SpagoBI Architectural Design

Author Grazia Cazzin

Index

Version

Version/Release n° :	0.5	Data Version/Release :	August, 5 th 2005
Update description:	Draft – This is the first version released in the project's site.		

Document goal

This document provides the architectural design of SpagoBI platform, the description of all its components and their responsibilities. It draws the SpagoBI development guidelines.

This document is a work-in-progress: it's likely to have further and deep reviews. The sections "in progress" are pointed out.

References

Some of the concepts of this document refer to the following documentation:

[1] BAUER, C., KING, G., *Hibernate in Action*, Manning Pubblications, Greenwich 2004.

[2] BERRY, M. J. A., LINOFF, G. A. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management,* Wiley Publishing, New York 2004[2].

[3] HALPIN, T.,. *Information Modeling and Relational Databases. Form conceptual analysis to logical design*, Morgan Kauffman Publisher, San Francisco 2001.

[4] KIMBALL, R., REEVES, L., ROSS, M., THORNTHWAITE, W., *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*, Wiley Publishing, USA 1998.

[5] KIMBALL, R., Ross, M., *The data warehouse toolkit*, Wiley, New York 2000[2].

[6] KIMBALL, R., Merz, R., *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse*, Wiley Publishing, New York 2000.

[7] KIMBALL, R., CASERTA, J., *The Data WarehouseETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*, Wiley Publishing, New York 2004.

[8] MARCO, D., *Building and Managing the Meta Data Repository: A Full Lifecycle Guide,* Wiley Publishing, New York 2000.

[9] RICHARDSON, W.C., AVONDOLIO, D., VITALE, J., LEN, P., SMITH, K.T., *Professional Development With Open Source Tools*, Wrox, Indianapolis 2004.

[10] SILVERSTON, L., *The Data Model Resource Book*, Wiley Publishing, New York 2001.

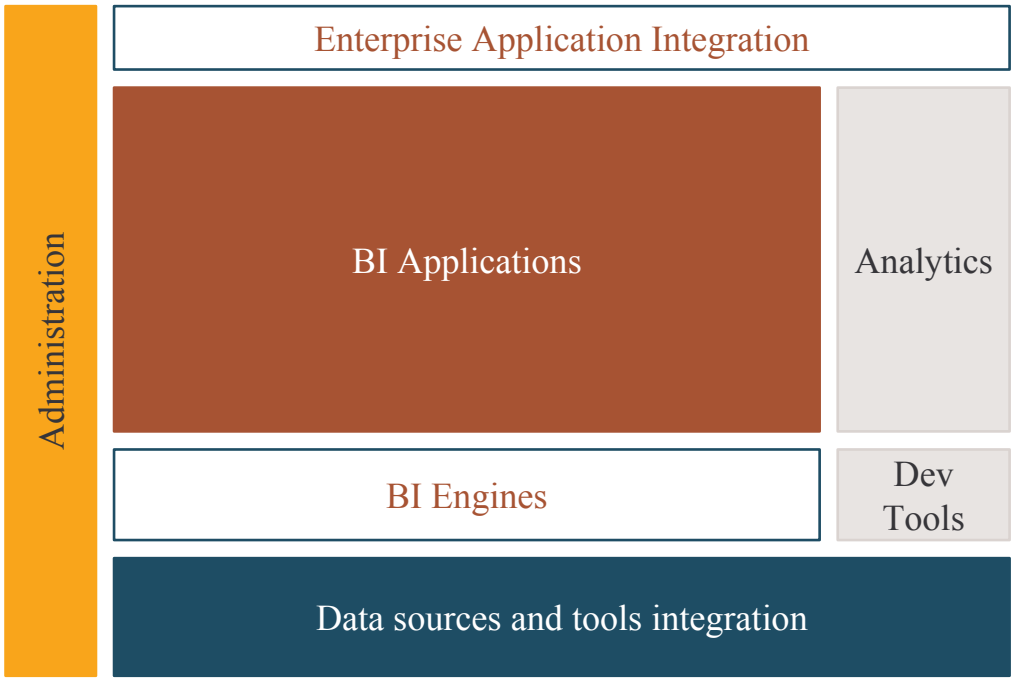
[11] WITTEN, I.H., FRANK, E., *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kauffman Publisher, 2000.

Information about the Spago framework are available in the Spago Project site (http://spago.eng.it).

1 Functional level architecture

A Business Intelligence platform provides all the necessary tools in order to manage and deliver documents, analysis models, control and forecast systems. Every kind of document maintains its features, but everyone integrates itself in a common infrastructure which provides the right application level and execution environment.

The functional SpagoBI architecture follows.



Picture 1 – Functional SpagoBI architecture

The **Enterprise Application Integration** is related to the integration with generic portals; they can be specific BI portals or generic enterprise portal services. It allows SpagoBI to manage usable Business Intelligence services.

The **BI Application** supports the analytical core of the platform. It manages in a similar way the production of the results validation cycles, the parametric activation, the navigation, the versioning and storage, although every BI object maintains its distinctive feautures. It supports a common strategy for the parameters management, the activation and interaction modality, the visibility and authorizations policies and rules, organization and navigation across the different documents.

The **BI Engines** are the interfaces for the engines which realizes the analytical documents.

The **Data sources and tools integration** are related to the integration with source systems and to the all platform's aspects and used tools, in order to achieve data and metadata integration.

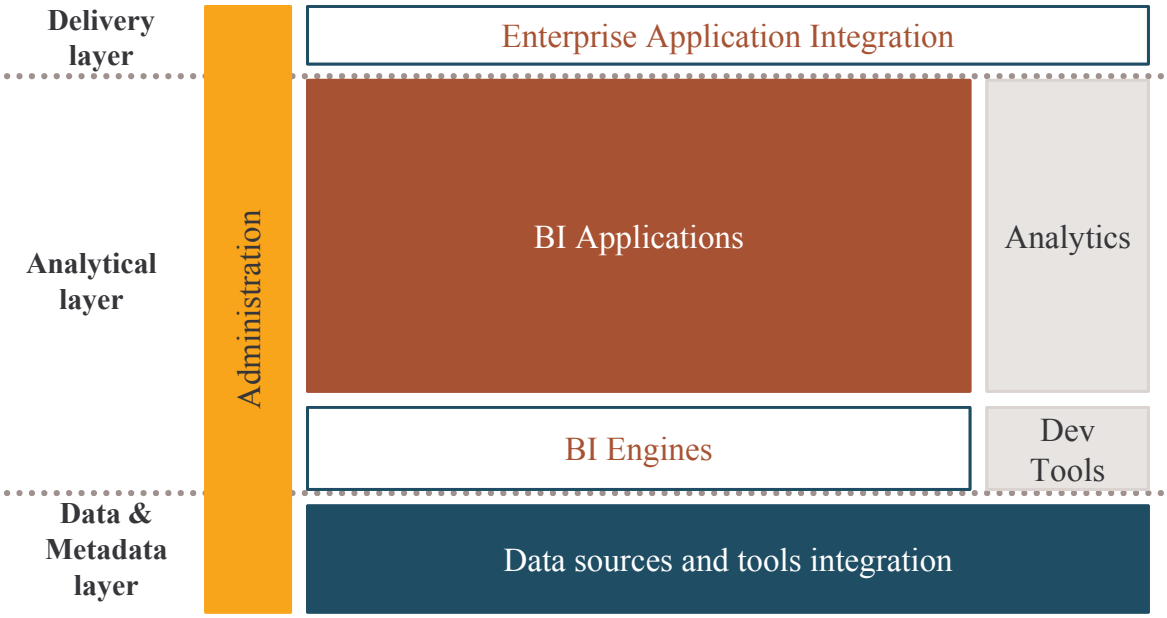
The **Administration** provides support for the management functionalities of the whole platform, like scheduling, workflow design, logging and auditing services, users profiling interface and so on.

The **Development tools** are the specific products used for designing Business Intelligence documents. SpagoBI hasn't a prefixed set of development tools; the user can choose his favourite ones.

The **Analytics**, as a thematic model (like CRM, HR, Budget, Sales, etc.), are out of the project scope, as well as development tools. They will be the result of single projects which can evolve to general models.

So, apart from *Development Tools* and *Analytics*, a layered functional architecture view is the following:

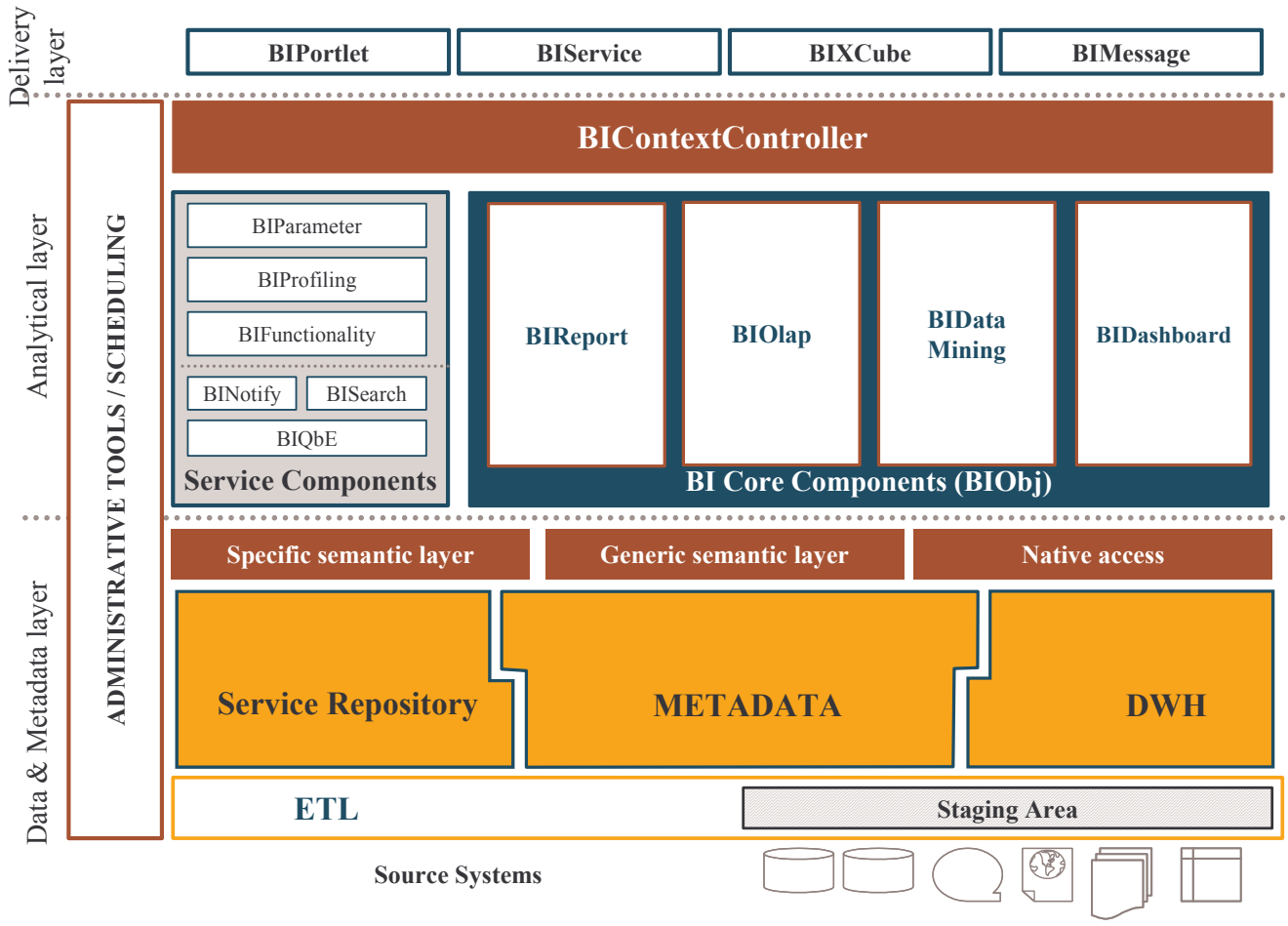
- **Delivery layer** realizing *Enterprise Application Integration* (output integration);
- **Analytical layer** realizing *BI Applications* and *BI Enginees* interfaces (the platform core);
- **Data and metadata layer** solving all *Data Source and tools integration* responsibilities (the input and integration core).



Picture 2 – SpagoBI's layers

2 Application level architecture

Every layer of functional architecture is composed by means of different application modules, as shown in the detailed picture and explained in this chapter.



Picture 3 – SpagoBI application level architecture

2.1 Delivery Layer

The *Delivery Layer* main goals are both to make the Business Intelligence services usable in different ways and to give to the external application the capability to interact with SpagoBI. It supports different channel/protocols:

- **Portlet** (HTTP to portlet container), exposing Business Intelligence services managed by SpagoBI
- **Web Services** (over SOAP/HTTP protocol) for Business Intelligences services invocation from enterprise portal and applications

- **JMS** for platform interaction by JMS messages.

These implementations are based on Spago's adapters (*AdapterPortlet*, *AdapterSOAP*, *AdapterJMS*).

SpagoBI's services and interactions are useful only with some of the available channels/protocols. In the next chapters the allowed channel/service and channel/interaction combinations will be outlined.

2.1.1 BIPortlet

All main BI services are exposed via portlet, according to the JSR 168 specification. It's possible to build your custom interface in a new Business Intelligence Portal, or as a part of an existing one, by including the necessary portlets.

The first available portlets, everyone deployed in the same WebApplication, are:

- **SBISettings**: it offers the necessary services to manage the technological platform configuration to a portal administrator. It provides for the engines configuration, the functional tree management and the extraordinary maintenance of the documents configuration and state.
- **SBIDevelopmentContext**: it enables the services for BI Objects configuration to be used in a portal setting. It's responsible for setting the execution context of the BI services (ex: the parameters definition and the use mode, the control rules, the category locations and the documents profiling). For further information, see to the section related to the services exposed by this portlet.
- **SBIFunctionality**: it offers the functional view of the documents registered in SpagoBI platform and allows their parametric activation. It makes available the core of a BI solution in a portal setting. This portlet works in two different ways:

  for the test users, it only filters the document in TEST state and can activate them simulating all the expected final users' roles. If the test is successful, the document can be released for the end-user.

  for the end-users, the portlet filters the document in RELEASE state and according to his functional roles. In this case, you can navigate through the category tree, up to reach the documents which realize a specific functionality. Here you can execute it, either a Report, a OLAP, a Data Mining model or a Dashboard view. You can either navigate through the documents or shift from a document to another inheriting its settings.

- **BIObjSearch**.

  In progress.

- **BIQbE**.

  In progress.

2.1.2 BIService

You can use Web Services to interact with other enterprise applications or portals.

In progress.

2.1.3 BIXCube

A simple transmission of the information and the results is made easier by means of XML structures and models.

In progress.

2.1.4 BIMessage

JMS messages enables the analytical process; it also highlight the critical conditions.

In progress.

2.2 Analytical layer

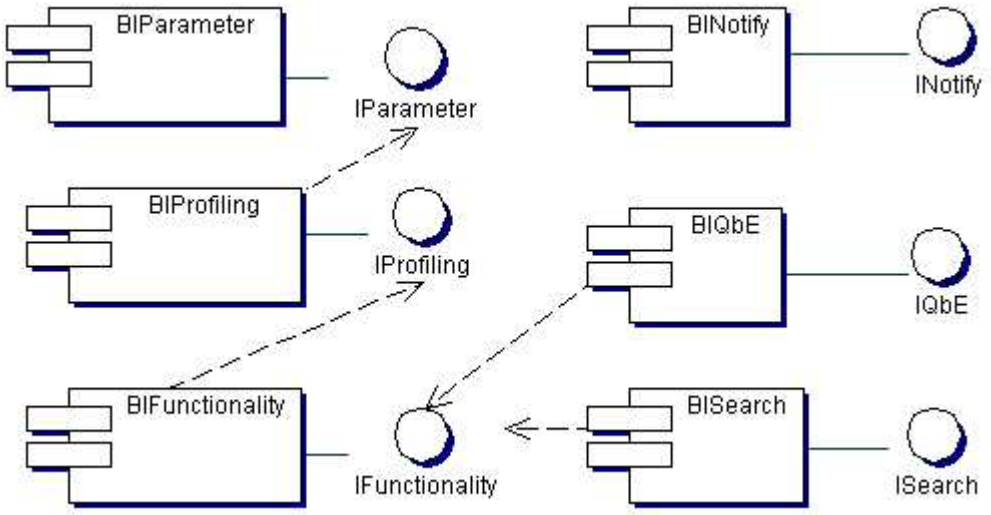
The **Analytical Layer** is the platform core which coordinates all the analytical activities supplying their supporting tools. Its main components are Report, OLAP, Data Mining, Dashboard and Scorecards modules: each of them corresponds to a specific functionality of the same architectural setting: this layer allows an easy learning and a modular use.

Some service components support the core effectiveness preparing the working environment in its collateral aspects: parameters unified management, filters and domains, query by example capabilities, engines and object profiling systems, structuring of categories for documents classification, documents storage and search, approval and management workflow.

The controller takes the responsibility for a successful coordination of the core components interaction and of their interaction with the service components.

2.2.1 Service Components

In the *analytical layer*, the **service components** don't correspond to an analytical task (unlike OLAP or Data Mining), but they support and make easier the process. They are described in the following.



Picture 4 – Service Components

2.2.1.1 BIParameter

BIParameter takes the responsibility to manage the input values for the analytical modules startup and navigation; it also regulates the data visibility according to users' roles. So, *BIParameter* component enables:

- input modality definition (manual input, predefined list of values, values from lookup tables or view, default value and so on);
- validation rules to check the inserted value;
- parameter's behaviour definition (its presentation form and value checks modality) in relation with user's roles.

BIParameter is an independent component; it can interact with any analytical one's (report, OLAP, Data Mining, Dashboard and scorecard - see at the corresponding section for details -)

2.2.1.2 BIProfiling

BIProfiling is responsible for the analytical documents registration. Every analytical module can be realized with different tools and it can use different engines to run. For example, you can have some reports running on JasperReport (FOSS product) together with others reports working on CrystalReport (proprietary product).

So, the *BIProfiling* component enables:

- engines registration with standardized information set, useful for its execution;

- analytical document registration. For every document (that is already developed) some standardized information are kept, including the specific engine to use and the parameter input to receive.

BIProfiling depends on *BIParameter* component for the parameters' interface.

2.2.1.3 BIFunctionality

BIFunctionality takes the responsibility to define and to manage the Business Intelligence services realized on the SpagoBI platform.

In a enterprise portal, the services offered are clear: for example, a banking portal will have functionalities such as 'transfer management', 'accounting operations list', 'trading'. Moreover they can be organized in other sub-functionalities.

Otherwise, usually in a Business Intelligence portal the functionalities aren't so defined. You can define services such as 'Sales analysis', 'Budget', 'Marketing models', 'Simulation area'. Each of them collects many detailed documents like 'sale trend forecast for product industry', 'annual sales reporting', 'sales deviation from plan' and so on.

BIFunctionality manages the structuring of a functional tree providing for:

- the tree creation and maintenance;
- the production of a generic analytical documents' keeping, under the tree lowest level (of every type, also Data Mining models too);
- the visibility and autority regulation on the users roles base;
- the maintenance of the historical versions for the relevant documents;
- the approval and certification procedure for the documents' production, in cooperation with the workflow supplied by the administration tools (looks later for additional information).

BIFunctionality depends from the *BIProfiling* component for analytical documents' interface.

2.2.1.4 BISearch

BISearch is a documental search engine component.

In progress.

2.2.1.5 BIQbE

BIQbE is a component for inquiring of data in a semantic form. It offers a visual mode for data inquiring. It enables the saving of a template structure for subsequent reports development, or the results' exporting for external elaborations (ex: cvs, XML).

In progress.

2.2.1.6 BINotify

BINotify is an alerting and events notification component.

In progress.

2.2.2 CORE COMPONENTS

The SpagoBI platform's heart and its analytical center are the modules directly faced to the datawarehouse inquiring and to its stored information analysing.

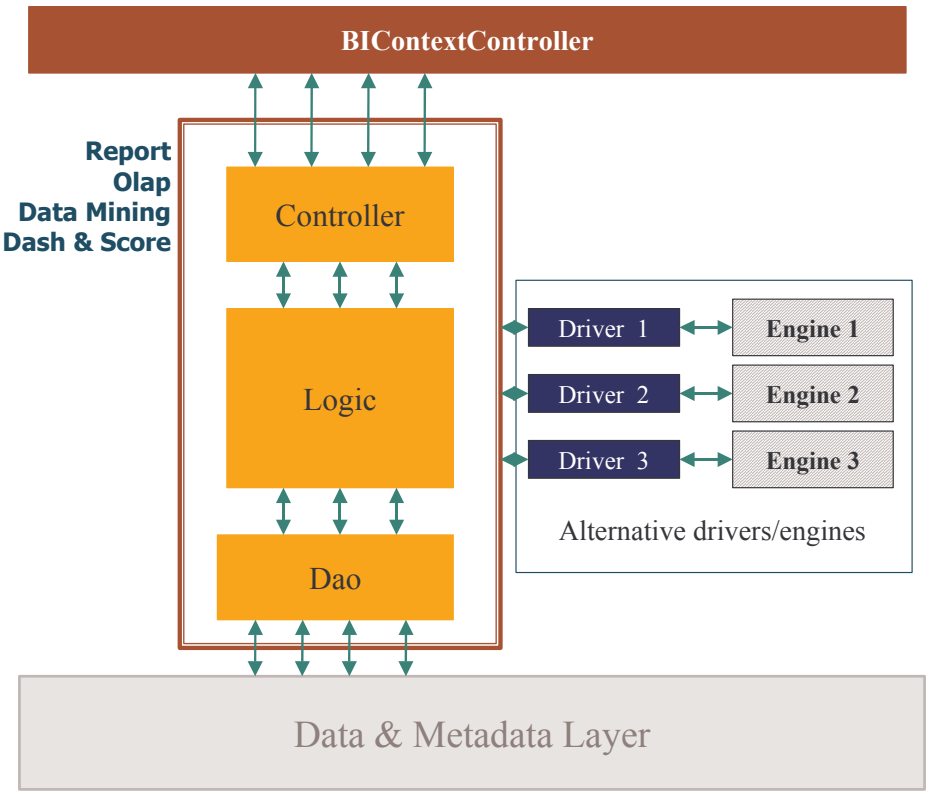
Reporting, OLAP Analysis, Data Mining, Dashboards and Scorecards are the BI core components of SpagoBI. They have an interface layer and a common architecture; they manage their own application level context in a specific way.

In more details:

- **Reports** realize the structured information views; they have a greater diffusion level according to a static structure (.pdf, .xls, .csv, .html, etc). SpagoBI enables the navigation capability between different reports, inheriting the settings.
- The multidimensional structures for the **OLAP analysis** add a higher degree of freedom and variability. The analysis axis and the observation measures are structured. This enables the obtaining of data examination at various detail levels and from various perspectives, by means of drill-down, drill-across, slice and dice operations.
- **Data Mining** algorithms and processes (Neural Networks, Decision Trees, etc.) will enable the data analysis, with the aim to find out the hidden information. SpagoBI supports Data Mining models' implementations and their results' analysis trough the other Business Intelligence objects.
- In the **Management Performance** context, SpagoBI provides many widgets for the dashboards structuring and the parametric evaluation of performance scores.

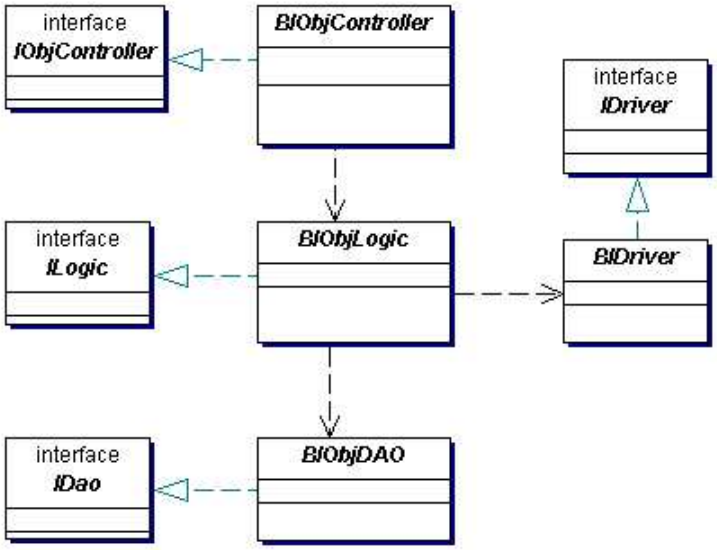
The architecture aims don't concern the analytical objects development's instruments. You can draw a report using a tool for JasperReport or by means of the CrystalReport designer. SpagoBI doesn't provide the specific tool but it's able to manage their different configuration and execution phase.

The common architecture for all the core components (generically called *BIObj* in the sequel) is the following:



Picture 5 – Core Components interaction

According to the classes, the picture is:



Picture 6 – Core Component classes

An abstract definition of the components generalizes the common behaviours and the interactions with the other architectural layers; then, the different implementations specialize their own work logics.

A *BIObj* can work in three different ways:

- by totally delegating the execution and the management of the interaction to the specify engine (DELEGATED mode);
- by keeping the control logic of the execution and by driving the engine interaction (SUPERVISED mode);
- by directly implementing the analytical logic, without using outside engines (PROPRIETARY mode).

According to the chosen working mode, the component's classes have a different weight and are composed by different tasks.

2.2.2.1 BIObjController

The *BIObjController* is the abstract definition of a delegated structure which maintains the control on a specific analytical area related to the general context. Cooperating with the *BIContextController* (look after for its description), this object turns the generic parameters into specific ones that are suitable for their managed analytical area. It also returns some additional data or new settings to the *BIContextController*.

Independently from the *BIObj* execution mode, the *BIObjController* takes the following responsibility:

- it gives a unique interface to the *BIContextController* (i.e. Façade pattern);
- it manages a specific interface toward the *BiObjLogic;*
- it manages the *BIObj*'s state according to the approval and certification workflow;
- it translates the general setting received from the *BIContextController* into specific ones that are understandable from the specific component.

Furthermore, in the SUPERVISED and PROPRIETARY execution way, it must:

- supervise the specific logic of the analytical components;
- manage the user interaction in its analytical area;
- register and give back the new settings to the *BIContextController* when final user changes it.

BIObjController depends on *BIContextController* component for the general context interface (in a bi-directional relationship) and from *BIObjLogic* class for the analytical logic execution.

2.2.2.2 BIObjLogic

The *BIObjLogic* class realizes the specific logics of an analytical area. It works in a different way according to the *BIObj* behaviour mode:

- in DELEGATED mode, it only turns the *BIObjController* request to the specific driver;

- in SUPERVISED mode, it interacts with the specific engine through its driver, in order to realize the analytical functions of its competence;
- in PROPRIETARY mode, it directly implements the analytical functions of its competence, without the drivers support.

BIObjLogic depends on *BIDriver* for the general engine's interface and on *BIDao* for generic Data Layer interface.

2.2.2.3 BIObjDAO

The *BIObjDao* class takes the responsibility to interact with the data layer. It has a relevant role only in the proprietary mode of BI component execution, because:

- in DELEGATED and SUPERVISED mode, the specific engine directly manages the data layer access through the specific semantic layer;
- in PROPRIETARY mode, it access the data warehouse using the rdbms analytical features directly (if there are) or through the generic semantic layer.

BIObjDao interfaces the chosen rdbms by means of the generic semantic layer.

2.2.2.4 BIDriver

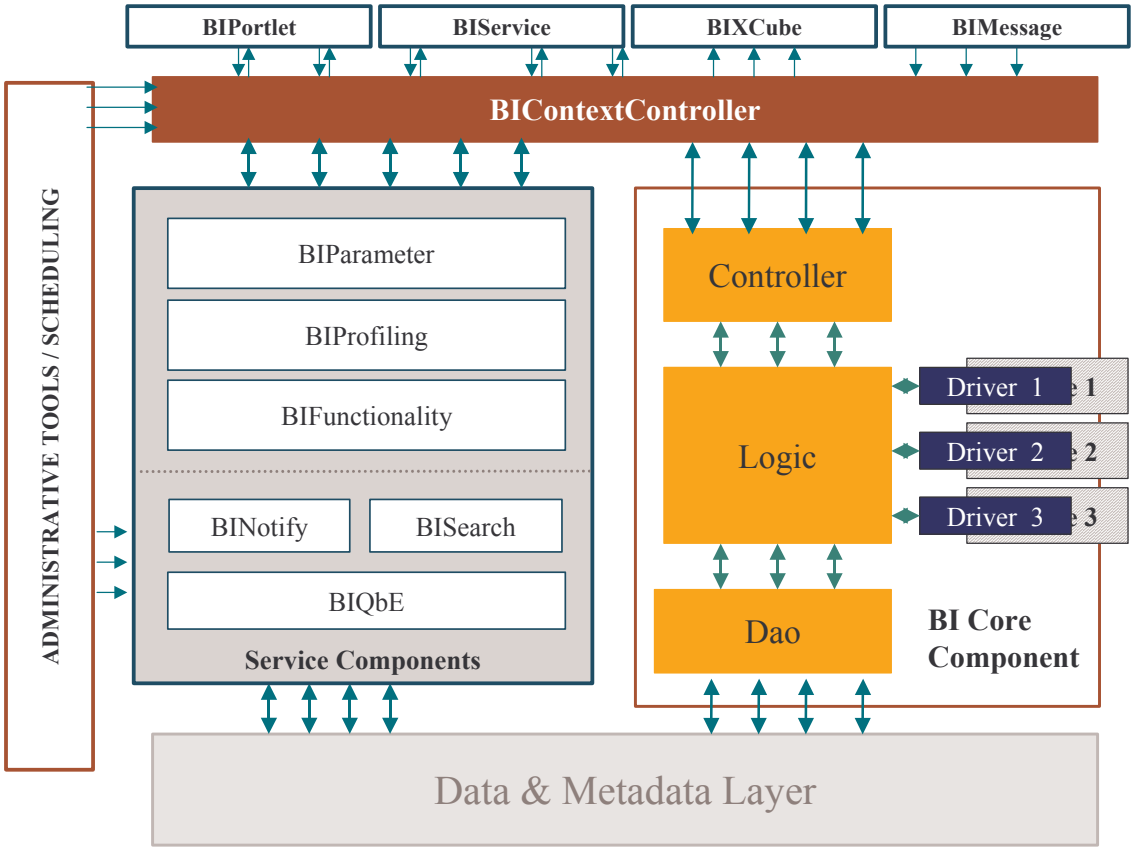
The *BIDriver* class takes the responsibility for the engine's interfacing, providing the translation of the general analytical method into a specific engine invocation. It works like this:

- it translates the parameters' values in a QueryString format;
- it extents the QueryString with the specific engines' parameters;
- it creates the engine execution context;
- it invokes the engine.

BIDriver depends on the particular engine which it works with.

2.2.3 BIContextController

The *BIContextController* acts as a focal point for the coordinating operations across the whole platform. It manages the parameters exchange between the objects ensuring the values' heredity during the user driven navigation.



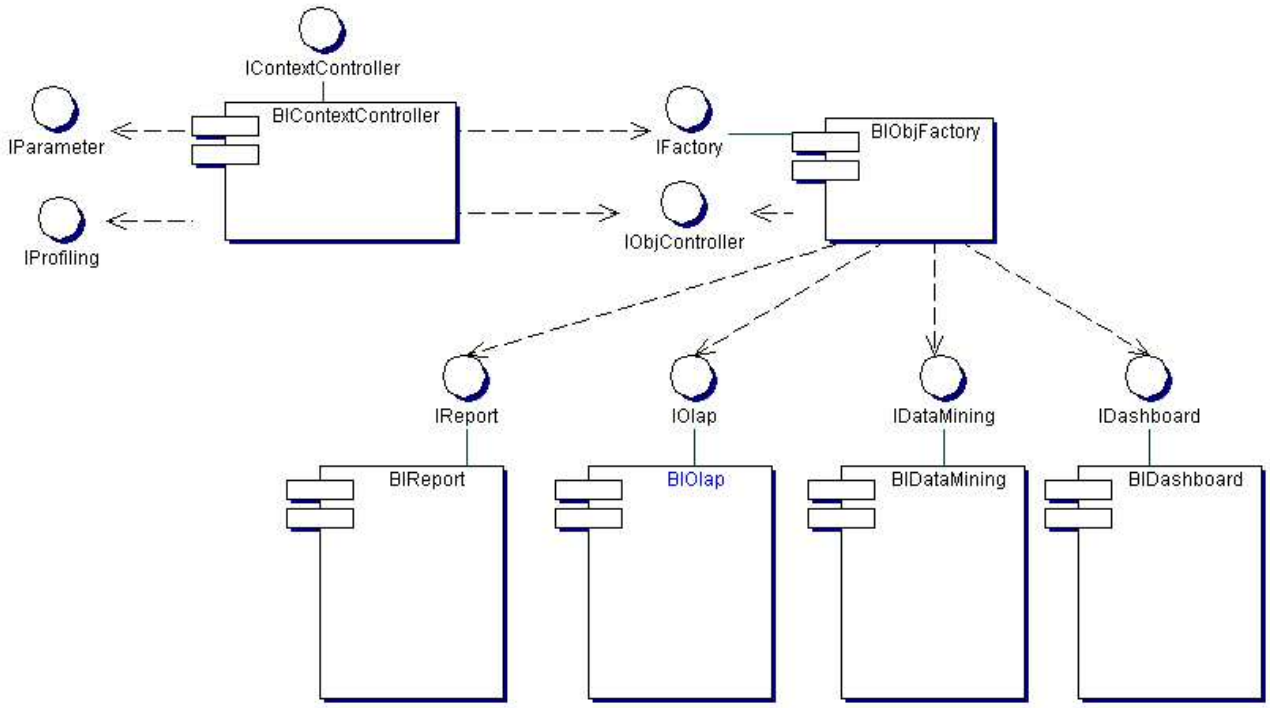
Picture 7 – BIContextController interaction

The *BIContextController* component enables:

- the interaction management between Service Components and *BIObj*s;
- the interaction management between Administrative Components and *BIObj*s;
- the navigation management through the different *BIObj*s;
- the caching of frequently used data;
- the management of a pool of *BIObj* instances, in order to improve the performances and the memory allocation strategies;
- keeping a limit to the *BIObj*s instances number, in order to regulate the workload;
- managing the objects instances and their time life according to their scope and level;
- interacting with the session at the right level;
- the factory implementation for the *BIObj*s instance;
- the parameters' inheritance in documents navigation;
- the management of a policy for the *BIObj*s releases.

The *BIContextController* depends on the *BIParameter*, *BIProfiling*, *BIObjController* components for their relative interface.

For the *BIObj*s factory implementation, the component schema follows.



Picture 8 – BIObj factory component schema

2.2.4 BIOBJFACTORY

A *BIObj* characterizes itself according to the document type it represents (report, OLAP, data mining model, dashboard and scorecard widget) and to the execution way it supports (delegated, supervised or proprietary mode).

Moreover, each different *BIObj* is implemented as a statefull and/or stateless component, in order to guarantee the user's functionalities as much as the platform performances.

Then, when the *BIObjFactory* has to allocate a *BIObj* which works in a stateless mode, it works like this:

- it takes a generic instance of the right *BIObj* from a pool;
- it sets a specific data retrieve from the *BIContextController;*
- it returns the *BIObjController* 's handle to the *BIContextController*.

On the other way, when the *BIObjFactory* has to allocate a *BIObj* which works in a statefull mode, it works like this:

- it instances the right *BIObj* type (report, OLAP, data mining model, dashboard and scorecard widget)
- it instances the *BIObj* type according to its running mode (delegated, supervised or proprietary)
- it returns the *BIObjController* 's handle to the *BIContextController*.

In general, a stateless *BIObj* needs to interact with the *BIContextController* a lot, in order to register all its state changes, and to success in recovering them at a later time.

On the contrary, a statefull *BIObj* keeps its settings and the *BIContextController* is only be able to recover its right handle.

2.3 Data and Metadata Layer

The **Data and Metadata Layer** locates itself at the data warehouse level. The data warehouse design is out of the SpagoBI scope because a DWH necessary refers to a specific customer's world; so SpagoBI refers only to the metadata level.

SpagoBI provides a meta-description for its technical aspects, for the business meanings of the data and for the processing information.

The metadata exchange is implemented according to the CWMI standard and with an hub for the external metadata collected by a bridge.

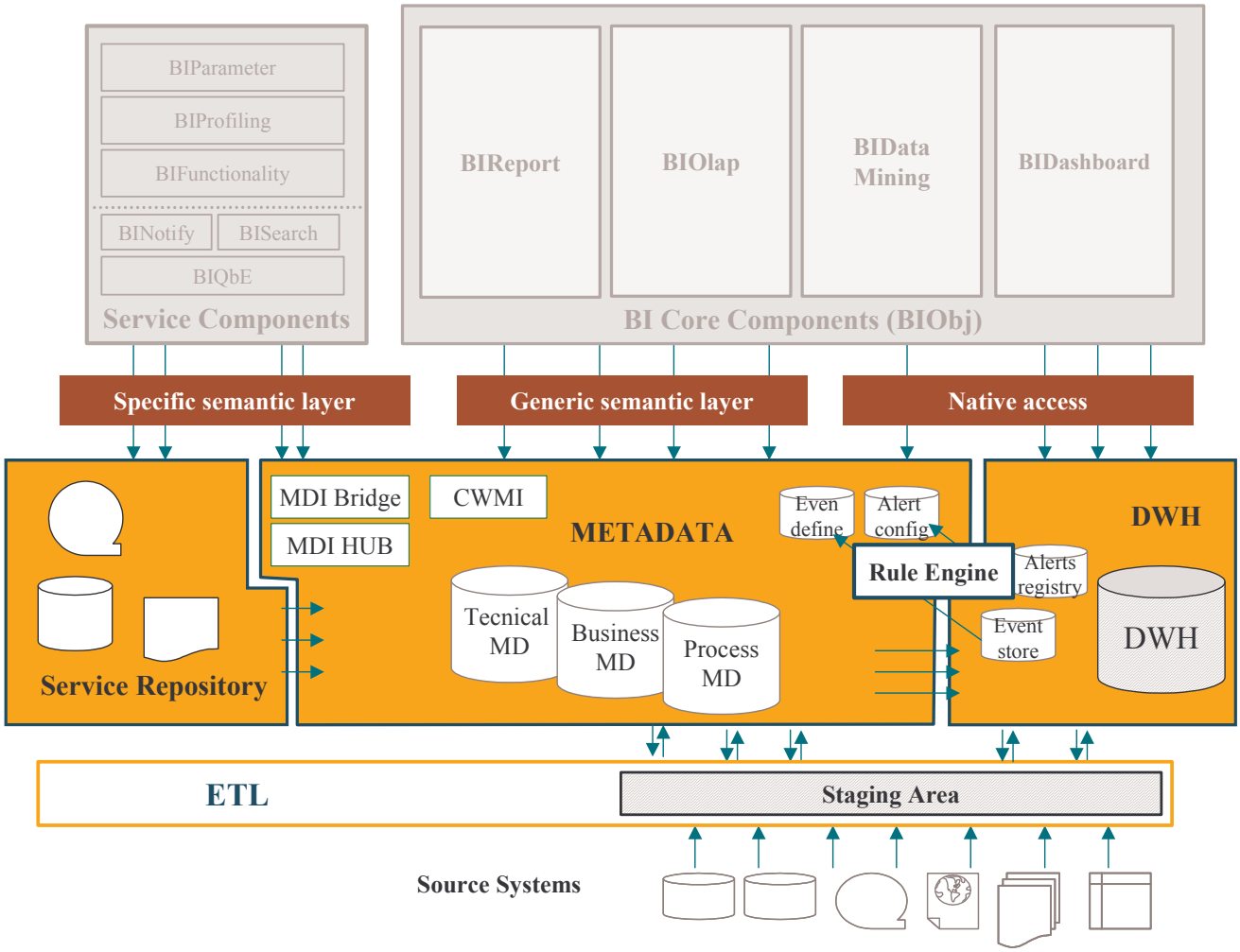
For the event-driven strategy, also not having specific data in the DWH, SpagoBI provides a formal description of event to be monitored and alerts to notify.

Some service repositories support specific tools or platform aspects such as the document management, the private repositories and so on.

The interaction with the core business components works in a native way or through a semantic interface layer, which can be generic (for example with Hibernate interface) or specific, because many tools already have their own semantic layer.

The relation with the source systems is organized by an ETL module, that provides some features for data extraction, transformation and loading. A Data flows acceptance area, where many formal checks are applied, is implemented too.

You can see in the following picture how these aspects get one's act together:



Picture 9 – Data & Metadata Layer

2.3.1 SERVICE REPOSITORY

The service repository contains private data structures which support some particular tools usage. The first one is the CMS repository, until it will migrate as an integrated metadata structure.

2.3.2 METADATA REPOSITORY

Metadata are data about data; so, the metadata repository contains information related to:

- the technical characteristics of the data stored in the data warehouse, like their formats' description, the formal validations' rules, the mapping with the source systems and so on;
- the data business meanings, in the term of semantic validation rules and description, usage preconditions, derivation and aggregation methods and so on;

- the data process describing all the actions carried out on the information, starting from the rough data value up to its semantic meaning building.

2.3.2.1 CWMI, MDI Hub and MDI Bridge

SpagoBI adopts the CWMI standard (Common Warehouse Metadata Interchange) for the metadata exchange with others platforms and products.

The Metadata Integration Hub (MDI Hub) is the first collection point of the metadata coming from other platforms and products that are able to interact via the CWMI standard.

A bridge (the MDI Bridge) allows SpagoBI to communicate with other CWM-compliant systems.

In progress.

2.3.3 DATA WAREHOUSE

The data warehouse design is out of the SpagoBI scope; however some structures can be realized in order to guide the event-driven behaviour of the platform. In this case the metadata interaction gains a strong importance.

2.3.3.1 Events Define and Alerts Configuration

At the metadata level you can provide a formal definition about the events to be monitored and about the alerts' strategy to notify them.

In progress.

2.3.3.2 Events Store and Alerts Registry

At the data warehouse level you can find the implementation of the structures on which both the real events are stored and the notification requests are written, according to their formal description's set up on the metadata level.

Setting up an operating strategy without a specific data warehouse is possible.

In progress.

2.3.3.3 Rule engine

When a specific data warehouse is available, the rule engine is able to fill Event Store and Alerts Registry tables according to the metadata level set-up rules (Event Define and Alerts Configuration tables).

In progress.

2.3.4 SPECIFIC SEMANTIC LAYER

The Specific Semantic Layer includes every private way of access to the data, that are often implemented in the specific products.

In progress.

2.3.5 GENERIC SEMANTIC LAYER

The Generic Semantic Layer makes the data layer independent from the data representation, in a standardized way.

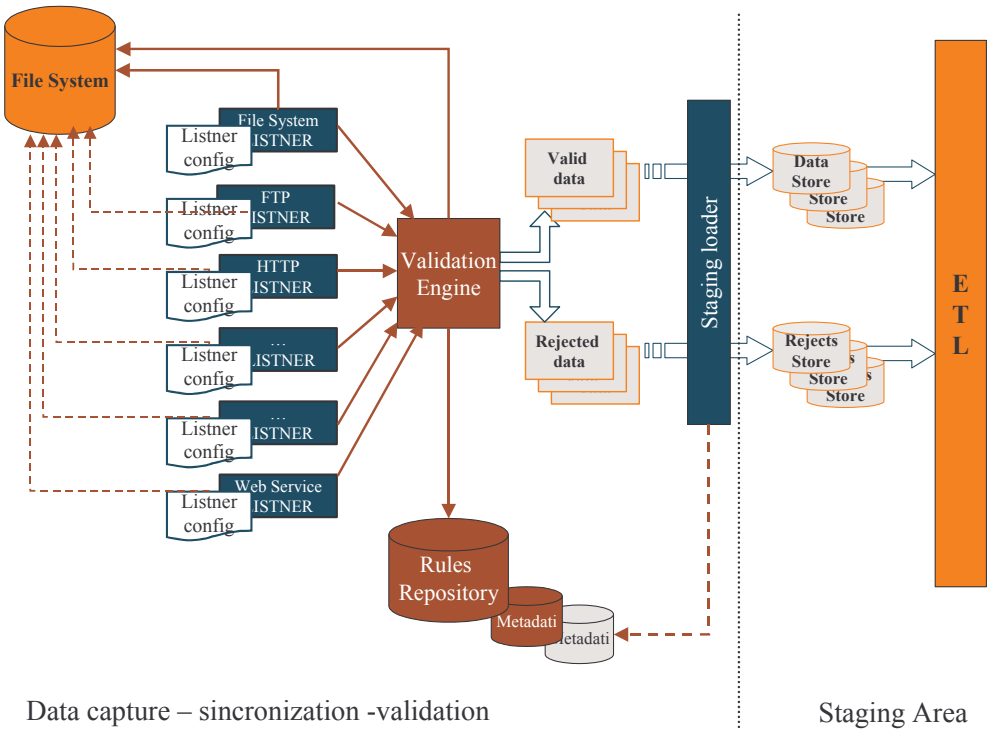
In progress.

2.3.6 ETL – STAGING AREA

An ETL module organizes the relation with the source systems; it provides some features for data extraction, transformation and loading.

In progress.

The first level is an acceptance area for the data flows, structured as follows:



Picture 10 – Acceptance data flows area

Many listeners check the possible channels and when a flow comes on, the following operations are carried out:

- the formal controls are executed on the whole flow, in terms of a set up on the metadata;
- the flow is synchronized with the expetected ones;
- the formal controls are executed on all the data in the flow, in terms of a set up on the metadata;
- the right data are separated from the wrong ones, producing the discards flow;
- the right and wrong data are provided in input to the ETL process, that sets up the management strategies for both of them.

2.4 Administration

Administration users are supported by many tools for the whole platform configuration and control, which are described below.

2.4.1 Scheduler

The *scheduler* is the administration component for the back-ground activation of the relevant processes. It interacts with the *BIObjs* and the *BIParameters* through the *BIContextController*, enabling the deferred production of the documents. In general, it allows the planning of any platform management process.

In progress.

2.4.2 Logging / Auditing

In progress.

2.4.3 Workflow

In progress.

2.4.4 Metadata Management Interface

In progress.

3 Executive level architecture

This chapter outline the platform integration strategy and a first selection of the tools used by SpagoBI is provided.

3.1 Integration strategy

SpagoBI can integrate different FOSS tools using them for the services' realization and execution.

The relation modes will support:

- Simple use. SpagoBI uses a CMS solution according to the JSR 170 specification; it allows any substitution with another JSR 170 solution with no modification.
- Integration. SpagoBI integrates the reports developed for JasperReport engine, but it keeps its platform core clearly separate from the objects which realizes the integration (i.e. *BIDriver*). In this way, for every additional engine there is a specific driver which is developed and managed as a SpagoBI plug-in.

The same methods can be extended integrating proprietary products too (i.e. CrystalReport).

3.2 First tools selection

The following is a list of Free Open Source Software solution used by SpagoBI.

3.2.1 Application Framework: Spago

Spago is a J2EE framework that enables the development of multichannel applications and integration services. It allows the development of web applications, the integration of existing infrastructures and the publishing of the services on different channels.

For further information, refer to :

- http://spago.eng.it

3.2.2 Content repository: JackRabbit

JackRabbit is a content repository API based on JSR-170.

For further information, refer to :

- http://incubator.apache.org/projects/jackrabbit.html

3.2.3 SEARCH ENGINE: LUCENE

Lucene is a Java text search engine library.

For further information, refer to :

- http://lucene.apache.org/

3.2.4 REPORT ENGINE: JASPERREPORT

JasperReports is an open source Java reporting tool that allows the delivery of rich contents onto the screen, to the printer or into PDF, HTML, XLS, CSV and XML files.

For further information, refer to :

- http://jasperreports.sourceforge.net/

3.2.5 OLAP CLIENT/ENGINE: JPIVOT/MONDRIAN

JPivot is a JSP custom tag library that renders an OLAP table and chart. By means of it, users can perform typical OLAP navigations like drill down, slice and dice. It uses Mondrian and XMLA as its OLAP engines.

Mondrian is an OLAP server written in Java. It enables you to interactively analyze very large datasets stored in SQL databases without writing SQL code.

For further information, refer to :

- http://jpivot.sourceforge.net/
- http://mondrian.sourceforge.net/

3.2.6 DATA MINING ENGINE: WEKA

Weka is a collection of machine learning algorithms for data mining tasks.

For further information, refer to :

- http://www.cs.waikato.ac.nz/~ml/weka/

3.2.7 QBE SUPPORT: HIBERNATE

Hibernate is an object/relational persistence and query service for Java.

For further information, refer to :

- http://www.hibernate.org/

3.2.8 ETL SUPPORT: BIE

The *Business Integration Engine (BIE)* is designed to help organizations to exchange data created in different applications in order to streamline the processes and to improve efficiency.

For further information, refer to :

- http://www.brunswickwdi.com/bie

3.2.9 ETL SUPPORT: OCTOPUS

Octopus is an ETL tool (Extraction, Transformation, Loading) connecting to JDBC data sources and performing XML-defined transformations.

For further information, refer to :

- http://forge.objectweb.org/projects/octopus/

3.2.10 ADMINISTRATION – SCHEDULER: QUARTZ

Quartz is a job scheduler for executing jobs whose tasks are defined as standard Java components or EJBs.

For further information, refer to :

- http://www.opensymphony.com/quartz/

3.2.11 PORTAL ENVIRONMENT – EXO PLATFORM

eXo Platform is an application suite providing the more common services for building web based information portals. It implements the JSR 168 portlet API specification.

eXo Platform is not a SpagoBI's component, but it is the first portal environment that has been verified by SpagoBI Platform. Furthermore, SpagoBI implements the interface for the recovery of the users' roles stored in eXo portal settings.

More information about eXo Platform are available starting from :

- http://exoplatform.objectweb.org