilities. Here the reader will find out how ESB concepts map onto a real product. It will be shown in great detail how the basic requirements (Section 2.2.1) regarding *ESB*:

- mediation
- routing
- address and protocol virtualization

are realized in this product.

4.1.1. Basic concepts

Before digging into the *DBPop* working mechanisms details, it is crucial to introduce a couple of basic concepts. The names like: *Data Population Request (DPR)* and *Data Container* have already appeared in previous sections, but were not defined yet.

Both classes implement the *Serializable* interface and are used to exchange information between the main components of the *DBPop* system. Here is the proper place to explain them.

Data Population Request

A *Data Population Request* (*DPR*) is a very simple object containing information about the measurements to be populated, and the range of time to be considered. It is a classic *POJO*, containing the following fields:

```
public class DataPopulationRequest implements Serializable {
    private String measurement;           // identifies uniquely populated measurement
    private Timestamp startTime;          // beginning of the timespan
    private Timestamp endTime;            // end of the timespan
    private Exception failureCause;        // in case of failure while crossing the Camel route
    private String dbpopJobId;             // job identification
    private Integer dbpopSeqId;            // number of the request within the job
}
```

Listing 4.1. The DataPopulationRequest class fields.

DPR objects are used for administrative purposes and circulate together with the actual data. A *DPR* is included in the Data Container class as a metadata.

Data Container

DataContainer class is a conveyor for transferred data. It is as simple as:

```
public class DataContainer implements Serializable {
    private DataPopulationRequest request;           // metadata
    private HashMap<Variable, VectornumericElements> mappings; // mappings, if present
    private HashMap<Variable, TimeseriesDataSet> data; // measurement data
}
```

Listing 4.2. The DataContainer class fields.

DataContainer contains the data itself as well as the metadata: a *DPR* containing essential information regarding this data package and mappings, if they are present.

4.1.2. Architecture

Figure 4.1 presents the high-level view on *DBPop* architecture using Apache ServiceMix 4.x. In order to see how ServiceMix fulfils the *ESB* contract (Section 2.2.1) it is recommended to compare and match the elements in Figures 3.11 and 4.1. In the diagram distinguishable from the others are:

- *OSGi* Services
- *JMS* Queues
- Camel routes
- External systems (data sources and data sink)
- *Command-Line Interface* (*CLI*) for user interaction

While reading the following sections it is recommended to have this picture in mind in order to have a clear view of the system. All of the components will be given a detailed overview throughout the following sections.

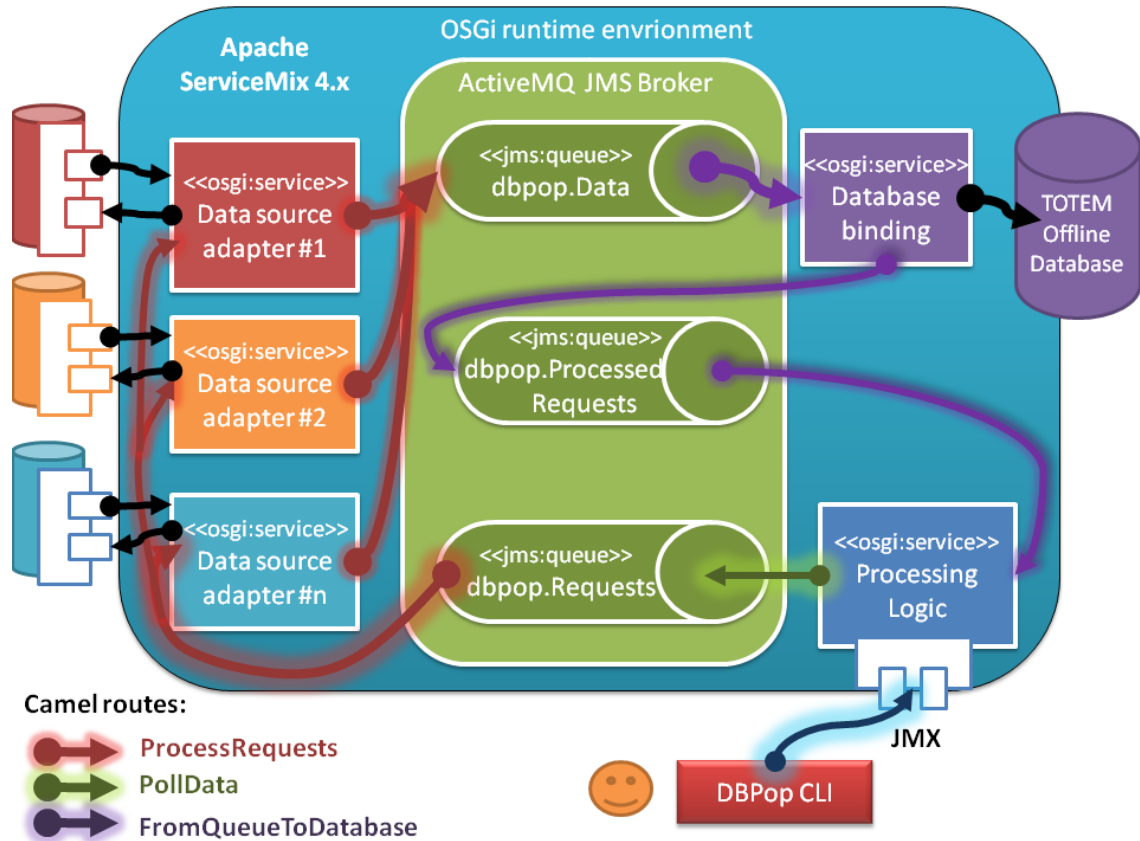


Figure 4.1. DBPop architecture using Apache ServiceMix 4.x (FUSE ESB 4).

4.1.3. Working mechanism of DBPop

The system user triggers data population by issuing it from the *DBPop CLI*. The *ProcessingLogic* instance is generating *DPRs* and then places them on the *dbpop.Requests* queue. Camel route called *ProcessRequests* is responsible for delivering the *DPRs* to the *data source adapters* components. After successful download of desired data, Camel transfers the generated *DataContainer* object on the *dbpop.Data* queue, ready to be carried by another Camel route - *FromQueueToDatabase*. This route transports the *DataContainer* objects from the Data queue to the *DatabaseBinding* component in order for the data to be stored in the database. After successful database population the *FromQueueToDatabase* route delivers message status to the *dbpop.ProcessedRequests* queue, which is digested back by the *ProcessingLogic* component.

Above, high-level description of the system operation will be clarified in the next sections. Below, all of the most important system ingredients are given a detailed overview. At first, the *OSGi* services are described, which are the basic functional blocks of the *DBPop* system. Next, the *JMS* queues and their versatile usage is described. At the end of the section, the Camel routes are described, which are sticking the components together.

4.2. DBPop components

DBPop is built of a set of modules. In the *OSGi* world, a module realisation is a *service* (Section 2.2.1). *OSGi service* is an independent, coarse-grained piece of reusable component identified and discoverable by its properties (e.g. an interface). They are basically Java objects which are registered and published in the *OSGi Service Registry*. Worth noticing is that each of the modules is a single-responsibility *POJO*, thanks to using Spring and applying principles of *SOA*, which makes the modules' code simple. Each of the services is exposing certain set of actions and is identified in the *Service Registry* by its interface, so one object (implementing many interfaces) can be used for different purposes. Any *service* can use functionalities offered by other *services* accessible in the *registry*. Using *OSGi* facilities provides the means to realize the find-bind-execute approach - feature, which is desired by the *SOA* paradigm (Section 2.2.1).

How does it look like in code? It is very simple, since *DBPop* takes advantage of the Spring facilities integrated in ServiceMix. Spring Dynamic Modules¹ is a very handy package used in order for the modules code to be simple and vendor-independent. One configures all the service's relationships by means of Spring *XML* with *OSGi* extensions, which is similar to standard Spring beans configuration. Spring Dynamic Modules² hides the complexity and relieves the programmer from using *OSGi API* directly. The following example configuration file is the *spring-osgi.xml* file for the *ProcessingLogic* component. It is as simple as:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:osgi="http://www.springframework.org/schema/osgi"
  xmlns:osgix="http://www.springframework.org/schema/osgi-compendium"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/osgi
    http://www.springframework.org/schema/osgi/spring-osgi.xsd
    http://www.springframework.org/schema/osgi-compendium
    http://www.springframework.org/schema/osgi-compendium/spring-osgi-compendium.xsd">

  <!-- In order to retrieve properties stored in SERVICE_MIX_HOME/etc -->
  <osgix:cm-properties id="dbpop-conf" persistent-id="ch.cern.totem.dbpop"/>
  <context:property-placeholder properties-ref="dbpop-conf"/>

  <!-- This declaration find a service of LHCApiPoller interface in the
    registry and binds it to lhcApiPoller bean id -->
  <osgi:reference id="lhcApiPoller" bean-name="lhcApiPoller"
    interface="ch.cern.totem.dbpop.lhcApiPoller.LHCApiPoller" />

  <osgi:reference id="databaseBinding" bean-name="databaseBinding"
    interface="ch.cern.totem.dbpop.databaseBinding.DatabaseBinding" />

  <osgi:reference id="lhcLoggingMeasurements" bean-name="lhcLoggingMeasurements"
    interface="ch.cern.totem.dbpop.common.LHCLoggingMeasurements" />
```

1. <http://www.springsource.org/osgi>

2. The specification of Blueprint Container in *OSGi Compendium* from Release 4.2 is the *DI* framework for *OSGi*. Spring Dynamic Modules is the *Reference Implementation (RI)* for Blueprint specification

```

<!-- This declaration exposes the bean processingLogic in the registry
      which will be identified by ProcessingLogicMBean interface -->
<osgi:service id="processingLogicService" ref="processingLogic"
      interface="ch.cern.totem.dbpop.processingLogic.ProcessingLogicMBean"/>
</beans>

```

Listing 4.3. Example Spring Dynamic Modules for OSGi configuration file.

Running *DBPop* in the *OSGi* environment delivers certain qualities, like: hot-deploy of the system parts (*OSGi bundles*) or independent modules governance (start, stop, update, configuration refresh) without the application restart. The *DBPop* system is built from single-responsibility *OSGi services* cooperating together to achieve the aim. The proceeding subsections discuss all of the *DBPop* modules separately.

4.2.1. LHC API Poller

The LHC API Poller is a module responsible for interacting with the LHC Logging Databases (discussed in Section 3.2.1). LHC API Poller is a representant of a data source adapter depicted in figure 4.1. It is built on top of the *Logging Data Extractor Client (LDEC)* Java API, created and maintained by the Beams department at CERN. The LHC API Poller component is a poller abstraction built upon the API to provide any data from LHC Logging Databases to be accessible by *DBPop*. In order for the LHC API Poller component to be able to use the Java API it was required to register *DBPop* in the Beams department as one of the applications using the data extractions mechanisms. The component implements a simple interface which exposes the functionalities to:

- acquire data based on the *DPRs* generated by the ProcessingLogic component
- generate *DPRs* or timeperiods objects in case of *LHC Fill*-based data population

Sequence diagram in Figure 4.2 presents the chain of actions that have to be taken in order to successfully acquire data by means of the Java API. First, the administrative objects have to be created. including the most important:

VariableController

is used to retrieve correct measurement “names”, called variables. Using this object one creates sets of variables for which the data will be gathered.

FillDataController

allows to retrieve information related to the *LHC Fill*. Based on acquired information (e.g. duration of the fill, timespan of STABLE beam mode) it is easier to get needed measurements data.

TimeseriesDataController

is responsible for the actual data extraction, based on previously acquired variables and timestamps.

Afterwards, component is ready to receive jobs by means of the *DPRs* (disussed in Section 4.1.1). One can easily configure the LHC API Poller module to acquire any measurements stored in the LHC Logging Databases by means of an *XML* configu-

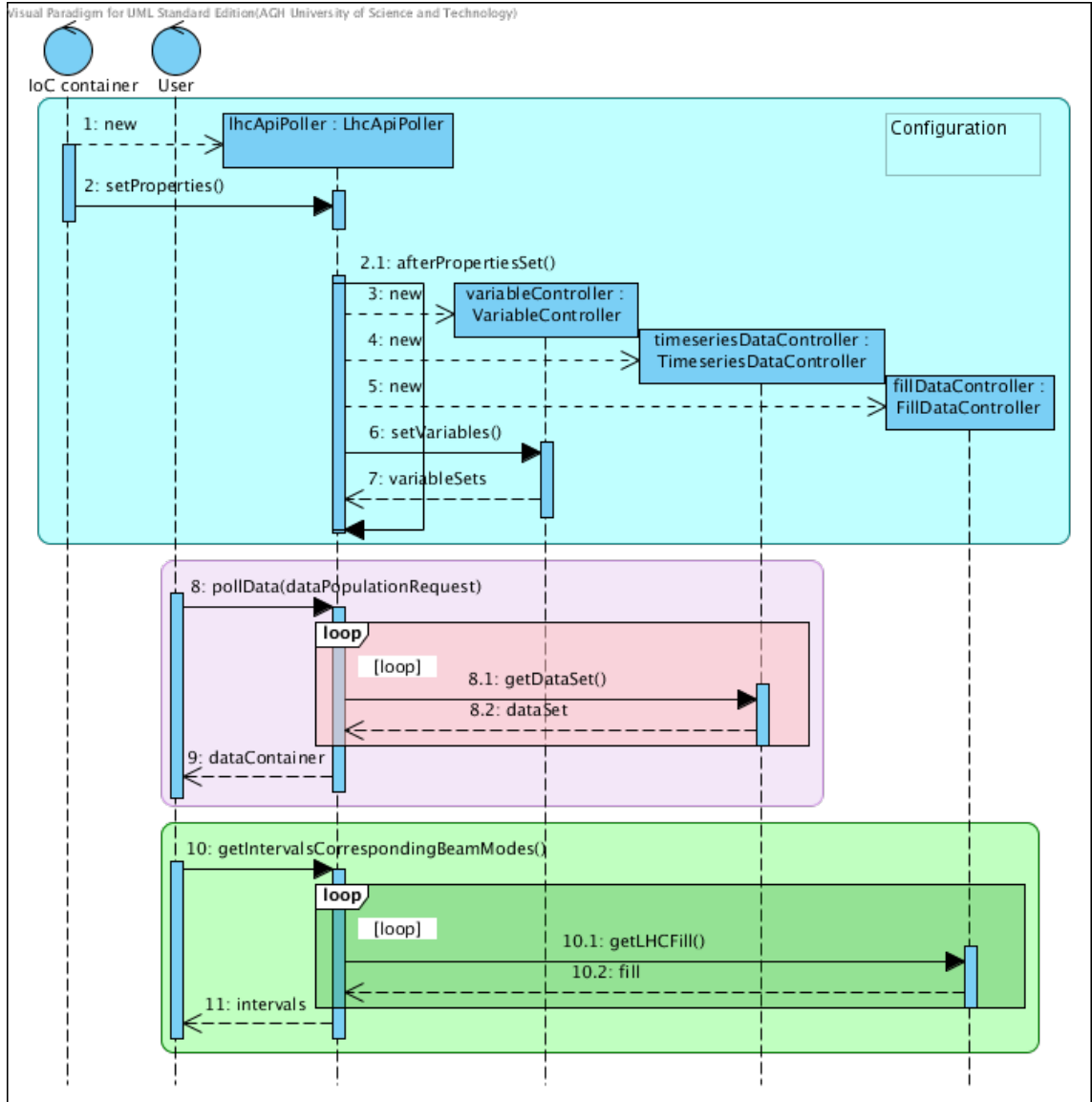


Figure 4.2. Sequence diagram of the LHC API Poller module of DBPop.

ration file. It lists the measurements names and aggregates them in groups for easier maintenance. The following listing presents a piece of the configuration file:

```

<?xml version="1.0" encoding="UTF-8"?>
<lhcLoggingMeasurements>
  <group name="BPM" mappings="true">
    <variable>LHC.BOFSU:POSITIONS_H</variable>
    <variable>LHC.BOFSU:POS_ERR_H</variable>
    <variable>LHC.BOFSU:BPM_STATUS_H</variable>
    <variable>LHC.BOFSU:POSITIONS_V</variable>
    <variable>LHC.BOFSU:POS_ERR_V</variable>
    <variable>LHC.BOFSU:BPM_STATUS_V</variable>
  </group>
  ...
</lhcLoggingMeasurements>
  
```

Listing 4.4. Example XML configuration file for the LHC Logging measurements.

The same names used in the configuration file are used by other components of the *DBPop* system, including *DBPop CLI* discussed in Section 4.5

4.2.2. Database Binding

The Database Binding component is responsible for interacting with the TOTEM Offline Database, as illustrated in Figure 4.1. It encapsulates the complexity of the mechanisms for:

- connecting to the database
- data format transformation
- filtering of unessential data
- dealing with exceptional cases
- optimizing write performance
- realizing transactional processes

To the other modules it is visible by means of a simple interface with only one method:

```
public interface DatabaseBinding {
    DataPopulationRequest saveData(DataContainer dataContainer)
        throws DatabaseBindingException;
}
```

Listing 4.5. The DatabaseBinding interface.

Basing on the *DataContainer* object (discussed in 4.1.1) passed to this method, *DatabaseBinding* is able to determine a proper way of storing the data in the TOTEM Offline Database. *DatabaseBinding* component uses standardized *Java Persistence API (JPA)*, which makes the module's code simple and vendor-independent. Current *ORM* provider working behind the *JPA* facade is *Hibernate*, which shows its presence only through the configuration files. All of the module's dependencies are configured via *Spring* and *Spring Dynamic Modules for OSGi* in a similar way to the Listing 4.3.

Sequence diagram in Figure 4.3 presents the life-cycle of the *DatabaseBinding* component. At first, the administrative objects (like *DataSource*, *EntityManagerFactory* and *TransactionTemplate*, not all shown in the diagram for clarity) need to be created. It is done seamlessly by the *Spring* container on behalf of the programmer. The container delivers references to these objects which are configured and ready to be used by *DatabaseBinding*. Using *IoC* container prevents the module to contain boiler-plate, configuration code to appear in its content.

Figure 4.3 shows the interactions of objects which occur during *saveData* method invocation. The following piece of pseudocode is a simplified algorithm for data persisting:

```
EntityManager em = entityManagerFactory.createEntityManager();
try {
    em.getTransaction().begin();
    int howMany = 0;
    // Prepare single measurement to be written to the database:
    // 1. Extract significant data
    // 2. Set the data references to other tables in the database
    em.persist(entity);

    if (++howMany % databaseFlushChunk == 0) {
        em.flush();
    }
}
```

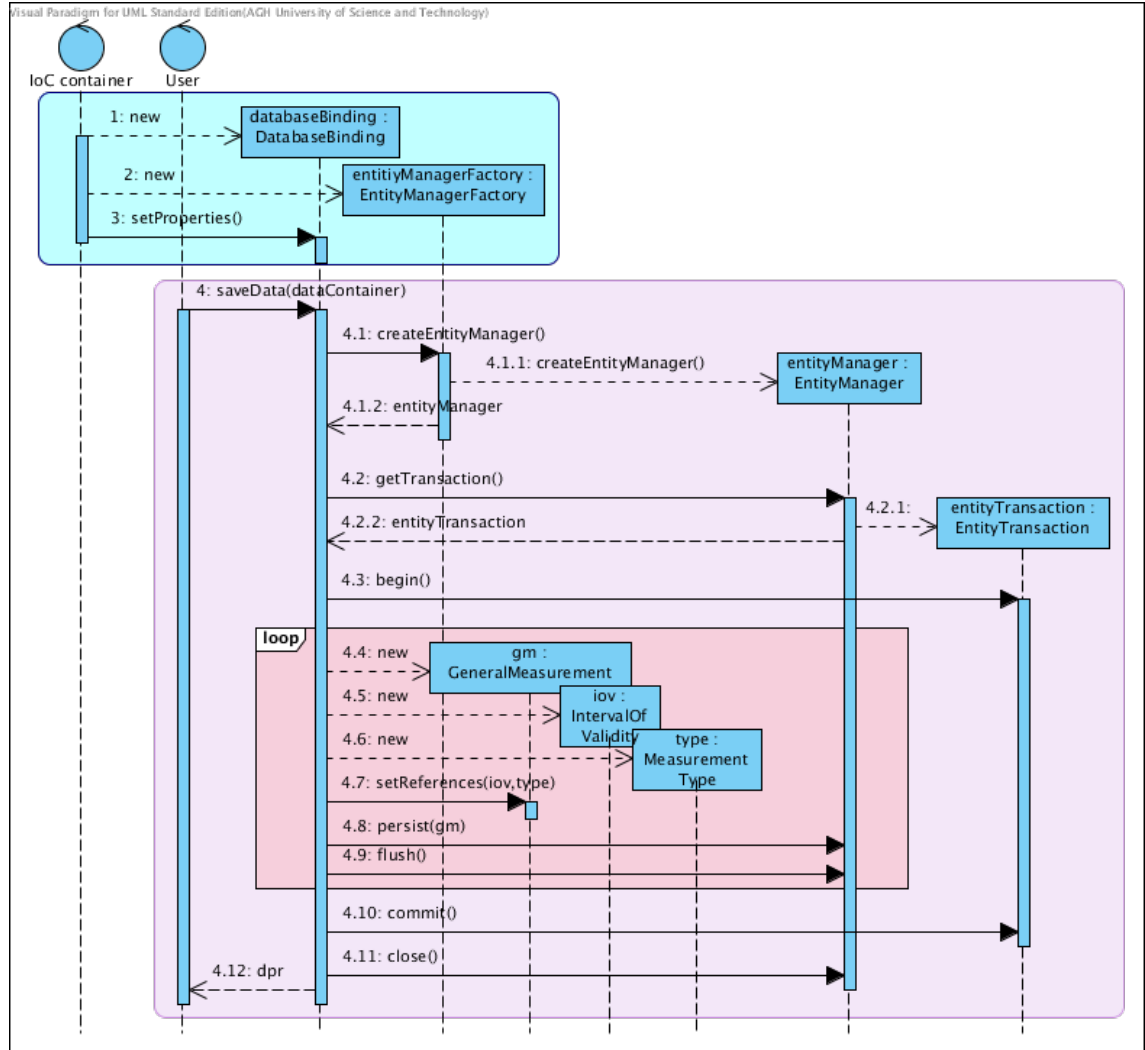


Figure 4.3. Sequence diagram of the DatabaseBinding module.

```

    em.clear();
  }
  em.getTransaction().commit();
} catch (ExceptionHierarchy e) {
  // Deal with exceptional conditions
}

```

Listing 4.6. General overview of the data writing mechanism in DatabaseBinding component.

The writing is organized into chunks of the same size as the bulk size in Hibernate configuration (`hibernate.jdbc.batch.size`). The `EntityManager` is forced to flush and clear its cache periodically to improve writing performance. Besides the *DBPop* side configuration, the database itself needs to be adjusted to work on the highest pace of data throughput, which is discussed in Section 4.3.

The `DatabaseBinding` component is the most important module of the *DBPop* system when it comes to performance. Proper *SQL* statements generation together with using performant storing techniques should provide the highest data writing

throughput. The Hibernate framework working behind the scenes has to be properly configured to achieve satisfactory working conditions. Main purpose of the *DBPop* system is to store the data in the database. There are only few circumstances in which the component has to read the data from the database. This is why the Hibernate's configuration was optimized for writing. Important aspects of configuration that need to be taken into account include:

- batch (bulk) inserts
- transactional processing
- statement caching
- turning off the secondary level cache facilities
- mitigating database constraints performance issues
- connections pooling

Section 5.1.2 in the Tests chapter lists the most important configuration of the Hibernate framework used in the tests scenarios.

The working mechanisms of the DatabaseBinding component are likely to be substituted in future releases of *DBPop*. Pure *JDBC* usage might be more beneficial, because of performance issues and greater control over the storing process. Hibernate was chosen at the beginning of the project, because of its great versatility and vendor-independence. Since *CERN* is using only Oracle databases, it might be considered to use proprietary solutions for database interactions, like direct usage of the Oracle JDBC driver.

4.2.3. Processing Logic

The ProcessingLogic module is the heart of the *DBPop* system (depicted in Figure 4.1) The following list presents the main responsibilities of the discussed component:

- exposing system functionality to the user interface
- registering, tracking and managing the data population jobs
- generating *DPRs* and distributing them among the working nodes
- configuring components behaviour
- generating reports

ProcessingLogic manages other modules and orchestrates their actions by distributing jobs objects - the *DPRs*. The ProcessingLogic module is also an entry point for user interactions with the system. For now it is being steered by means of the *DBPop CLI*, but can be easily extended to receive interactions from any kind of interface, including *GUI*. Listing 4.7 presents the most crucial methods of the ProcessingLogicMBean interface.

```
public interface ProcessingLogicMBean {

    // invoked regularly to check for new data
    void populateData();

    // interactive (blocking) data population
    String populateDataWithinTimestampsSynchronously(
        Timestamp startTime,
        Timestamp endTime,
        Set<String> measurements
    ) throws ProcessingLogicException;

    // batch (asynchronous) data population
```

```

String populateDataWithinTimestampsAsynchronously(Timestamp startTime,
    Timestamp endTime, Set<String> measurements)
    throws ProcessingLogicException;

// interactive (blocking) data population for LHC fill
String populateDataForFillSynchronously(int fillNumber,
    Set<BeamModeValue> beamModeValues, Set<String> measurements)
    throws ProcessingLogicException;

// batch (asynchronous) data population for LHC fill
String populateDataForFillAsynchronously(Integer fillNumber,
    Set<BeamModeValue> beamModeValuesSet, Set<String> measurements)
    throws ProcessingLogicException;

// get job status
String queryRequestStatus(String jobId) throws ProcessingLogicException;

// turn on/off the automatic data population
String enableAutomaticDataTaking(boolean enable,
    Set<BeamModeValue> beamModeValuesSet)
    throws ProcessingLogicException;

// turn on/off the queue-based data transfer infrastructure
String enableAsynchronousInfrastructure(boolean enable)
    throws ProcessingLogicException;

// retrieve valid measurements names
Set<String> getValidMeasurements() throws ProcessingLogicException;
}

```

Listing 4.7. The ProcessingLogicMBean interface main methods.

There is a special module, the ManagementService, exposing functionality of the ProcessingLogic as a WebService by means of the Apache CXF integrated in ServiceMix.

Figure 4.4 shows the actions sequence for a few common use cases:

- initialization, done by the *IoC* container (Spring)
- asynchronous data population triggered from the *User Interface (UI)*
- receiving job update message
- querying for job status from the *UI*

At first, administrative objects of the *JMS API* are created “by hand” (Spring *JMS* template is not being used because of performance issues³). The ProcessingLogicMBean is registered in the *JMX* MBeanServer, to be easily discovered by any *JMX* compliant management console (e.g. *jconsole*).

ProcessingLogic uses auxiliary objects: LHCLoggingMeasurements for proper *DPRs* generation and JobRegistry for jobs management. Reports are delivered to the system user by an e-mail realized by means of Spring Mail⁴, which hides the complexity of Java Mail *API*⁵.

Improvements of the ProcessingLogic module are foreseen in the future releases of *DBPop*. Implementing a persistent storage for the JobRegistry object would provide the statelessness of the ProcessingLogic component, which is now the only one not conforming fully to the *SOA* principles. Batch jobs are currently supported by means

3. <http://activemq.apache.org/jmstemplate-gotchas.html>

4. <http://static.springsource.org/spring/docs/3.0.x/reference/mail.html>

5. <http://www.oracle.com/technetwork/java/javamail/index.html>

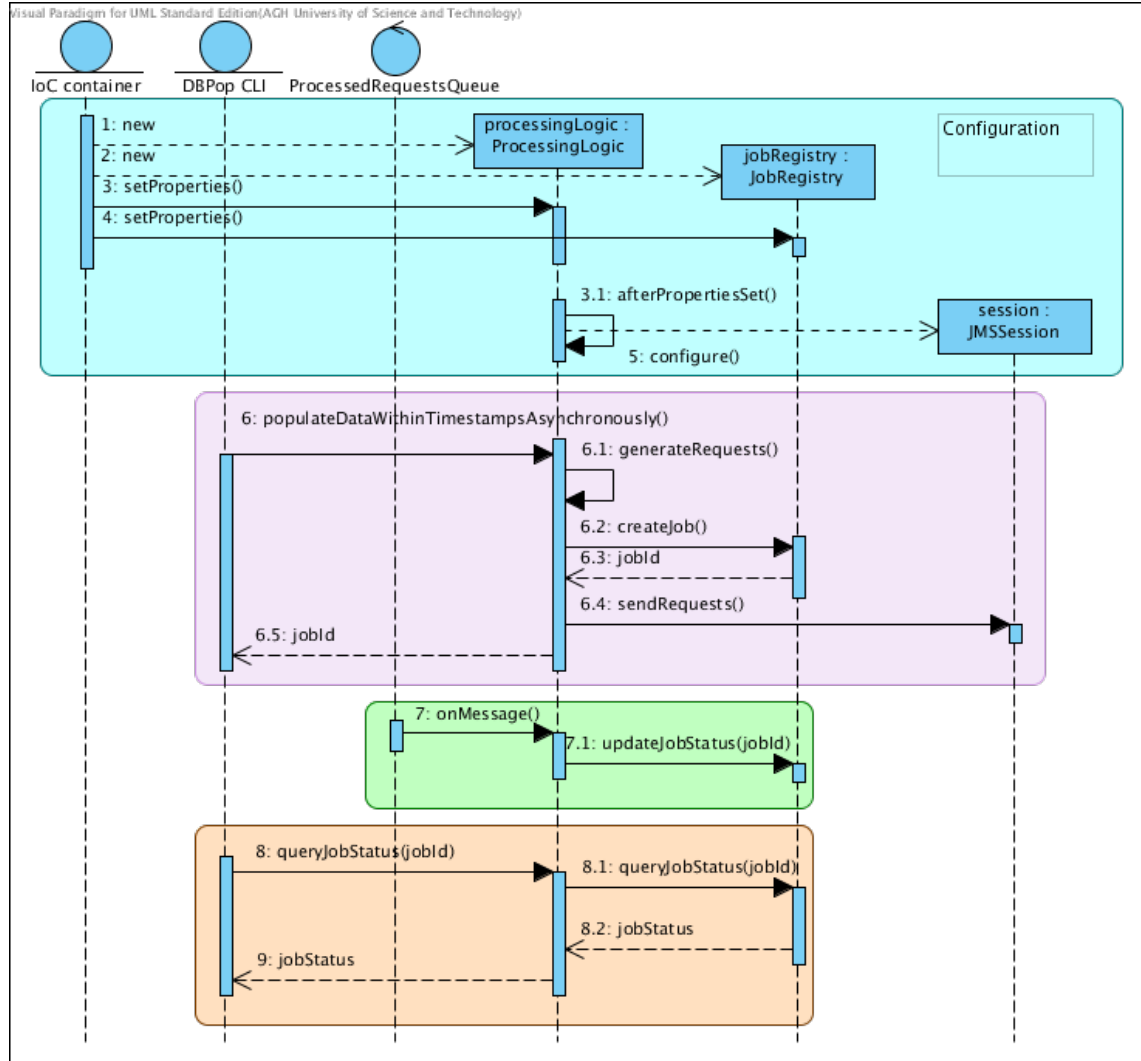


Figure 4.4. Sequence diagram of the ProcessingLogic module.

of self-developed mechanisms. Usage of Spring Batch might be a more standards-based approach. Spring JMS template usage is also considered, but needs careful preparation.

4.2.4. Other data accessing modules

At the time of this thesis creation, works on introducing new data sources were ongoing. The scientists are interested in putting general measurements data, which come from intermediate calculations, into the database for persistent storage. Using *XML* files seems to be the best way to provide this data writing functionality. *Java Architecture for XML Binding (JAXB)* provides proper means of transforming *XML* files into Java objects as well as from objects into *XML* files.

DBPop can listen on an agreed folder in the file system and process the files which get placed in it. File system poller component can be as simple as the Camel endpoint with *JAXB* mechanisms for *XML* parsing. This solution was already proposed at the

meetings of TOTEM Offline Software Group. It is however not discussed further in this thesis, since it is currently in development.

TOTEM Offline Software is supposed to have the means to write data into the database in a more direct manner. To achieve this, a distributable library needs to be built, which could be used from different programming languages, used within the *TOTEM* experiment, mainly C++, Python and Java. Ideas on introducing these new functionalities are emerging and need to be considered in future *DBPop* releases.

4.2.5. JMS queues

Apache ActiveMQ is used as the messaging infrastructure in ServiceMix 4.x. In *DBPop* it is extensively used as the communication backbone between services co-operating together as an application. JMS queues in *DBPop* play an important role, they are used for:

- communication between services
- jobs distribution
- mean of data transfer
- notification mechanisms

To instantiate *JMS* queues the mechanism of *JNDI* registry is used. It allows to register objects in a common pool and then administratively retrieve them in the application. The binding and lookup is done invisibly from the code by means of the Spring *JNDI* template. It helps to keep the code single-oriented, it leverages *POJO*-oriented programming, but needs some configuration. Worth noticing is that there are very few or even none signs of the *JMS API* in the modules code. It is all because of extensive usage of Camel, which mitigates the impact of variety of technologies on the application.

Communication

Instead of direct method invocations (which causes tight-coupling), modules communicate completely asynchronously via *JMS* queues. It assures maximum efficiency and reusability, because the requesting component does not have to wait for the servant component to finish its actions. It simply puts a message on the queue and continues its own processing. The format of the data being passed (without any routing and transformation framework like Camel) has to be agreed on both sides of the exchange, which is similar to defining a kind of a communication protocol. Routing framework like Camel (discussed in Section 2.2.3 and practically in Section 4.2.6) mitigates this effort leaving components completely unaware of the communication mechanisms and the fact that queues are used for data exchange. All of the work regarding format transformation is done behind the scenes by Camel.

Jobs distribution

The *DPRs* are generated by the ProcessingLogic module and placed on the *dbpop.-Requests* queue. Each of the data source adapters receives messages from this queue, containing information about the data that has to be acquired. After fulfilling the *DPR* contract, acquired data is placed on the *dbpop.Data* queue. By total separation

of the management and work components, perfect scalability is achieved. The *DPRs* can be collected by any number of independently and concurrently working nodes.

Data transfer

A naive approach to moving data from one place to another, would be to have a single-threaded application which reads from the data source into internal memory and then places it in the destination (depicted as A in Figure 4.5). It works very good for small data amounts. When the data is being continuously produced, and in large amounts, moving it in this fashion would become inefficient. Memory consumption for large chunks of data might be very high. It is also not always possible for the data source to provide the streaming functionality (getting data byte by byte), most often the data sources and data sinks provide a block access.

An intermediate solution would be to introduce more threads: some to read the data and some to write it in the destination (depicted as B in Figure 4.5). This results in performance enhancement, but with somewhat complicated concurrent programming model, including synchronization mechanisms like locking, semaphores and monitors. Effectively managing memory consumption is an important issue, since the data producers and consumers are very unlikely to work on the same pace. Data source might deliver the data faster than the sink is able to acquire. This can lead to memory exhaustion at some point of execution thus an application needs careful design not to be vulnerable to such threats.

What with a situation, when the data amount is very large, comes from variety of sources and a single application cannot deal with it efficiently? In order to be ready for these challenges, a highly performant and scalable solution needs to be used. In this thesis work, the means provided by the *ESB* are given a trial.

In the *DBPop* system, the *JMS* queues provided by ActiveMQ are used as the means of data transfer. The ProcessingLogic component is responsible for governing all the modules which are performing data access, but does not participate in the data transfer itself. The data gets passed by means of an external entity, a JMS queue: *dbpop.Data*, which functions as the data buffer for the DatabaseBinding component.

In order to improve the data transfer efficiency an innovatory approach is used (illustrated as C in Figure 4.5). What can be done is to govern the data transfer process in a way that all the resources are used efficiently. The most effective way of transferring large amounts of data is the "fire and forget" model - an asynchronous, batch process, without any interactions from the user. The ideal way would be to have the data being passed in a serial stream (byte per byte) with concurrent read and write. It is unfortunately not possible, since the data sources and data sinks use rather a "block" approach (e.g. LHC Logging: get data from a time range, database: *SQL* statements need to be organized into bulks of certain size executed in a transaction). What we can do is to adjust the data transfer process to maximally resemble the stream model. We cut the data into small pieces and instead of the basic unit of division being a byte, we have a so-called *chunk*. These *chunks* are being passed through the system in a similar way to a byte stream. What we achieve is that data endpoints (data-sources adapters) and the data sink work on maximum pace (constantly engaged by processing small pieces of data). This leads to maximum resources usage, which is a desired virtue.

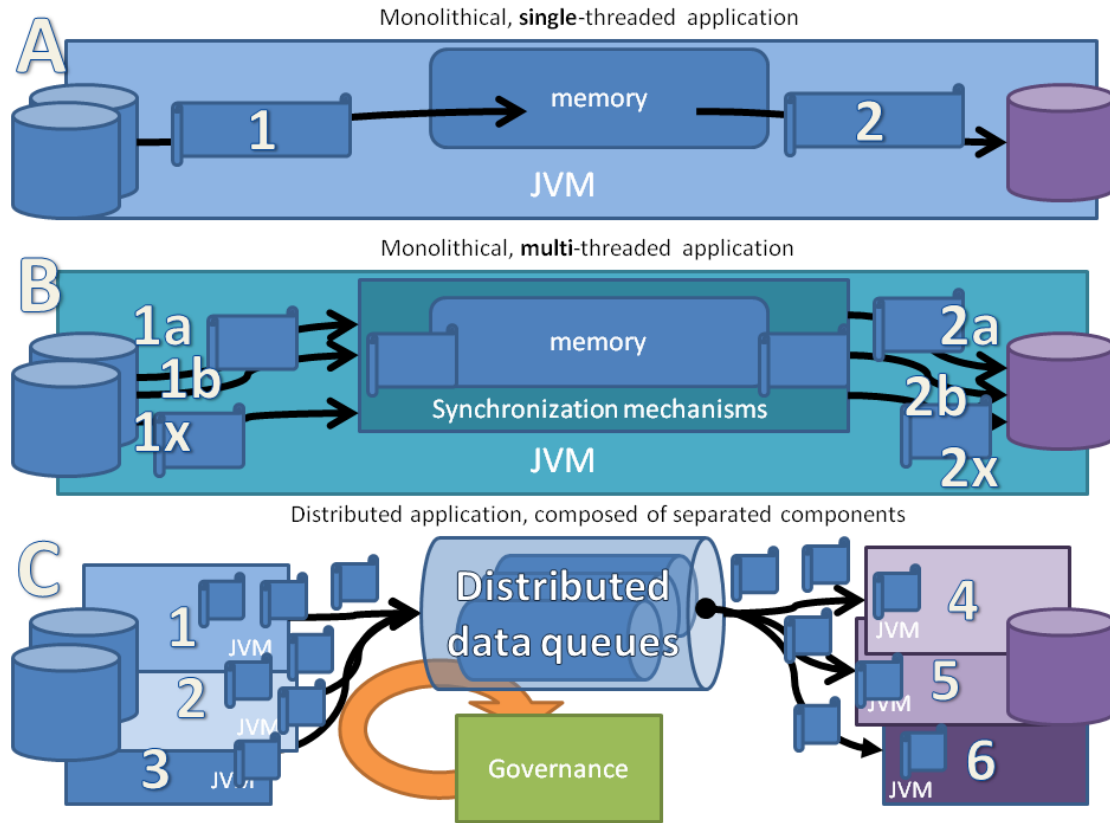


Figure 4.5. Data transfer strategies.

If the data stream produced by the sources is too large to be handled by a single queue, it might be distributed across several *JMS* brokers (on the level of *network connectors*⁶), so one queue is physically distributed across the brokers. If the data to be written is independent from each other (the writing process might happen out of order), the writing component might be deployed in several instances and used concurrently. Using the *ESB* approach it is not required to reengineer the whole system architecture. Only the deployment scenario changes. Instead of having one writing or reading component, system administrator adds several others and connects them to the system, easily realizing load balancing policies. Figure 4.5 illustrates the discussed data transfer strategies.

It turns out that using asynchronous data transfer infrastructure based on *JMS* queues has many advantages. Presented concepts are put under tests in the last chapter of this thesis (Chapter 5).

Notification

ProcessingLogic module is a consumer at the *dbpop.ProcessedRequests* queue. This queue receives messages from working nodes of *DBPop* about the status of fulfilling the *DPR* contract. As these messages arrive randomly, in unordered manner, the ProcessingLogic module implements the *MessageListener* interface with *void onMes-*

6. <http://activemq.apache.org/networks-of-brokers.html>

sage(Message msg) method. The JobRegistry tracks active jobs and updates their statuses generating reports in case of job completion or failure together with its cause.

4.2.6. Camel routes

Components described in previous subsections: *OSGi* services and *JMS* queues are the working blocks of *DBPop*, but do not know anything about each other. They are independent, loosely-coupled modules able to be reused by other components deployed in the same execution environment. Connecting them together (to create a working mechanism) is done by means of the Apache Camel routing framework. Camel routes provide the following features:

- **communication protocol abstraction**

The components do not have to implement sophisticated communication protocols, which might include: *FTP*, *HTTP*, *JMS* and many others⁷. It is done by means of Camel endpoints, which are described by *URIs*. Camel promotes declarative way of programming. Instead of coding the functionality of the connection it is configured as the *URI* option. The following piece of code is an example of *URIs* providing access to an *FTP* server and a Java bean:

```
from("ftp://me@myserver?password=secret")
.to("bean:foo?method=bar");
```

Listing 4.8. Example Camel route.

- **localization abstraction**

The components participating in the data exchange do not have to be deployed in the same *JVM*. Location of the components is declared in the endpoint's *URI*.

- **data format transformation**

Data provided by one component might not stick to the format accepted by the other. It is transformed on-flight by Camel.

- **realizing complex *EIPs***

Messages are routed to the destinations based on the content or environmental conditions. Accounting, logging, load-balancing policies can be applied.

There are three main Camel routes currently deployed in the *DBPop* system, configured by means of the Java *DSL*. For the sake of clarity, detailed configuration as well as exception handling is not covered in the code listings (please refer to Figure 4.1 to see graphical representation of the routes):

1. PollData

Periodically issues data taking from the data sources (including LHC Logging Databases). If the current Beam Mode in the *LHC* is one of those specified in the system configuration - the database population starts automatically.

```
from("timer://pollData?fixedRate=true&period=" + pollingInterval)
.to("bean:processingLogic?method=populateData");
```

Listing 4.9. PollData Camel route periodically issuing data population.

7. <http://camel.apache.org/uris.html>

2. ProcessRequests

This route delivers *DPRs* from the *dbpop.Requests* queue to the data source adapters. After successful data acquiring, it is placed on the *dbpop.Data* queue by means of this route. If the *DataContainer* happens to be empty it is placed directly on the *dbpop.ProcessedRequests* queue.

```
from("activemq:queue:" + requestsQueueName)
  .to("bean:lhcApiPoller?method=pollData").choice()
  .when(new Predicate() { // We filter out empty messages
    // Predicate content resulting in true or false values
  })
  // If the message has some payload
  .to("activemq:queue:" + dataQueueName)
  .otherwise() // If the message is empty
  .to("activemq:queue:" + processedRequestsQueueName);
```

Listing 4.10. ProcessRequests Camel route distributing DPR's to the data source adapters and placing data on the buffer queue.

3. FromQueueToDatabase

This route takes the data which appears on the *dbpop.Data* queue and delivers it to database writing components. After successful data writing, the notification is placed on the *dbpop.ProcessedRequests* queue.

```
from("activemq:queue:" + dataQueueName)
  .to("bean:databaseBinding?method=saveData")
  .to("activemq:queue:" + processedRequestsQueueName);
```

Listing 4.11. FromQueueToDatabase Camel route realizing data writing to the database and notification of finished jobs.

Another route is being deployed, designed to support *XML* files from the file system to the queue infrastructure.

4.3. TOTEM Offline Database

In this section, the data sink for the *TOTEM DBPop* system - TOTEM Offline Database, is given a general overview. Only the most important aspects are discussed, since the database itself is not the main subject of this thesis. A more detailed description of the database, including tables description, stored procedures and triggers as well as indexes and constraints definitions can be found in [39] and [41].

4.3.1. Schema

TOTEM Offline Database is an ongoing project. There are constantly new projects and tools being developed that are related to the database. Because of new requirements appearing, the schema is not in its final state.

The central point in the database schema is an entity called *Interval Of Validity* (*IOV*), which represents a range of events (or timespan) when the kept data is valid. The Totem Offline Software retrieves data on the basis of the *IOV*, therefore it is bound to every measurement data. Figure 4.6 shows the concept of *IOV*. The more

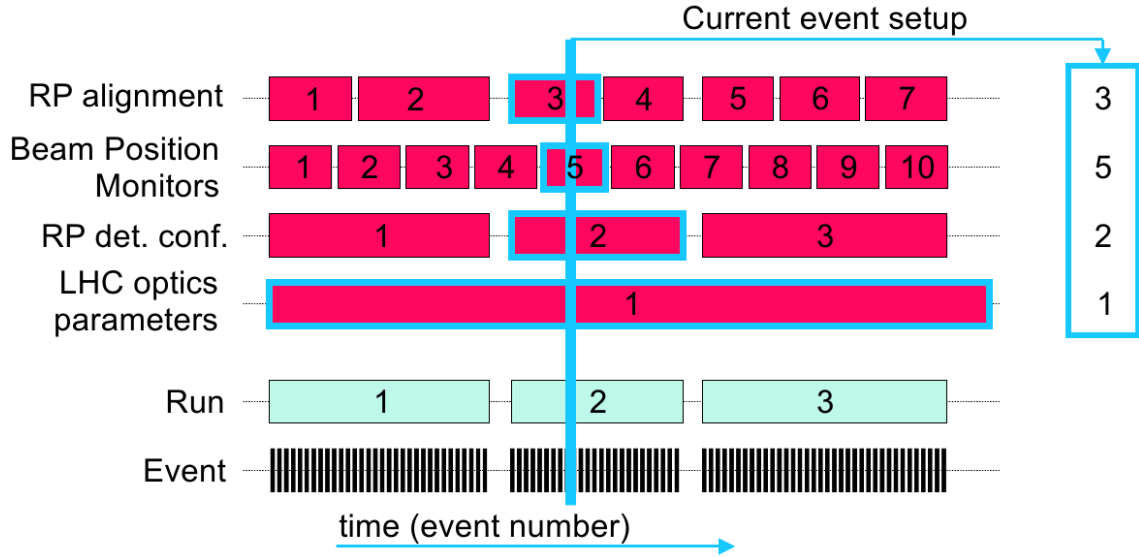


Figure 4.6. The concept of Interval of Validity. Figure from [1].

detailed explanation for this model of managing data can be found in [1], Appendix C.3.

Figure 4.7 shows the actual state of the database schema. There are 18 tables in the schema, which can be divided into 6 groups:

- **Timeline** tables: T4RunTimelineMap, T19EventTimelineMap
- **Detector structure** tables: T9StructureElementType, T10StructureElement, T11SensorPart and T12SensorPartType
- **Alignment** tables: T1DetectorAlignment, T2AlignmentSource, T5Alignments, T6UsedAlignments, T8DefaultAlignment
- **Validity interval** tables: T3ValidityInterval, T7IovType
- **Sensor part status** tables: T13SensorPartStatus, T14StatusList
- **Measurements** tables: T15MeasurementType, T16SensorPartMeasurement, T17StructElementMeasurement, T18GeneralMeasurement

The most important tables for the *DBPop* system at the current state of development are those in the Validity Interval and Measurements groups. The description of these tables is covered in the proceeding subsection. Other tables' description can be found in [39].

4.3.2. Tables description

Since for now, the *TOTEM DBPop* does not provide the *DAQ* and related (alignments, run timelines) data support, there are only few tables that are currently being used. These tables will be given a detailed overview.

T3ValidityInterval and T7IovType

Interval Of Validity (IOV) is the central point of the offline software and the database. The Interval information is stored in one table and uses one dictionary

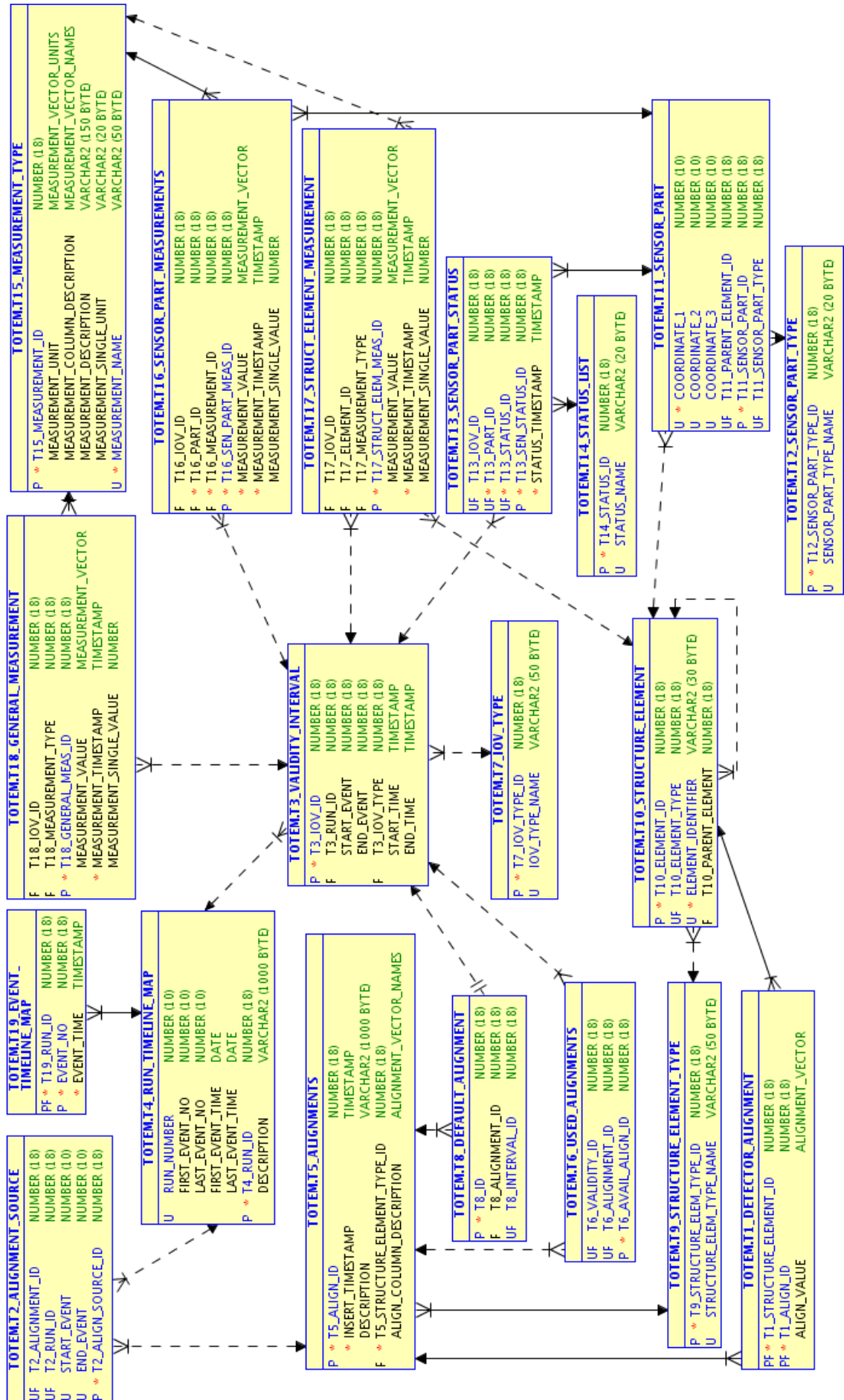


Figure 4.7. TOTEM Offline database schema in August 2011.

Table name	T7IovType	
Description	This table keeps information about the types of Interval of validity	
Fields	t7IovTypeId	primary key of the table, number, has no domain meaning
	iovTypeName	name of the Interval type, label
Outside relations	With T3ValidityInterval: many to one	

Table 4.1. T7IovType table details.

Table name	T3ValidityInterval	
Description	This table keeps information about Intervals of validity	
Fields	t3IovId	primary key of the table, number, has no domain meaning
	t4RunTimelineMap	foreign key, determines the run this Interval belongs to
	t7IovType	foreign key, determines the type of this Interval
	startEvent	number of the first event of this Interval
	endEvent	number of the last event of this Interval
	startTime	start of the Interval in case of non-event measurements
	endTime	stop of the Interval in case of non-event measurements
Outside relations	T6UsedAlignments	many to one
	T8DefaultAlignment	one to one
	T13SensorPartStatus	many to one
	T16SensorPartMeasurement	many to one
	T17StructureElementMeasurement	many to one
	T18GeneralMeasurement	many to one

Table 4.2. T3ValidityInterval table details.

Table name	T15MeasurementType	
Description	This class keeps available measurements type name. As each of the measurements is a vector of numbers (double), it also contains list of labels for each of the vector elements (mappings), and also units for each of the values	
Fields	t15MeasurementId	primary key of the table, number, has no domain meaning
	measurementUnit	array of strings, containing a list of units for each of the values
	measurementColumnDescription	array of strings, containing a list of labels for each of the values
	measurementDescription	a label for this measurement type
Outside relations	T16SensorPartMeasurement	many to one
	T17StructureElementMeasurement	many to one
	T18GeneralMeasurement	many to one

Table 4.3. T15MeasurementType table details.

Table name	T18GeneralMeasurement	
Description	This class keeps single measurement or a vector of doubles that is general - not related to any element of the detector structure	
Fields	t18GeneralMeasId	primary key of the table, number, has no domain meaning
	t3ValidityInterval	foreign key of the Validity interval table, binds the given measurement with its validity interval
	t15MeasurementType	foreign key for the measurement type table, defines the kind of measurement
	measurementValue	vector of doubles that keeps the vector measurement
	measurementSingleValue	single measurement value
	measurementTimestamp	timestamp of the measurement
Outside relations	T15MeasurementType	many to one
	T3ValidityInterval	many to one

Table 4.4. T18GeneralMeasurement table details.

table that defines the type of *IOV*. Tables 4.1 and 4.2 gather the most important information regarding considered tables.

T18GeneralMeasurement and T15MeasurementType

Any measurements different than those which have designed tables for storing them are kept in one of three tables, depending on the element of the structure that this measurement is connected with. There are three types of measurements: general (not related to any structure element), related to a Structure element and related to a Sensor part. All current data is put in the T18GeneralMeasurement table, since the measurements are not related to the *TOTEM* detectors structures. Tables 4.3 and 4.4 gather the most important information regarding considered tables.

4.3.3. Mappings

DBPop currently uses the *ORM* approach of database manipulation. Hibernate is used as the provider in the *JPA* model (described in Section 2.2.5). In order to have the means to manipulate the database programmatically, the object mappings of the tables had to be created. Because the schema had been designed first, the bottom-up approach had to be used - the object model of the data was generated from the *Data Description Language (DDL)* of the schema with the tools provided by Hibernate. The mappings are a part of the *DatabaseBinding* component.

4.4. Development environment

This section covers some aspects regarding development environment. Because of innovatory application architecture and remote works in the network of *CERN*, the development environment had to be properly established. The most important aspects of this topic are presented in the proceeding subsections.

4.4.1. Using Maven

Apache Maven⁸ is used as the project life-cycle management tool for *DBPop*. All of the project configuration, including dependencies, artifact repositories and deployment scenarios are described by means of a *Project Object Model (POM)* file. A *POM* file is an *XML* file conforming to a common project schema. Each of the *DBPop* modules is described separately by a *POM* file, but the whole project is managed by the parent-*POM*.

Maven uses the concept of artifacts as the output of a project build. Artifact repository is the central place of storage for produced packages as well as for the dependant libraries. A dedicated Maven repository (Artifactory⁹ was used as the Maven repository provider) was established at the *TOTEM* experiment to store proprietary packages and to function as a proxy for central repositories.

8. <http://maven.apache.org>

9. <http://www.jfrog.com/products.php>

4.4.2. Working with IDE

Having control over the development process is not an easy task. There is a large scale of technologies and environments that need to be used, but are rarely related to each other. To simplify the development process it is suggested to use an *Integrated Development Environment (IDE)*, which tries to gather all the environments in one application.

Eclipse¹⁰ is the recommended *IDE* for developing *DBPop*. Useful plug-ins¹¹, which are essential for the work to be convenient, are: m2eclipse¹², Spring and *Subversion (SVN)* plugins. Since *DBPop* project structure is governed by Maven, each of the components is visible as a separate project (it posesess its own *POM*).

4.4.3. Working remotely

As some of the tools and infrastructure used by *DBPop* (Java API for accessing *LHC* Logging Databases, documentation and hardware resources) are restricted to be used only at *CERN* there was a need to tackle this problem. A *Secure SHell (SSH)* tunnel was established to forward connection to ports on desired computers. A *Virtual Network Computing (VNC)* protocol implementation - Tight *VNC*¹³ - was used in order to provide remote desktop functionalities.

4.5. User interface - DBPop CLI

TOTEM TOTEM DataBase POPulation System (DBPop) system is divided into two parts: the system backend - working as a daemon in the ServiceMix *ESB* environment and the *Command-Line Interface (CLI)*, a separate application which gets connected to the backend from a remote location. Extension of the interface to include *GUI* is considered as future works. This section discusses briefly the functionality and working mechanism of *DBPop CLI*.

4.5.1. Implementation

DBPop CLI is a standalone Java application. It transforms the user's orders into proper invocations on the *DBPop* backend.

Interpreting command-line instructions

Apache Commons *CLI*¹⁴ is used as the tool for acquiring, validating and interpreting values provided by the user from the command line. It provides simplified and standardized *API* which helps to perform boiler-plate tasks and clears the code from cumbersome parsing attempts.

10. <http://www.eclipse.org/>

11. Eclipse is built on top of the *OSGi* framework. Eclipse plug-ins are basically *OSGi bundles*.

12. <http://m2eclipse.sonatype.org/>

13. <http://www.tightvnc.com/>

14. <http://commons.apache.org/cli/>

Steering channel

Java Management eXtensions (JMX) is used as the connection between the *DBPop CLI* and the backend. It runs on top of the *Remote Method Invocation (RMI)* - technology providing distributed computing facilities in Java environment. A proxy of the *ProcessingLogic* component is created in the application and by means of the connection established to the MBean server, the interaction is performed. Exposing the *ProcessingLogic* component as a *JMX* MBean gives the possibility to use any *JMX* compliant client to invoke its methods.

4.5.2. Functionality

DBPop supports two usage scenarios for data population:

- automatic, current data population based on the Beam Mode in the *LHC*
- manually triggered population (via *CLI*)

There are two manual population modes:

- **batch (asynchronous)**
submitting jobs to populate data within long periods of time
- **interactive (synchronous)**
for short periods of time population, useful in scripts

Manual population can be issued based on two possibilities: Timestamp-based (provide start and end timestamps) or Fill and Beam Modes-based (provide *LHC Fill* number and interesting Beam Modes). Any issued data population job can be tracked by a query mechanism. After job completion, an e-mail notification will be sent on the system administrator's account. User can turn on or off the automatic data taking and the usage of asynchronous queue infrastructure.

4.5.3. Usage

Running *DBPop CLI* without any arguments lists all the options which might be passed as an input:

```
usage: DBPopCLI
-a,--ask                Prints the request summary and waits for user
                        acceptance
-b,--beam <arg>        Beam Mode Values for the data to be populated
-d,--disable <arg>    Disables the automatic data-taking/asynchronous
                        infrastructure
-e,--enable <arg>     Enables the automatic data-taking/asynchronous
                        infrastructure
-f,--fill <arg>        LHC Fill Number for the data to be taken
-h,--help              Prints the usage information
-i,--interactive       Waits for the data population to be completed
-m,--measurements <arg> Measurements to be populated
-q,--query <arg>      Checks the status of specified job number
-t,--time <arg>       Time interval for the data to be populated
```

Listing 4.12. DBPop CLI commands list.

Examples

- Populate data for *BPMs* and *BLMs* within specified period of time in batch mode (default):

```
Request:
# dbpop --time 2011-08-02_19:00:00.000 2011-08-02_19:20:00.000 --m BPM,BLM
Output:
# DBPopJobID = DBPop#2011-08-10_20:59:56.357#4
```

Listing 4.13. DBPopCLI usage example - issuing batch (asynchronous) job

- Interactively populate data for *Beam Quality Monitor (BQM)* and *BLM* within specified period of time:

```
Request:
# dbpop -t 2011-08-02_04:00:00.000 2011-08-04_19:50:00.000 -m BQM, BLM --i
Output:
# Job done!
```

Listing 4.14. DBPopCLI usage example - issuing interactive (synchronous) job

- Populate data for ALL measurements within LHC Fill number 1147 while the beam mode was STABLE and SQUEEZE, ask for confirmation in batch-mode:

```
Request:
# dbpop -f 1147 -m ALL --beam SQUEEZE,STABLE --ask
Output:
# Request summary:
Fill number: 1467
Populated measurements: ALL
Beam Mode Values: SQUEEZE,STABLE
Is this correct? (Y/N): Y
# DBPopJobID = DBPop#2011-02-16_20:59:56.357#4
```

Listing 4.15. DBPopCLI usage example - populate data for LHC fill

- Query for the given DBPopJobID job status:

```
Request:
# dbpop --query DBPop#2011-02-16_13:47:16.954#1
Output:
# DBPop#2011-02-16_13:47:16.954#1 is in PROGRESS with (41/83) DPRs pending...
```

Listing 4.16. DBPopCLI usage example - querying job status

4.6. Chapter summary

This chapter begun with introducing system architecture using an implementation of the *Enterprise Service Bus (ESB)* concept - Apache ServiceMix 4.x. All of the main components, including *OSGi* services, *JMS* queues and Camel routes were given a detailed insight. The means of transporting data in the *ESB* environment were discussed together with presenting the approach used in the designed system. Next, the data sink of the *DBPop* system - TOTEM Offline Database was given a general overview. Development environment together with the tools aiding the development process were described afterwards. At the end, the *DBPop CLI* was given a short introduction.

The *DBPop* system is in constant development. The works under the *TOTEM* Offline Database project are ongoing and *DBPop* needs to be adjusted to meet new requirements. The reader should not treat this section as a strict programmer's guide, since the method names, configuration files structure etc. might not resemble the current state of development of *DBPop*. Nonetheless information provided in this chapter give the general overview of the system working mechanisms which are crucial to understand in order to develop the system further.

Chapter 5

Tests

Previous chapters described the *TOTEM DBPop* system in great detail. System design and implementation were covered (Chapters 4 and 3), together with theoretical background containing information regarding used technologies and *TOTEM* experiment environmental conditions (Chapter 2). The tests performed in this section will bring answers to the topic of this thesis: how efficient an *ESB* can deal with transferring large amounts of measurements data. In the following chapter, author is proving thesis genuineness, by evaluating designed system in several aspects.

First section introduces the tests environment. It is crucial to understand its assumptions in order for the tests to be meaningful. Next two sections evaluate key system values. Efficiency tests are described in Section 5.2, while scalability tests are depicted in Section 5.3. The proceeding section discusses fault-tolerancy and extendability issues. Whole chapter is concluded in Section 5.6.

5.1. Tests environment

In this section the scenario of performed tests is analyzed. In order for a test to be meaningful, the reader has to understand its aim and conditions in which it was performed. It is also desirable, that the test resembles a real situation from the system life-cycle.

Tests were performed at *CERN* using infrastructure of the *TOTEM* experiment and general *CERN* hardware. By a *test environment* it is meant the whole hardware and software infrastructure involved in specified scenario, conditions of external systems, as well as configuration of internal subsystems. First, the hardware infrastructure will be presented, including network topology and placement of the external systems within it. Next, software entities, their configuration and mapping on the hardware infrastructure are discussed. Test scenario and the methodology of measuring evaluation values are discussed in proceeding subsections.

5.1.1. Hardware

Machines

The tests were performed on three machines configured as it is presented in Table 5.1. Machines names are: PCtotem30 and PCtotem31 and PCtotem36.

Processors	2x(AMD Opteron 2210, 2 cores, 64bit, 1800Mhz, 2x1MB L2 cache)
Internal memory	4GB DDR2 667MHz ECC
Main board	Nvidia NForce MCP55, Dual 1207-pin Socket F
Network interfaces	2x (1Gb/s Ethernet)
Storage	Maxtor 6V160E0, 160GB, 7200RPM, 8MB cache, SATA2; In addition on PCtotem31: RAID1 2x (Western Digital VelociRaptor WD6000HLHX, 600GB, 10000RPM, 32MB cache, SATA3)

Table 5.1. Test machines configuration.

These stations in clustrization tests are accompanied with a laptop equipped with 3GB of DDR2 memory and Intel Core2Duo CPU clocked 2,13GHz. This machine runs only the *DBPop CLI*, and does not have significant impact on the whole system performance.

Hardware hosting the LHC Logging service is a powerful configuration adjusted to provide highly-available and efficient data extraction. The architecture of the LHC Logging services was depicted in Section 3.2.1.

Spacalized organization unit in *IT* department at *CERN* provides Oracle Database Services to host range of databases used by the experiments. Current state of the

Processors	2x (Intel Xeon L5520 (Nehalem), 4 cores, 64bit, 2270MHz, L2 cache 4x256 KB, L3 cache 8 MB)
Internal memory	24GB DDR3 1066MHz ECC
Main board	Nvidia NForce MCP55, Dual 1207-pin Socket F
Network interfaces	2x (1Gb/s Ethernet bonding Active-Passive (for CERN internal network)) + 2x (1Gb/s Ethernet bonding Active-Passive (for communication within the cluster))
Storage	Dedicated SAN with 4Gb FibreChannel connections, 2x storage array of 12x Western Digital 2TB SATA3 configured as JBOD

Table 5.2. Machine configuration hosting TOTEM Offline Database in the IT Department at CERN.

TOTEM Offline Database project uses database schema on Integration level¹, which is mainly used to test application's behaviour under the workload similar to the expected production one [42]. Database on Integration level delivers certain quality of service:

- 8/5 monitoring and availability

1. A database application at CERN follows an agreed development roadmap: Development - Integration - Production, described in [42]

- Dedicated node for testing application performance with expected data volume and concurrency
- The same hardware infrastructure as on Production level

TOTEM Offline Database (together with other databases of the *LHC* experiments) runs on 2 nodes of Oracle *Real Application Clusters* (*RAC*). Each of the machines has the configuration presented in Table 5.2. The *DBPop* system and TOTEM Offline Database schema are naturally not the only clients of these machines. Nonetheless the tests were performed at night, when the network traffic is very low.

Network topology

The tests were performed in the *CERN* internal network. A demonstrative picture of the network topology used for the tests is illustrated in Figure 5.1. A more detailed

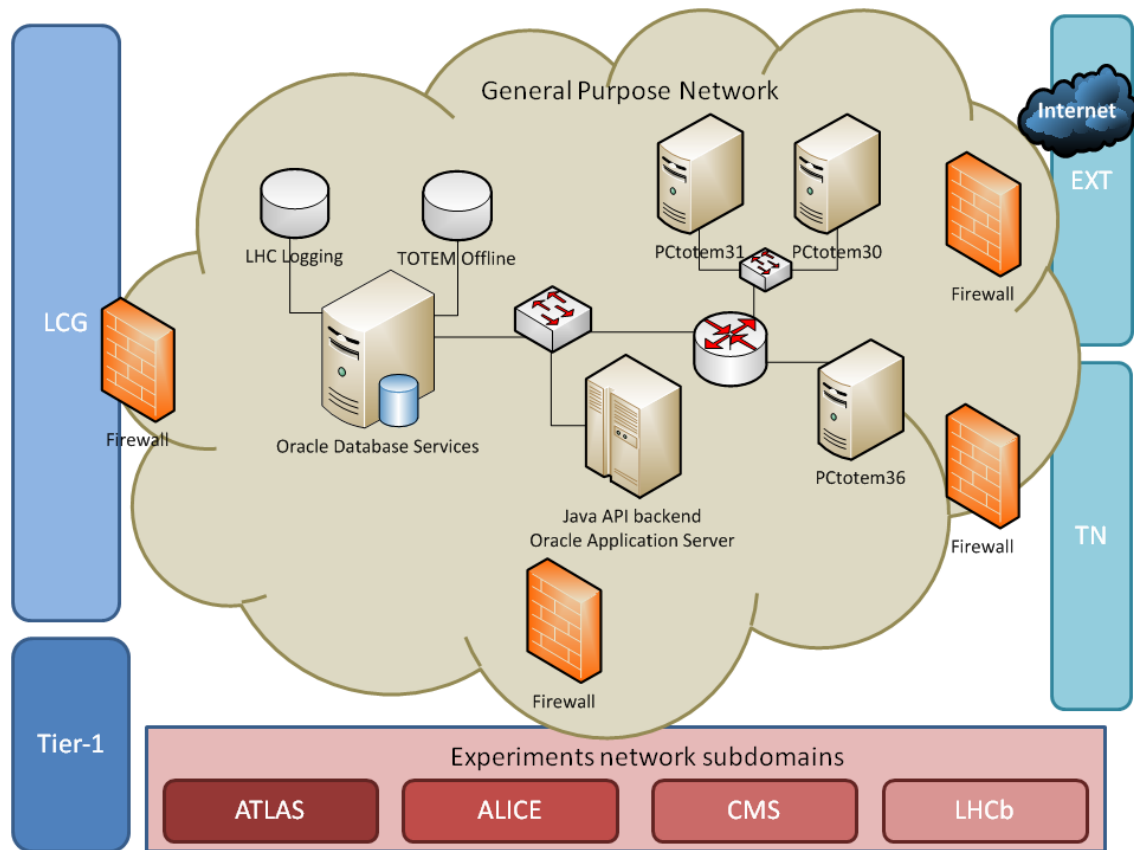


Figure 5.1. Simplified network topology at CERN with test infrastructure. All of the systems participating in the tests are accessible in the GPN.

insight in the network topology at *CERN* can be found in[43].

CERN network is divided into domains. The most important areas are: *General Purpose Network* (*GPN*), *Technical Network* (*TN*), *LHC Computing Grid* (*LCG*) and particular experiments' local networks. All of the systems involved into tests scenario are visible in the *GPN*, which is providing at least 1Gb/s Ethernet connections. LHC Logging Databases and TOTEM Offline Databases schemas are hosted by the IT

department in the Computing Centre at *CERN*. PCtotem31, PCtotem32, PCtotem36 machines are located in the *TOTEM* experiment offices.

On the presented hardware different software infrastructure was established, which is discussed in proceeding section.

5.1.2. Software

Tuning of the system as a whole, which means changing the configuration parameters of the subsystems, is a very complex task. What is being adjusted is not only the parameters of the base runtime environment, but also the operating system, *JVM*, the database, external systems and *MOM* brokers. Properly performed tuning might significantly improve overall system performance, however requires lot of patience and intuition.

The way of choosing optimal values and solutions are specific for each of the considered subsystems. Moreover, great complexity of the runtime environment and all of the building blocks results in mainly experimental research for optimal composition, based on the administrator experience [44].

Operating systems

Each of the machine runs a flavour of Linux - *Scientific Linux CERN (SLC)*, which is a Linux distribution maintained at *CERN*, equipped with specialized tools for physics scientists. *SLC* is based on *Red Hat Enterprise Linux (RHEL)* and accepts new kernel releases very slowly (in order for the operating system to be stable). PCtotem30 runs *SLC* 4 with a *Symmetric Multiprocessing (SMP)*-compatible kernel (version 2.6.9-100). It is an old configuration, but supports all of the operations needed for *DBPop* deployment and tests execution. It was unable to check the system behaviour on newer configuration since, at the time of this thesis creation, it was expecting major upgrades due to *SLC* 5 migration. PCtotem31 runs *SLC* 5 with a *SMP*-compatible kernel (version 2.6.18-274).

Java Runtime Environment

When deploying solution in production environment it is crucial to choose the right *JVM*, since the whole system relies on its performance and functionality. Development works were performed on the reference *JVM* implementation provided by Oracle in version 1.6.0.21, but because of not efficient *Garbage Collector (GC)* mechanisms dealing with static classes data (PermGen² space does not get collected properly causing out of memory exceptions) the implementation was switched to the *JVM* provided by IBM.

For tests, the chosen version was 64 bit IBM *Java Development Kit (JDK)* 6.0-9.2. The *JVM* from IBM deals better with mentioned problem by dynamically extending PermGen space which in Oracle's *JVM* stays constant leading to *OutOfMemoryError*.

2. PermGen is a place (so-called "generation") on the Java heap designed to store loaded classes structures

JMS Brokers

Apache ActiveMQ in version 5.5.0 was used for the tests. It was configured according to the guidelines provided in [45] and [26] as well as consulting ActiveMQ experts at *CERN*. Table 5.3 lists the most important configuration aspects of ActiveMQ³:

PrefetchSize	Set to 1 on each consumer, provides proper load balancing across the components
ProducerFlowControl	set to true, slowing down producers, when the buffer limit is about to exceed
SystemUsage	memoryUsage = 256mb, storageUsage = 2 gb, tempUsage = 512mb, prohibits memory exhaustion
TransportConnectors	vm:// for internal message broker, tcp:// for external broker
MessagePersistency	turned off, since in case of failure the data can be downloaded again

Table 5.3. Configuration of ActiveMQ JMS Broker used in tests scenario.

Messages passed in the *DBPop* system are of large granularity (megabytes), that's why the PrefetchSize parameter is set to 1 on each consumer. It prohibits slower components from consuming messages on behalf of the faster. ProducerFlowControl mechanism is turned on to save the queues from data exhaustion in case of very fast data producers. SystemUsage is adjusted to meet hardware conditions of the machines where the solution is deployed. Transport connectors are chosen regarding to the actual deployment scenario. There is no need to use persistent storage for passed messages, since the data can be downloaded on demand from the data sources.

Database

Oracle Database services at *CERN* currently run on 110 Oracle 10g *RAC* nodes, which host all of the *LHC* Experiments databases, including TOTEM Offline Database. *DBPop* was optimized for writing performance in consultation with the *IT* Department experts. Table 5.4 presents the most important configuration of the Database-Binding component of *DBPop* running on top of the Hibernate framework:

DataSource	oracle.jdbc.pool.OracleDataSource
hibernate.jdbc.batch_size	100, established on experimental basis
hibernate.cache.use_second_level_cache	false, does not have sense for writing
hibernate.statement_cache.size	0, does not work well with Oracle
hibernate.connection.isolation	READ_UNCOMMITTED, only writing
pool InitialLimit	3
pool MaxLimit	5

Table 5.4. Configuration of the DatabaseBinding component used in efficiency tests.

3. The meaning of each configuration parameters can be viewed on: <http://activemq.apache.org/>

JDBC batch size was set on basis of experience gathered while the tests performance. Caching does not have sense in a "writing-oriented" application, causing useless overhead. The Oracle *JDBC* driver does not seem to work well with prepared statements, so it was also turned off. Each instantiated DatabaseBinding component maintains its own connection pool to the database of minimally 3 connections. Transaction isolation level was set to `READ_UNCOMMITTED` - there is no need to increase locking overhead in writing-only application.

Monitoring system condition

DBPop code was instrumented to provide useful information about its working performance, which might be easily reached in application logs. Several small tools were developed besides the *DBPop* itself dedicated to measure performance of external systems. A *classmexer* project⁴ was used in order to precisely measure the amount of data dealt by *DBPop*. For monitoring *JVM* status as well as having control over *JMS* queues and Camel routes, the *jconsole*⁵ tool was used, which is provided together with Oracle's *JDK* distribution.

5.1.3. Tests scenario

The tests scenario is actually the main use case of the *DBPop* system - data transfer from the LHC Logging Databases to the TOTEM Offline Database. One can imagine the following situation. After a successful *LHC Fill* there are meaningful data in the LHC Logging databases. With a successful *LHC Fill* usually a *TOTEM Run* is bound. The shifter decides to populate the data, based on the start and end timestamps of interesting period or for the *LHC Fill* with specified Beam Modes. Relations between *LHC Fill*, *TOTEM Run* and certain Beam Modes is clarified in Figure 5.2. An average *LHC Fill* lasts for about 8 to 16 hours going through many,

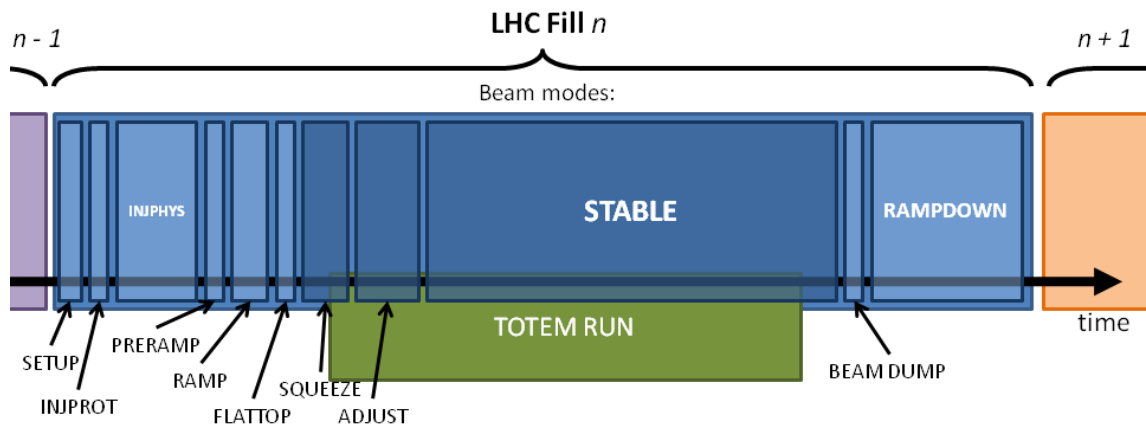


Figure 5.2. LHC Fill, Beam Modes and TOTEM Run relations.

so-called Beam Modes. Each Beam Mode represents a certain phase in the running accelerator run-cycle, starting from SETUP, through adjusting states like ADJUST,

4. <http://www.javamex.com/classmexer/>

5. <http://download.oracle.com/javase/1.5.0/docs/guide/management/jconsole.html>

SQUEEZE, to finally reach STABLE mode. At the end the beam has to be dumped which is represented with last phases, including BEAMDUMP and RAMPDOWN.

A single *TOTEM Run* might span across different phases in the Beam life-cycle, but usually interested data is delivered while ADJUST and STABLE Beam Modes. In the end, it is decided by the shifter, what period of time is relevant for the experiment. Meaningful data should be transferred to the TOTEM Offline Database in order for the *TOTEM* scientists to proceed with their research.

Let us take an example of LHC Fill number 2009 performed from 2011.08.08 00:48:06.451 till 2011.08.08 02:46:32.530⁶. We will consider *TOTEM*-relevant measurements during STABLE Beam Mode from 2011.08.08 00:48:06.451 till 2011.08.08 02:46:32.530. Table 5.5 shows the amount of data to be transferred by the *DBPop* system from one hour of STABLE beam.

Aggregated measurement name	Explanation	Data volume
<i>BPM</i>	6x (1076 values in an array) each 2 to 10s	18,76MB
<i>BLM</i>	36x 1 measurement each second	5,15MB
<i>BQM</i>	24x (3100 values in an array) each 2 to 10s	121,94MB
Roman Pots positions	48 x 1 measurements per second	13,7MB
Beam energy	2 measurements each second	0,56MB
CMS Luminous region	14 x 3 measurements per minute	0,13MB
Sum	267 802 measurements	160,86MB

Table 5.5. List of measurements acquired from LHC Logging Databases while one hour of STABLE beam.

Gathered together this volume is more than 160MB of raw data. These data is only an experimental set, which does not resemble the current needs of the scientists in the TOTEM experiment. It might be changed at any time using the configuration file (described in Section 4.2.1). Based on the given tests scenario, system's behaviour was evaluated in the following areas:

- Efficiency
- Scalability

5.2. Efficiency

The natural measure for efficiency in the data transfer system is the information amount processed in the unit of time: MB/s. However it is not used in the tests of *DBPop*. The reason is twofold. Firstly, measurements are divergent in their nature. Each has different sampling frequency, data format, some of them have mappings associated. Secondly, the data volume differs among representations. The same data in an internal memory on the Java heap has different volume than its textual representation. This differs also from the information amount passed in the *SQL* statements.

6. Current performance of the *LHC* is publicly accessible at <http://lhc.web.cern.ch/lhc/>

Instead, the quantity of processed measurements is used, which is more significant for the physics scientists.

In order to depict the differences between different ways of transferring data the test scenario was divided into three parts, in which the data was passed by means of:

- direct Java methods invocation
- external message broker
- internal memory message broker

Test scenario is illustrated in Figure 5.3. First, all three variants are described for

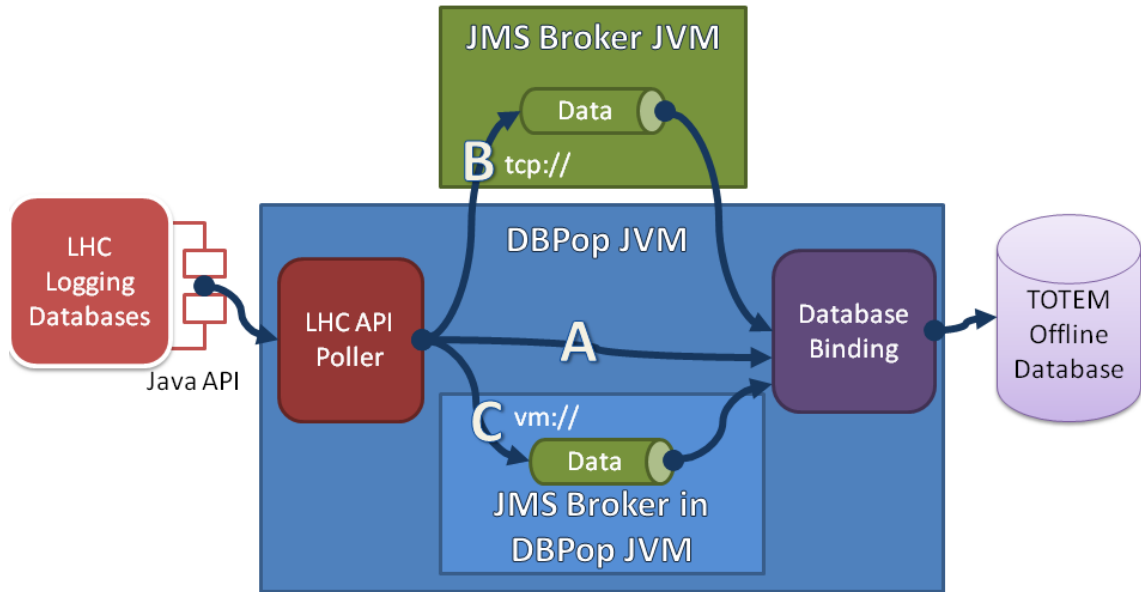


Figure 5.3. Efficiency tests scenario including: A - direct Java methods invocation, B - external message broker, C - internal memory message broker.

better understanding what actually happens behind the code. Then, at the end of the section, test results are provided, together with conclusions.

5.2.1. Direct methods invocation (scenario 1.a)

In this scenario, the data is passed in the application internal memory by means of traditional, synchronous, blocking Java method invocations, which is indicated by letter A in Figure 5.3. ProcessingLogic module in the key part of the population is performing the following operation:

```
for (DataPopulationRequest request : requests) {
    // Acquire data from LHC Logging Databases in given timespan
    DataContainer dataContainer = lhcApiPoller.pollData(request);
    // Save acquired data in the TOTEM Offline Database
    databaseBinding.saveData(dataContainer);
}
```

Listing 5.1. Direct methods invocation algorithm

This way of data population might be used in the *DBPop* but is strongly recommended not to do so. Here it is presented only to compare the traditional approach to the usage of asynchronous infrastructure based on JMS queues.

This naive way of populating data has some advantages:

- very easy programming model
- no need to use sophisticated technologies
- might be suitable for small data chunks

Nonetheless, it is not suitable to use such approach, when the application needs to deal with huge amounts of data, because:

- data chunks get passed in internal memory of the application, which might cause memory management problems
- sequential execution because of synchronous, blocking invocations are wasting resources, which are able to run concurrently
- high coupling (because of direct methods invocations) hinders application clustrization and distribution

The average time of transferring the test data achieved in this scenario was: **199.371s**.

One could improve this model by implementing sophisticated mechanisms based on threads to achieve concurrent execution. However it would be very demanding and cumbersome process, because threads synchronization and safeness is very difficult to implement and maintain. Especially hard would it be to implement the balancing policy, that would prevent the processing components from being overloaded. Such situation might be the cause of system failure or data writing rejection.

Section 5.4 includes the results of efficiency test performed in this scenario.

5.2.2. External message broker (scenario 1.b)

The disadvantages pinpointed in previous subsection are overcome in the component approach by entities specialized to perform the synchronization and safeness precautions on behalf of the programmer. It is achieved by introducing the approach of loosely coupled, asynchronous services working together by means of a messaging infrastructure (*MOM*). One of the best examples of such components are JMS queues, which are extensively used in the *DBPop* implementation.

Second scenario assumes that the data gets passed by means of a JMS broker, which is executed outside of the *DBPop* runtime environment. The ActiveMQ instance runs within its own *JVM*, completely separated from *DBPop*.

By maintaining separate, external message broker the system administrator may deploy it on other machine, than the *DBPop* works. The memory management is also totally separated from *DBPop* which gives the possibility to optimally adjust system configuration. The following list present the advantages:

- provides data buffer for the writing component
- protects the writing component from processing peaks (when there is a lot of data in one moment to be stored). Such case might end with data writing rejection, because of system overload

- ensures smooth work on its highest pace

There are also disadvantages of this model:

- another component brings the need to configure and maintain it separately
- requires programmer to adjust to new, innovative programming model
- might cause (generally negligible) overhead

The average result of transferring the test data achieved in this scenario was: **129.317s**. The test results are gathered in Section 5.4.

5.2.3. Internal memory message broker (scenario 1.c)

A particular case of the scenario described in previous subsection is an internal memory message broker configuration. In order to tune the performance of the *DBPop* system and mitigate the marshalling overhead on the data transfer when using external JMS broker, it is possible to embed the broker in the same *JVM* that is running *DBPop*. Since Apache ServiceMix incorporates Apache ActiveMQ for its messaging infrastructure it can be easily used by the applications deployed in ServiceMix.

Apache ActiveMQ is optimized for this way of execution. Application configuration needs to be changed to use the `vm://` connector for establishing broker connection instead of any other (TCP, STOMP, OpenWire⁷ etc.). In the configuration with external broker, the application needs to first serialize (also called *marshalling*) the Java object and then send it via network interface to the broker. Broker does the opposite procedure (*unmarshalling*) to reconstruct the data in its *JVM*. Using `vm://` connector all message passing is done by direct Java methods invocations, which does not need any marshalling and in effect is the most efficient way of operating on the *JMS* queue.

On the opposite site, the *JMS* queue resides in the same memory pool as the deployed system (with all of the packages forming *OSGi* environment). The system administrator has to adjust the memory settings to be capable of working all these systems together. This deployment is also more exposed to potential node failure threats.

The average result of transferring the test data achieved in this scenario was: **112.045s**. The test results for the scenarios discussed above are gathered in Section 5.4.

5.3. Scalability

From most modern information systems it is required to be prepared for extension. Extension does not only has to mean functionality extension. The problem, which is tackled by the system can grow. To what extent? This questions rarely has an answer. Nonetheless it is crucial to design the developed system in such way, that it can easily adopt to more demanding throughput. The scalability of the *DBPop* system will be considered in two areas:

- application distribution
- clusterization

7. All supported protocols are listed at <http://activemq.apache.org>

These deployment scenarios reveal the main advantages that come from adopting *Service-Oriented Architecture (SOA)* principles in modern application design. The reader will find out how easy it is to distribute and clustize an application using *Enterprise Service Bus (ESB)* as its backbone.

5.3.1. Distributed DBPop (scenario 2.a)

Building *DBPop* on top of the *Enterprise Service Bus (ESB)* was a strategic choice when it comes to application distribution. Distributed *DBPop* takes advantage of the mediation and routing frameworks of ServiceMix: Camel and ActiveMQ.

It is very easy to deploy a distributed solution using these frameworks. All that needs to be done is to set a proper *OSGi* runtime environment (separate *JVM*) at each considered node and deploy the *DBPop* components into it. What differentiates the centralized and distributed version is that the *EIP* module containing Camel routes was split into independent packages deployed separately in every runtime environment. Also the Commons module needs to be present on each of the modules in order for the *DBPop* components to have access to common domain model. Table 5.6 includes the results of efficiency tests which were performed on the same data set as the standalone solution. The considered scenario is presented in Figure 5.4.

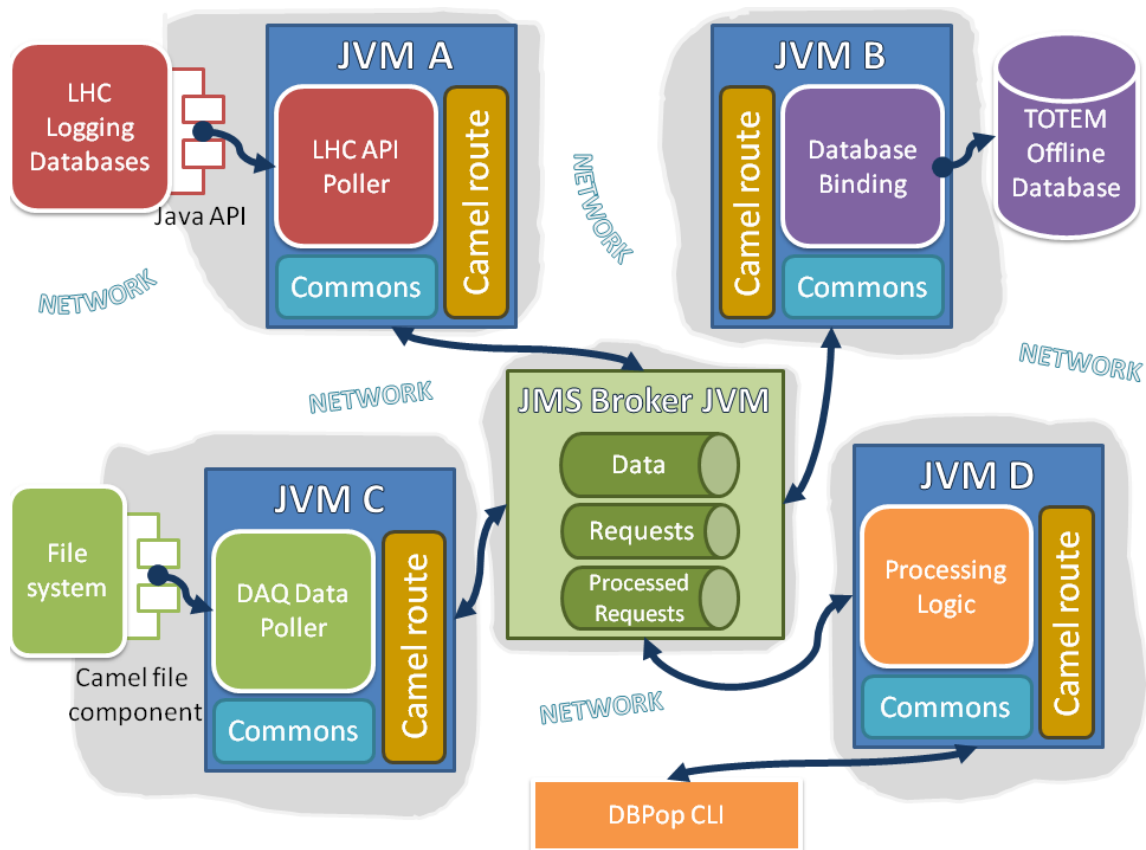


Figure 5.4. Distributed deployment of DBPop. Each of the JVMs hosting DBPop components might be deployed on separate machine (grey spots).

Distributed deployment scenario is characterized with the following features, divided into advantages and disadvantages. The virtues of distributed deployment are:

- Better, single purpose utilization of the nodes machines
- Better performance over centralized *DBPop* deployment
- Improved fault-tolerancy because of separate runtime environments

The disadvantages might include:

- The need to maintain every machine node separately
- Greater administration effort for deployment and monitoring

The average result of transferring the test data achieved in this scenario was: **117.855s**. Please note, that this scenario assumes the usage of *JMS* queues for all the application communication and data transfer, which is the key value of the evaluated system which was depicted in Section 5.2. Another approach would be to use "direct method invocations" mechanisms implemented on top of some distributed computing frameworks. By using these framework one can use services deployed in remote locations in the same way as they would be deployed in the local *OSGi* environment. There are already undertaken works to implement the *OSGi* Remote Service Admin specification, chapter 122 in the *OSGi* 4.2 Enterprise Specification⁸. Examples of implementations are: Apache DOSGi by CXF⁹ or R-OSGi[46].

5.3.2. Application clusterization (scenarios 2.b, 2.c)

Application clusterization is achieved on the level of messaging and mediation layer of ServiceMix, incorporating ActiveMQ and Camel respectively. In order to achieve such configuration one can deploy several instances of the same components in distinct execution environments (might be distributed). Then all that needs to be done is the configuration deployed for Camel and ActiveMQ to support failover, which will redirect non-delivered messages to other components of the system with the same responsibility.

Not only the *DBPop* components are clusterized. Also the *JMS* message broker - ActiveMQ supports clusterized deployment scenario. It is advisable to distribute messaging providers into brokers network¹⁰. In such network, no matter which instance of the broker is used. The message gets directed to the destination by means of ActiveMQ internal routing mechanisms. In case of a node failure, the *failover* option supports redelivery to a running broker.

Table 5.6 includes the test results performed in clusterized scenario. The numbers are self-explanatory. The data provider in the test scenario (LHC Logging Databases with LHCApiPoller) is faster then the DatabaseBinding with TOTEM Offline Database. Additional, redundant DatabaseBinding component deployed on another machine (more general - *JVM*) realizes the load-balancing policy providing concurrent database population. Application is very flexible and can easily be adjusted to upcoming needs and external systems utilization.

8. together with *OSGi Compendium* these specification are extensions to core *OSGi* specification

9. <http://cxf.apache.org/distributed-osgi.html>

10. <http://activemq.apache.org/networks-of-brokers.html>

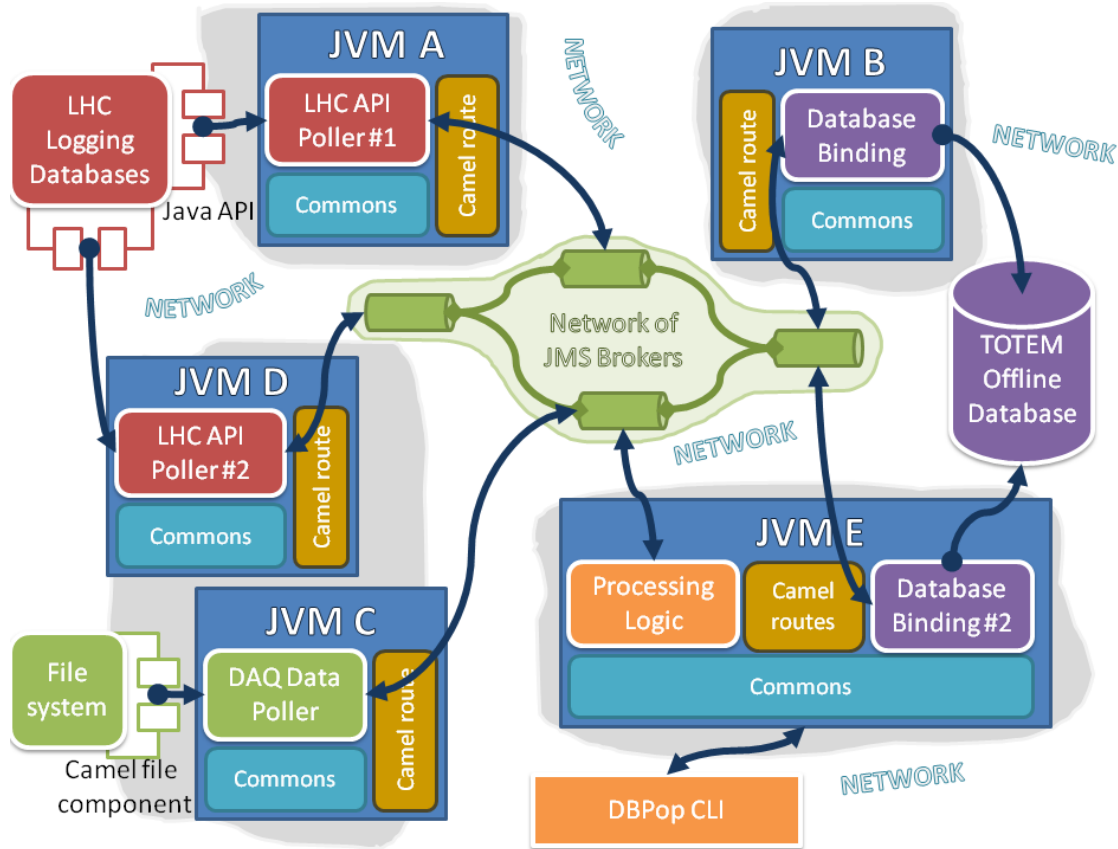


Figure 5.5. Clustrization deployment scenario for DBPop. Each of the components might be deployed in several instances resulting in higher performance and improved fault-tolerancy

Clusterization is the key advantage of using *ESB* in application building over monolithic solutions. It provides several benefits, including:

- improved performance
- improved fault-tolerancy
- optimized external systems utilization
- load-balancing and concurrency

This deployment scenario deals with the data transfer much better, than the previously described variants. The average result of transferring the test data by two simultaneously working DatabaseBinding nodes was: **71.916s**. With three nodes, the result is oscillating around **61.012s**

The disadvantages of this model are similar to the distributed scenario, mainly somewhat larger effort needed to administer the application.

5.4. Tests results

In this section, the test results were gathered together in order to analyze and comment on the system performance. Table 5.6 lists the results for all performed

Scenario	ID	Description	Time[s]	Msrmts/s	MB/s
Single machine	1.a	direct methods invocation	199.371	1345,74	0,80
Single machine	1.b	external broker	129.317	2075,98	1,24
Single machine	1.c	internal broker	112.045	2391,09	1,43
Distributed	2.a	each component on different machine	117.855	2288,96	1,37
Clusterized	2.b	two DatabaseBinding components	71.916	3771,86	2,25
Clusterized	2.c	three DatabaseBinding components	61.012	4390,19	2,63

Table 5.6. Results of efficiency tests for different deployment scenarios.

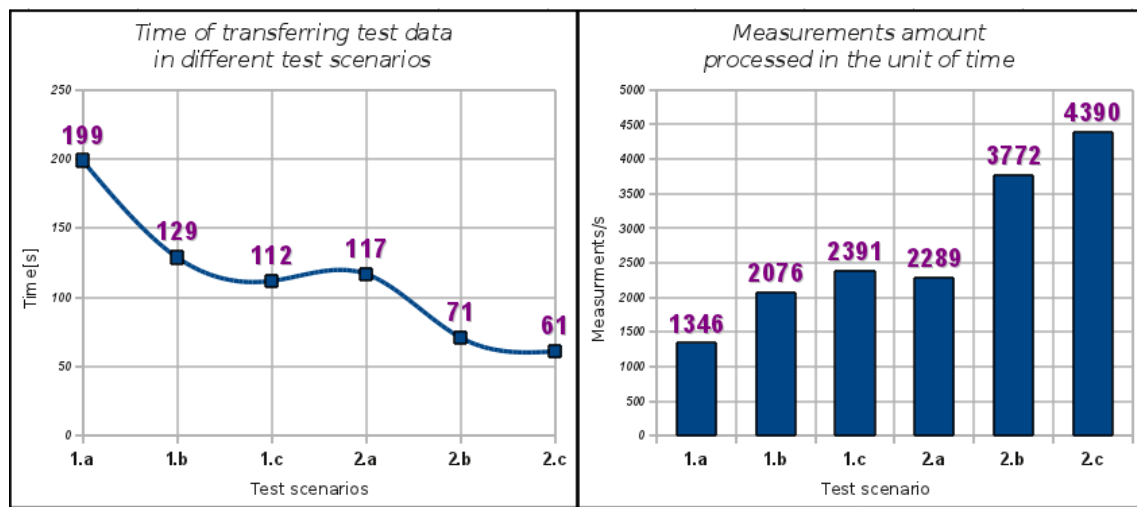


Figure 5.6. Charts presenting system performance, time of data transfer and measurements amount processed in the unit of time.

tests in an order from the simplest to the most sophisticated scenarios. Figure 5.6 illustrates the test numbers graphically.

Basing on the numbers one can draw a couple of noteworthy conclusions:

- the strategy used in *DBPop* using asynchronous queues infrastructure built on top of the *ESB* proves to provide efficient data transfer facilities
- performance of the application can be aided by introducing more processing components and realize load balancing policies
- system throughput can be increased by adding more nodes (source adapters, writing components and *JMS* queues) to the application environment

Using *JMS* queues the system is self-adjusting to the current environmental conditions, like network traffic, performance of the data sources and the database. The *ESB* application brought increased performance, but not without pitfalls - the system developer and administrator need to have great amount of knowledge about the technology being used. The deployment of distributed solution requires more sophisticated system administration and monitoring.

5.5. Other properties evaluation

After evaluating system performance in two main test areas it was desired to investigate other values of considered system. For the following features it is hard to find a strict measurement, since they are non-functional aspects of the *DBPop* system.

5.5.1. Fault-tolerancy

Fault tolerancy of the *DBPop* system can be considered on several levels, including:

- exceptional external systems behaviour
- hardware/operating system crash-down on node running *DBPop*

Exceptional external systems behaviour

The *DBPop* system needs to cooperate with many external systems which provide the measurements data, messaging infrastructure and databases services. Each of this components can run into exceptional conditions resulting in access requests rejection. When any of the components participating in data exchange refuses operating, the Camel route realising this particular exchange tries to redeliver message before throwing exception¹¹. When all redelivery attempts fail, the system administrator is informed of exceptional conditions by an e-mail.

System crash-down

Fault-tolerancy in case of system crash-down on the level of hardware or operating system is directly related to the level of *DBPop* distribution and clustrization. Since the modules of *DBPop* are independent, every component may be deployed in separate *JVM* (even separate machine), which was illustrated in Section 5.3.2. When any of the nodes malfunctions, the failover mechanism of ActiveMQ delivers messages to one of the healthy nodes containing proper module:

```
failover:(vm://broker1:61616,tcp://broker2:61616)?initialReconnectDelay=100
```

Listing 5.2. Failover mechanism for connection to the ActiveMQ JMS Broker

As most of the *DBPop* components are stateless (only ProcessingLogic component is stateful, which might be improved as a future work, discussed in thesis summary), each can be substituted by another one realising the same contract. This approach is exploiting the *SOA* guidelines discussed in Section 2.2.1.

5.5.2. Extendability

It is easy to add new components to the system built on top of the *ESB* architecture. One has to only conform to the philosophy of loosely coupled components and design the additional component to take advantage of the asynchronous infrastructure. *DBPop* system is ready to be extended for other data sources.

Adding new data source

In order to add a new data source the programmer needs to gather full knowledge of how to interface this data source. It might happen, that the data source interface

11. <http://camel.apache.org/dead-letter-channel.html>

would be one from a large scale of endpoints supported by Camel. In such case, the Camel component needs to be properly configured, and a route from the data source to inner *DBPop* infrastructure has to be deployed. Otherwise, a new OSGi service needs to be implemented to meet the *MEP* of the new data source. The ProcessingLogic module needs to be adjusted to support population from new data source as well as the DatabaseBinding to form proper *SQL* statements and place the data in desired table in the TOTEM Offline Database.

5.6. Chapter summary

This chapter begun with test environment description. Hardware, software and external systems configuration were discussed. Next, the test scenario was introduced, which is one of the main use-cases of the *DBPop* system. The meritum part of this chapter were efficiency and scalability tests. Different approaches for data transfer were depicted in order to evaluate the system behaviour. System clustrization as a mean of improving system performance and fault-tolerancy was examined. The key advantages of using asynchronous data transfer infrastructure were presented together with self-explanatory tests results. Other metrics for system evaluation, namely fault-tolerance and extendability were discussed in the last section.

Summary

This chapter verifies the thesis statement by summarizing the content of this manuscript as well as discusses possible directions of future works.

Thesis verification

At the beginning of this thesis, author proposed the following thesis statement:

It is possible to use the ESB facilities to construct effective and scalable integration solution for synchronizing large volumes of measurements data

The implementation works behind this thesis were similar to performing an experiment. From the first point of view, it seemed that applying *ESB* approach to tackle the database population problem in the *TOTEM* experiment would be a matching choice, but not without doubts. In order to prove that proposed statement is correct, a full software development cycle was taken. The process of *System analysis* was complicated because of very scientific domain of the problem - a *High Energy Physics* (*HEP*) experiment. To get the answer, implementation of the system as well as its test deployment were performed. The *DBPop* system is still under development, because it will need to face new usage scenarios, which are foreseen in the future.

Nonetheless for the purpose of this thesis research, the system might be considered as in the final state. Definitely its backbone, which is serving for the basis of this thesis assumptions, meets the requirements against the class of computer systems dealing with the data transfer. The tests described in Chapter 5 finally confirmed the primary assumptions. Using *ESB* provides the means to construct efficient data transfer system, which can easily scale together with the throughput growth. Proposed system architecture can be extended to add new data sources as well as integrated with large variety of external systems by means of mediation and routing frameworks provided by the used *ESB* implementation.

Through the chapters of this thesis it was shown, how to build a modern solution conforming *SOA* principles and how the newest technologies find appliance in building flexible, highly-scalable and performant systems. Details pointed out in this thesis allow the author to conclude, that the correctness of the thesis statement was proved. Nonetheless, author admits, that the decision of applying *ESB* and *SOA* approach in

application building needs to be carefully considered since it requires large awareness of the concepts and technologies which are still in their early state.

Future works

The *TOTEM DBPop* system will definitely be extended to meet new requirements according to growing needs in the *TOTEM* experiment. Currently it supports efficient data transfer from the LHC Logging Databases. At the time of this thesis defense, the works on introducing next data sources are ongoing, which is the basic system extension.

Not all of the tools and technologies, which might have been incorporated in the implementation were known to the author from the beginning of the works. A good example might be the usage of already implemented mechanisms for managing batch jobs, provided by Spring Batch¹². It could be better to follow a standards-based solution than using self-developed pieces of code.

ProcessingLogic auxiliary component - the JobRegistry should be based on a persistent store to improve fault-tolerancy and to eliminate statefulness from the ProcessingLogic component, which is discussed in Section 4.2.3. This change would make the ProcessingLogic component strictly follow the *SOA* principles (Section 2.2.1).

The author of this thesis is not a professional database maintainer and was not a part of the *TOTEM* Offline Database schema creation team. The schema definition might need slight changes when a new, not foreseen data structure will need to be stored in the database. Performance of *DBPop* relies heavily on the database writing component, which is definitely a field for improvements. There is no need for the application to be portable across different database vendors, since Oracle is the exclusive database provider at *CERN*. *JPA* and *Hibernate* usage might be denied in favour of direct usage of *JDBC* and *SQL* language with all of the Oracle's specific features provided by the *JDBC* driver to improve data writing performance.

Final words

The author hopes that the effort put in the described project will be beneficial to the research performed in the *TOTEM* experiment at *CERN*. The manuscript contains enough details, to be used by computer scientists to get introduced into the computing model in the *TOTEM* experiment.

Component approach in computer system building (especially the *SOA* paradigm) proves to be very versatile, which has already been appreciated in the industry. Most of the modern computer systems are nowadays built in a *SOA*-conforming manner. It turns out, that the newest *IT* technologies find appliance also in the scientific area, like the *High Energy Physics (HEP)* experiments. We will see in the nearest future, how far will the progress of *Information Technology (IT)* affect the way the industry works, scientists perform their research and the normal people live their lives. Taking lessons from the past, we have to be prepared for significant changes.

March 2010 - August 2011

12. <http://static.springsource.org/spring-batch/>

Glossary

Data acquisition is the process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer. Data acquisition systems (abbreviated with the acronym DAS or DAQ) typically convert analog waveforms into digital values for processing[47]. 100

Optical Receiver is the most advanced part of the TOTEM *DAQ* system. It is responsible for fast optic to electrical signal transformation. It is build using the FPGA technology. An OptoRX is the part of the *TOTFed*. 101

VMEA data format stands for VERSAmodule Eurocard A. VME is the standard of crates. The abbreviation VMEA does not stand for any specific words, it basically indicates that it VME-a different from VME. it is a data format where events are represented in raw state.. 102

Alignment is the difference between the real location of detector and the one described by ideal geometry. There are three numerical values used to align the detector - it is (u, v, y), Calculating alignment aims to calculate positions of Roman Pots in the *LHC*. As Roman Pots are movable devices it is crucial to have the most precise measurements since it affects the meaning of all collected data by *DAQ* systems. 39, 41, 42

Cross-cutting concerns is an appendix to the main OSGi specification containing many additional services and improvements which might be implemented by the OSGi runtime vendor. The current status of OSGi specification can be easily reviewed at this website: <http://>. 20

Grid is a computing and storage infrastructure built upon systems distributed geographically and connected via computer network, using open standards (protocols, authentication, authorization, services discovery) and providing certain *Quality of Service* (*QoS*). 14

LHC Fill is the complete and coherent set of actions taken while single operation of the *LHC*. An LHC Fill comes through a chain of phases called Beam Modes. Beam Modes vary from INJPROT, SETUP, through STABLE beam and BEAMDUMP. . 37, 55, 73, 81

Luminosity describes the collision rate of particles in the accelerator. Luminosity is the number of particles per unit area per unit time times the opacity of the target, usually expressed in the cgs units $cm^{-2}s^{-1}$. The integrated luminosity is the integral of the luminosity with respect to time. The luminosity is an important value to characterize the performance of an accelerator.. 8, 12

OSGi Compendium is an appendix to the main *OSGi* specification containing many additional services and improvements which might be implemented by the *OSGi* runtime vendor. The current status of *OSGi* specification can be easily reviewed at this website: <http://www.osgi.org/Specifications/>. 54, 87

ROOT is an object-oriented program and library developed by *CERN*. It was originally designed for particle physics data analysis and contains several features specific to this field, but it is also used in other applications such as astronomy and data mining. 34, 37, 39

Standard Model is a physics theory concerning the electromagnetic, weak, and strong nuclear interactions, which mediate the dynamics of the known subatomic particles. This theory includes: strong interactions due to the color charges of quarks and gluons, a combined theory of weak and electromagnetic interaction, known as electroweak theory, that introduces W and Z bosons as the carrier particles of weak processes, and photons as mediators to electromagnetic interactions.. 8

System analysis is a process of defining basic values of a designed computer system. Includes requirements analysis (encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users), functional analysis (defines the behavioral aspects of designed system) and feasibility study. 43, 92

TOTEM Counting-room is the operational room located in close neighbourhood to the *TOTEM* detectors and the *LHC*. It is located Underground near the Interaction Point5 site and has direct connection to *TOTEM DAQ* systems.. 33, 37

TOTEM Run is the complete set of actions performed while *TOTEM* data-taking, which includes the data taking by the *TOTEM DAQ* systems acquiring data from detectors: T1, T2 and Roman Pots.. 34, 37, 40, 41, 81, 82

Trigger is something that causes a data acquisition system to start collecting data. It may be as simple as pressing a software button or a set of conditions which when met trigger data capture (internal triggers), or an externally generated, hardware signal (an external trigger). Trigger is dedicated to filter out the considered useless signals in order to reduce the data stream to a manageable rate. 13

Acronyms

AFS *Andrew File System*. 14, 35
AOP *Aspect-Oriented Programming*. 20
API *Application Programming Interface*. 23, 27, 29–31, 38, 40, 47, 54, 55, 60, 62, 72
ASIC *Application Specific Integrated Circuit*. 13
BLM *Beam Loss Monitor*. 40, 74, 82
BPM *Beam Position Monitor*. 40, 74, 82
BQM *Beam Quality Monitor*. 74, 82
CASTOR *CERN Advanced STORage manager*. 14, 16, 34, 37
CERN *European Organization for Nuclear Research*. 8, 10–14, 32, 33, 35, 36, 48, 50, 55, 59, 71, 72, 76–80, 93, 99
CLI *Command-Line Interface*. 51–53, 57, 59, 72–74, 77
CMSSW *Compact Muon Solenoid SoftWare*. 37, 39
CMS *Compact Muon Solenoid*. 15, 16, 32, 39
CORBA *Common Object Request Broker Architecture*. 30
CSV *Comma Separated Values*. 39
DAQ *Data acquisition*. 13, 16, 33, 35, 37, 39, 43, 67, 98, 99
DBPop *TOTEM DataBase POPulation System*. 11, 31–33, 39, 41–46, 49–55, 57–67, 71–82, 84–87, 89–93
DCS *Detector Control System*. 35
DDL *Data Description Language*. 71
DDR *Double Data Rate*. 77
DI *Dependency Injection*. 25, 26, 29, 54
DPR *Data Population Request*. 51–53, 55, 59, 60, 62–64, 66
DSL *Domain Specific Language*. 24, 65
ECC *Error Correcting Code*. 77
EIP *Enterprise Integration Patterns*. 24, 25, 28, 32, 65, 86
EJB *Enterprise Java Beans*. 27
ERD *Entity Relationship Diagram*. 27
ESB *Enterprise Service Bus*. 10, 11, 18–21, 24, 28–32, 48–52, 63, 64, 72, 74, 76, 86, 88–90, 92
FPGA *Field Programmable Gate Array*. 13
FTP *File Transfer Protocol*. 32, 65

GC Garbage Collector. 79
GOL Gigabit Optical Link. 33
GPN General Purpose Network. 78
GUI Graphical User Interface. 42, 59, 72
HEP High Energy Physics. 10–12, 32, 33, 41, 92
HSM Hierarchical Storage Management. 14, 34
HTTP HyperText Transfer Protocol. 30, 48, 65
IDE Integrated Development Environment. 72
INFN Istituto Nazionale di Fisica Nucleare. 34
IOV Interval Of Validity. 36, 40, 42, 66, 67, 71
IT Information Technology. 17, 26, 35, 77, 80
IoC Inversion of Control. 25, 26, 29, 57, 60
JAR Java ARchive. 21
JAXB Java Architecture for XML Binding. 61
JB Java Business Integration. 20, 30
JBOD Just a Bunch Of Disks. 77
JDBC Java DataBase Connectivity. 38, 59, 81, 93
JDK Java Development Kit. 79, 81
JEE Java Enterprise Edition. 20, 22, 23
JMS Java Message Service. 22–25, 29, 30, 46, 48, 52, 53, 60, 62–65, 74, 81, 85, 87, 89
JMX Java Management eXtensions. 21, 60, 73
JNDI Java Naming and Directory Interface. 23, 30, 62
JPA Java Persistence API. 25, 27, 29, 57, 71, 93
JVM Java Virtual Machine. 21, 65, 79, 81, 84–87, 90
LCG LHC Computing Grid. 78
LDB Logging DataBase. 38, 40
LDEC Logging Data Extractor Client. 55
LHC Large Hadron Collider. 8–10, 13, 15, 16, 37, 65, 72, 73, 78, 80, 82, 98, 99
MDB Measurements DataBase. 37, 40
MEP Message Exchange Pattern. 46, 48, 91
MOM Message-Oriented Middleware. 10, 22, 29, 79, 84
OOP Object-Oriented Programming. 25
OO Object-Orientation. 17, 24
ORM Object-Relational Mapping. 27, 29, 32, 44, 57, 71
OSGi Open Services Gateway initiative. 20, 21, 28, 30, 52–55, 57, 65, 72, 74, 85–87, 99
OptoRX Optical Receiver. 33, 98
POJO Plain Old Java Object. 26, 52, 54, 62
POM Project Object Model. 71, 72
PSB Proton Synchrotron Booster. 9
PS Proton Synchrotron. 9
QoS Quality of Service. 98
RAC Real Application Clusters. 78, 80
RAID Redundant Array of Independent Disks. 77
RHEL Red Hat Enterprise Linux. 79

-
- RI** *Reference Implementation.* 54
- RMI** *Remote Method Invocation.* 21, 40, 73
- RPC** *Remote Procedure Call.* 21
- RP** *Roman Pot.* 39
- SAN** *Storage Area Network.* 77
- SATA** *Serial Advanced Technology Attachment.* 77
- SLC** *Scientific Linux CERN.* 79
- SMP** *Symmetric Multiprocessing.* 79
- SOAP** *Simple Object Access Protocol.* 30
- SOA** *Service-Oriented Architecture.* 16–18, 20, 28, 32, 49, 54, 60, 86, 90, 92, 93
- SPS** *Super Proton Synchrotron.* 9
- SQL** *Structured Query Language.* 27, 28, 38, 40, 46, 58, 63, 82, 91, 93
- SSH** *Secure SHell.* 72
- SVN** *Subversion.* 72
- TN** *Technical Network.* 78
- TOTEM** *TOTal Elastic Scattering and Diffraction Measurement.* 10–12, 14–16, 31–35, 39, 41, 43, 48, 50, 62, 66, 67, 71, 72, 75, 76, 79, 82, 92, 93, 99
- TOTFed** *TOTEM Front End Driver.* 33, 98
- UI** *User Interface.* 60
- URI** *Uniform Resource Identifier.* 24, 65
- VFAT** *Very Forward ATLAS and TOTEM chip.* 33
- VMEA** *VMEA data format.* 34, 37, 98
- VNC** *Virtual Network Computing.* 72
- XML** *eXtensible Markup Language.* 24, 26, 27, 30, 39, 41, 54, 55, 61, 66, 71

List of Listings

4.1	The DataPopulationRequest class fields.	53
4.2	The DataContainer class fields.	53
4.3	Example Spring Dynamic Modules for OSGi configuration file.	55
4.4	Example XML configuration file for the LHC Logging measurements.	57
4.5	The DatabaseBinding interface.	58
4.6	General overview of the data writing mechanism in DatabaseBinding component.	58
4.7	The ProcessingLogicMBean interface main methods.	60
4.8	Example Camel route.	66
4.9	PollData Camel route periodically issuing data population.	66
4.10	ProcessRequests Camel route distributing DPR's to the data source adapters and placing data on the buffer queue.	67
4.11	FromQueueToDatabase Camel route realizing data writing to the database and notification of finished jobs.	67
4.12	DBPop CLI commands list.	74
4.13	DBPopCLI usage example - issuing batch (asynchronous) job	75
4.14	DBPopCLI usage example - issuing interactive (synchronous) job	75
4.15	DBPopCLI usage example - populate data for LHC fill	75
4.16	DBPopCLI usage example - querying job status	75
5.1	Direct methods invocation algorithm	84
5.2	Failover mechanism for connection to the ActiveMQ JMS Broker	91

List of Figures

1.1	LHC filling accelerator chain and experiments' location	9
2.1	Tier architecture in data processing at the LHC	14
2.2	TOTEM detectors location at the LHC	15
2.3	Find-bind-execute paradigm as the basic way of interactions between services in SOA	17
2.4	An abstract, graphical view of the Enterprise Service Bus acting as mediation middleware connecting Services together	18
2.5	Mediation between Services by means of ESB	19
2.6	JMS publisher/subscriber and point-to-point exchange models	22
2.7	Elements of the JMS architecture and relationships between them	23
2.8	Graph of the main concepts of the thesis presenting relations between them. . .	28
2.9	Main components of ServiceMix providing ESB facilities.	29
2.10	Architecture of Apache Karaf	30
3.1	Current data processing model in the TOTEM experiment.	34
3.2	Processing model with the central data storage - the TOTEM Offline Database. .	36
3.3	LHC Logging Databases data sources aggregation	38
3.4	LHC Logging Java API database isolation concept	39
3.5	Data route of measurements coming from LHC Logging Databases.	40
3.6	Data route of the DAQ-related data to the TOTEM Offline Database.	41
3.7	Data route for the intermediate calculations done by means of the TOTEM Offline Software.	42
3.8	Use-cases diagram for the TOTEM DBPop system.	45
3.9	Funnel model of the designed system	46
3.10	High-level view on the designed system with basic division into modules. . . .	47
3.11	System architecture using Enterprise Service Bus as its backbone.	49
4.1	DBPop architecture using Apache ServiceMix 4.x	54
4.2	Sequence diagram of the LHC API Poller module of DBPop.	57
4.3	Sequence diagram of the DatabaseBinding module.	59
4.4	Sequence diagram of the ProcessingLogic module.	62
4.5	Data transfer strategies.	65
4.6	The concept of Interval of Validity	68
4.7	TOTEM Offline database schema in August 2011.	69
5.1	Simplified network topology at CERN with test infrastructure	79
5.2	LHC Fill, Beam Modes and TOTEM Run relations.	82

5.3	Efficiency tests scenario	84
5.4	Distributed deployment of DBPop	87
5.5	Clustrization deployment scenario for DBPop	89
5.6	Charts presenting system performance	90

List of Tables

2.1	What SOA is and what SOA is not	18
2.2	An example route in notion of Enterprise Integration Patterns	25
4.1	T7IovType table details.	70
4.2	T3ValidityInterval table details.	70
4.3	T15MeasurementType table details.	71
4.4	T18GeneralMeasurement table details.	71
5.1	Test machines configuration.	78
5.2	Machine configuration hosting TOTEM Offline Database in the IT Department at CERN.	78
5.3	Configuration of ActiveMQ JMS Broker used in tests scenario.	81
5.4	Configuration of the DatabaseBinding component used in efficiency tests.	81
5.5	List of measurements acquired from LHC Logging Databases while one hour of STABLE beam.	83
5.6	Results of efficiency tests for different deployment scenarios.	90