POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica (Computer Engineering)

Master Thesis in Computer Engineering

Enterprise Service Bus

A business study case

Supervisor
prof. Maurizio Morisio

Candidate
Nunzio D'Agostino

Company Supervisor
Intesa Sanpaolo Group Services
dott. Ezio Gribaudo

Academic Year 2017 – 2018

Contents

List of Figures

Abstract

Today every company of any size needs more or less complex information systems in order to be able to manage all its daily activities and to develop strategies for the future. The centrality of information systems in companies has grown in recent decades in parallel with technological innovation: the same technology, in fact, has "climbed" the different levels of the Robert Anthony's pyramid[1].

The aim of this thesis is to analyze how it is possible to integrate a legacy information system that has undergone multiple stratifications and customizations over the years with a service-oriented architecture that is compatible with all features characterizing the most recent information systems.

In the first part of the work, after a brief presentation of Intesa Sanpaolo information system, we will illustrate the different orchestration paradigms and related architectures. Starting from orchestrations that currently can be defined as legacy, such as those that occur on the Mainframe platform, we will illustrate the more recent service-oriented architecture and its implementations.

In the second part of the work we will analyze a particular banking service, the withdrawal service, to show the implementation differences between an orchestration on the Mainframe compared to that realized on a service-oriented architecture.

The last part of the thesis is devoted to the comparison between the different implementations of the withdrawal service on the two architectures. Functional and non-functional requirements, KPIs and use cases will be analyzed to evaluate the effects that the transformation has had on the service.

Finally, we will illustrate how it is possible to continue to integrate innovative technologies with legacy systems that allow the creation of core services of the bank. Particular attention will be placed on microservices.

[1]The Anthony triangle (also Anthony's triangle or pyramid) is an organizational model. The triangle takes a hierarchical view of management structure, with many operational decisions at the bottom, some tactical decisions in the middle and few but important strategic decisions at the top of the triangle. The higher in the triangle an item is, the more scope it covers and less precise it becomes. As items move down they become more detailed and apply more precisely. In the past, computer science was used in the company to speed up manual operations, so it was mainly at the base of Anthony's pyramid (operation level). With the passing of the years, however, we have assisted to a climb of technological innovation towards the top of the organizational pyramid of companies, what Anthony defines as a strategic level. Just thinking the important developments in big data or machine learning technologies and how these technologies are changing the managerial approach in companies all over the world.

Chapter 1

Intesa Sanpaolo: company presentation

1.1 History

Figure 1.1. Intesa Sanpaolo Logo

Intesa Sanpaolo is the banking group which was formed in January 1, 2007 by the merger of two leading Italian banking groups, Banca Intesa and Sanpaolo IMI. The merger brought together two major Italian banks with shared values so as to increase their opportunities for growth, enhance service for retail customers, significantly support the development of businesses and make an important contribution to the country's growth. The first steps toward the merger process were taken on August 26, 2006, when the Boards of Directors of the two banks approved the guidelines for a merger plan. The plan was then agreed by the Boards on October 12 and ratified by the Extraordinary Shareholders' Meetings of the two banks on December 1 of the same year.

1.1.1 Banca Intesa

Banca Intesa was formed in 1998 from the merger of Cariplo and Banco Ambrosiano Veneto. In 1999, Banca Commerciale Italiana (BCI) joined the Intesa Group. With the subsequent merger of BCI into Banca Intesa (May 2001), the group took the name IntesaBci. In December 2002 the Shareholders' Meeting resolved the change of the company name to Banca Intesa, effective as of January 1, 2003.

Figure 1.2. Banca Intesa Logo

1.1.2 Sanpaolo IMI

Figure 1.3. Sanpaolo Imi Logo

Sanpaolo IMI was formed in 1998 from the merger of Istituto Bancario San Paolo di Torino and IMI (Istituto Mobiliare Italiano). These two banks were highly complementary: Istituto Bancario San Paolo di Torino was specialised in retail lending, IMI, a public entity founded in 1931 to support the reconstruction of the national industrial system, was a leading business and investment bank.

1.2 Company profile

Intesa Sanpaolo is among the top banking groups in the euro zone, with a market capitalisation of 38.5 billion euro. Intesa Sanpaolo is the leader in Italy in all business areas (retail, corporate, and wealth management). The Group offers its services to 11.9 million customers through a network of over 4,500 branches well distributed throughout the country with market shares no lower than 12Intesa Sanpaolo has a strategic international presence, with approximately 1,100 branches and 7.5 million customers, including subsidiaries operating in commercial banking in 12 countries in Central Eastern Europe and Middle Eastern and North African areas, and an international network of specialists in support of corporate customers across 25 countries, in particular in the Middle East and North Africa and in those areas where Italian companies are most active, such as the United States, Brazil, Russia, India and China.

1.3 ICT in Intesa Sanpaolo

Direzione Sistemi Informativi is the beating heart of Intesa Sanpaolo's ICT.

DSI is in charge of:

- Defining of ICT Group strategy and verifying its implementation
- Defining Group's technological standards
- Governing and overseeing IT and telecommunications
- Designing, implementing and managing applications and technological infrastructures
- Offering IT services through Service Management structures and controlling service levels provided
- Analyzing technological trends
- Planning procurement of IT goods and services
- Monitoring suppliers activities to guarantee compliance with Service Level Agreement

The ICT activities, which are part of the information systems management (DSI), are strongly structured and organized according to precise codified processes, a necessary choice due to the great complexity and importance of the managed services. The DSI is organized in two lines: **application development group** and **infrastructure services group**.

In particular, the **application development group** has the task of managing the entire service life cycle of applications, from realization to production delivery, until maintenance. Among DSI goals there are the definition of the technologies, configurations and standards, and the management of capacity planning according to the services provided.

The application is developed already divided into main components, such as the Web part and the back end, after which it is taken in charge by the **infrastructure services group**, which have the competence of activation and must ensure reliability, scalability and redundancy of the service.

In the last year DSI has become increasingly important in the bank as the digitalization and the innovation are becoming more and more lively.

Intesa Sanpaolo's innovation is both an instrument for the Group's Business Units to foster incremental innovation of products and services that help the bank to be increasingly competitive and close to its customers both in a medium-long perspective term, a means to develop new business models for the Bank and its customers in the various industries: from the sharing economy to the new Insurtech models, from the infomediation as a revenue resource to the development of big data, artificial intelligence and healthcare, all what could represent new revenue areas for Intesa Sanpaolo in the prospective horizon of the next 5-10 years.

Chapter 2

Definitions

From the 19th edition of KPMG's IOC Survey 2017, which involved over 4,500 CIOs and IT managers from 86 different countries, it emerged that 89% of the companies interviewed increased their IT investments to adapt to the sudden changes in the markets in which operate.[3]

It seems clear that more and more is invested in technology as this aspect is considered one of the few factors that can make it stand out on the market. Information systems have become a strategic asset for companies and the choice of a technological architecture can be a determining factor for success.

One of the most known architecture which enables to fast changes is Service Oriented Architecture implemented by an Enterprise Service Bus.

In this chapter there are some useful definitions of terms which will be recurrent in the thesis.

2.1 Enterprise Architecture

Enterprise architecture (EA) is a comprehensive operational framework that explores all of an organizations functional areas while defining how technology benefits and serves the organization's overall mission. The technological aspect of EA defines the hardware, operating systems, programming and networking solutions a business employs and how those may be used to achieve its current and future objectives.

EA includes the following components: mission, stakeholders and customers, processes, applications and infrastructure, networks, and data.

EA facilitates improvement of processes in the following ways:

- Discovering business processes that require change
- Efficiently and continuously managing change via clearly defined documentation
- Developing and implementing enterprise-wide procedures
- Encouraging effective enterprise-wide communication, which, in theory leads to better decisions.

2.2 Service

A service[2]:

- Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
- Is self-contained
- May be composed of other services
- Is a “black box” to consumers of the service

2.3 Service Oriented Architecture

Open Group definition of Service-Oriented Architecture (SOA) describes it as an architectural style that supports service-orientation.

Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services[2].

A Service Oriented Architecture supports, by integrating them, business processes as a set of services interoperable. It represents, on one side, an important solution based on standards to ensure interoperability between software programs independently from languages and operating systems, on the other, tends to fill the gap between processes and the activities carried out by the company and ICT tools that support them.

Main principles of Service Oriented Architecture are:

- Service-based: business functionality is made available in the form of ‘services’, which can be fairly granular. Each service provides a business function that is independent of other services.
- Platform-independent: SOA services can be on any platform that supports the service transport and interface requirements.
- Implementation-independent: services are designed to be independent from their own implementation, minimizing the impact of change management if the implementation changes. In a coupled system, an implementation change (e.g. a change in a back-end data type) would require a change to the interface and therefore the connecting systems.
- Single contact point: a single service registry/catalogue allows to locate each service deployed on the architecture and provides correct parameters to invoke it.

2.4 Enterprise Service Bus

"Enterprise Service Bus" term was introduced in the field of computer science at the beginning of the 2000s.

An Enterprise Service Bus (ESB) is an integrated platform that provides fundamental interaction and communication services for complex software applications via an event driven and standards-based messaging engine, or bus, built with middleware infrastructure product technologies. The ESB platform is geared toward isolating the link between a service and transport channel and is used to fulfill service-oriented architecture (SOA) requirements.

Main characteristics of an ESB are:

- Standardization of communication languages
- Integration of different systems and different applications
- Transport, transformation, routing, enhancement of messages
- Service orchestration enabling Service Oriented Architectures

Using an ESB in an Enterprise Service Architecture means that the applications that form it will communicate with each other via the bus. This has the advantage of reducing the number of point-to-point connections needed to make the applications communicate with each other. This facilitates the analysis of software changes that are simpler and clearer.

Finally, an Enterprise Service Bus implements a communication system between heterogeneous software that interacts with each other in a Service Oriented Architecture removing interdependencies between applications.

Chapter 3

Withdrawal service: orchestration on Mainframe

In this chapter will be analyzed the AS-IS architecture describing a very common bank service like withdrawal. It will be described Mainframe and the orchestration which is implemented on it by using two fundamental objects of Intesa Sanpaolo IT: Business Service and Business Component.

3.1 Description of withdrawal service

The withdrawal operation allows cash withdrawal at bank office from the customer's bank account or from savings account. The operation requires:

- Form: verbal request at the counter or filling a debit form.
- Check: own check or check made out to the customer himself. Check data capture is done by a scanner; in case the device is not installed or active, the operator must enter check identification data manually.
- "Third Party Withdrawal": for this mode it is necessary that the presenter has a letter for Disposition that authorizes him to withdraw funds from the current account.
- Simple Credit: this function is triggered "silently" by another application called CRESP; for this withdrawal method it is necessary the current account has a Credit Simple platfond on the specific account for the operating branch.

3.2 Mainframe

A Mainframe computer is a computer system specialized in high workload of centralized data elaboration. Mainframes are often used in large data centers by structured companies and governative organizations, where exist high levels of processing performances and

Figure 3.1. The IBM Z mainframe.

[?]

massive input/output volumes. In these types of organizations the use of mainframe systems are consequence of their needs of high availability and of central government of the infrastructure.

Modern mainframes can run multiple different instances of operating systems at the same time. This technique of virtual machines allows applications to run as if they were on physically distinct computers. In this role, a single mainframe can replace higher-functioning hardware services available to conventional servers.

Nowdays mainframes, notably the IBM zSeries, System z9 and System z10 servers, offer two levels of virtualization: logical partitions (LPARs, via the PR/SM facility) and virtual machines (via the z/VM operating system). Many mainframe customers run two machines: one in their primary data center, and one in their backup data center—fully active, partially active, or on standby—in case there is a catastrophe affecting the first building. Test, development, training, and production workload for applications and databases can run on a single machine, except for extremely large demands where the capacity of one machine might be limiting. Such a two-mainframe installation can support continuous business service, avoiding both planned and unplanned outages.

Mainframes are designed to handle very high volume input and output (I/O) and emphasize throughput computing. Since the late-1950s, mainframe designs have included subsidiary hardware (called channels or peripheral processors) which manage the I/O devices, leaving the CPU free to deal only with high-speed memory. It is common in mainframe shops to deal with massive databases and files. Gigabyte to terabyte-size record files are not unusual. Compared to a typical PC, mainframes commonly have hundreds to thousands of times as much data storage online, and can access it reasonably quickly. Other server families also offload I/O processing and emphasize throughput computing.

Mainframe return on investment (ROI), like any other computing platform, is dependent on its ability to scale, support mixed workloads, reduce labor costs, deliver uninterrupted service for critical business applications, and several other risk-adjusted cost factors. Companies develops on Mainframe their core services and make them available to many "consumers" who could need to use the same service at the same time: one need only think, for example, how many flights of the same airline are booked in a second or the amount of bank transactions which every minute are completed.

3.3 Orchestration elements

Mainframe technical characteristics make it a good candidate to implement a service oriented architecture. For this reason Intesa Sanpaolo, when the number of applications accessible to the end customer has rapidly increased, has invested in an infrastructure that implements the principles of SOA, still keeping the product centrality. More specifically, a group of services could be orchestrated only inside the same product silos. As first approach, the development team creates different services, called Business Components, which are orchestrated by more complex procedures written in COBOL, called Business Services.

3.3.1 Business Component

Business component can be defined as an elementary operation in which it is not necessary take care of transactionality, because it is assigned to business service logic. It allows to execute operations interfacing with mainframe resources (files, database connections, etc.) through a CICS (Customer Information Control System). If we imagine a more familiar C program, we can associate a business component with a C function called by a "parent" main.

3.3.2 Business Service

Continuing the analogy with C program, business service can be defined as the main() function which cares about transactionality of entire operation during business components orchestration. A problem in a single business component brings to rollback the entire business service. The "Business Service" application architecture in Intesa Sanpaolo can be consider a particular declination of the Service Oriented architecture (SOA) because it implements the orchestration of different legacy applications on Mainframe. However, we can not yet define the business service as real service because the business components are not yet atomic and contain code customization according to calling channel. Business service uses two distinct custom software, the ISP connector and ISP Middleware, developed since 2000 which allow respectively to invoke a business service and to track its activity. In this architecture the "consumer" component of the service it is a distributed application, while the "provider" component is a Mainframe application.

3.3.3 ISP Connector

Requests from the consumer are brokered through the "Connector". The Connector transforms the native data received from the application into a structured message and deals with addressing it through the network of channels to the Mainframe environment, which must provide the requested service. It receives the answer and returns it to the calling application in a readable format.

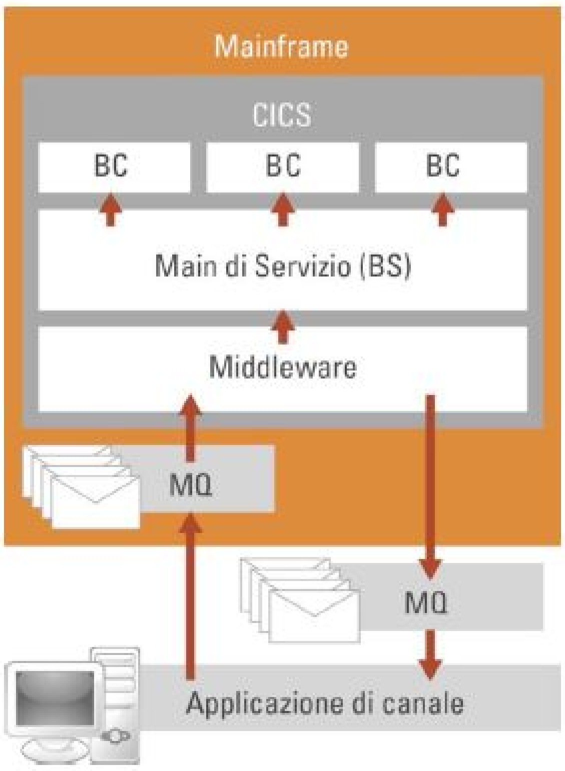


Figure 3.2. Service request and service response through ISP Middleware and ISP Connector.

3.3.4 ISP Middleware

The ISP Middleware is a Mainframe software infrastructure in charge of receiving all call requests directed to Business Service, interfacing applications that expose the service (the providers). The middleware tracks incoming and outgoing messages on a specific message infrastructure; verifies the user's authorization to perform the service; and finally manages the consistency of updates made by the different components of the Business Service (Business Component). The main model of communication with Business Services is "Request - Reply" approach by MQ messages.

3.3.5 IBM MQ

The huge amount of "request - reply" traffic among the different Business Services and Business Components is managed by a product, called IBM MQ[1].

IBM MQ allows to enable a correct dialogue between different applications through a mechanism similar to that of a common "messaging app", like Whatsapp or Messenger.

Sending a message from a point A to the point B can be performed by contacting a server which route the message on the correct destination. The message ends up in a sort of mailing box (the queue) and is available to be read as soon as the receiver is ready.

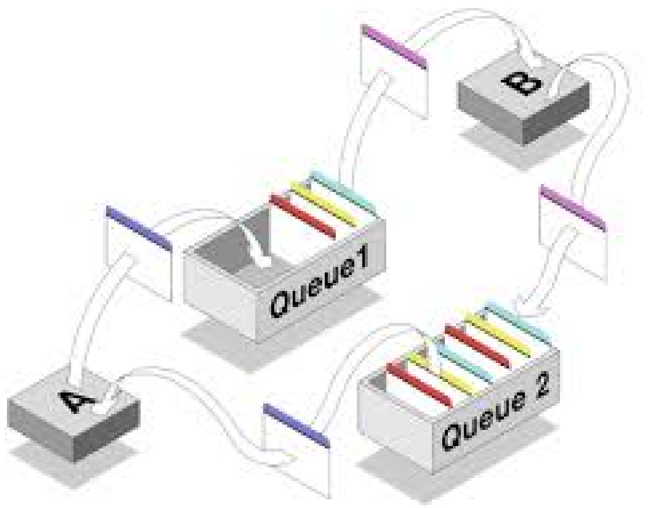


Figure 3.3. Description schema of basic MQ mechanism.

In other words if an application A needs to send a message to an application B, it has to communicate with a **Queue Manager** which is in charge of guarantee:

- delivery of the message to the correct recipient;
- message is not lost;
- message is not altered;

and provide a number of ancillary functions, such as receipt management return, distribution lists, publish and subscribe, etc.

Each application uses the services of one or more queue managers, that, in order to deliver messages to all recipients, must communicate with each other through a network of "connections",which are called "channels".

IBM MQ, as described above, is a product particularly suitable to an asynchronous communication but in Intesa Sanpaolo, exploiting its synergy with Mainframe environment

[1]MQ stays for Messaging and Queuing

and led by the necessity to do not lose messages, it has become a fundamental part of the "synchronous" dialogue among applications.

In addition to the request-Reply communication pattern (which in the case of transport via MQ is not a true synchronous dialogue), two other real asynchronous patterns are used: Fire & Forget and Notifications.

3.4 Logical architecture of withdrawal service on Mainframe

At the logical level, the withdrawal service could be divided in two main phases which involve several components:

- "Sportello" application
- Customer Identity Business Service
- Withdrawal Business Service
- Check Service on Mainframe (only if user wants to use check for his withdrawal)
- Accounts Service on Mainframe
- Saving Accounts Service on Mainframe

At the beginning of every withdrawal operation a user requires to a cashier to make the operation by a form or by a check. The cashier logs in "Sportello" application and starts to insert all data required by application.

Orchestration of withdrawal service on mainframe is managed at two different layers. Top layer orchestration is at front-end, by application "Sportello". "Sportello" application, in fact, after receiving all data entered by cashier orchestrate two different Business Services: Customer Identity Business Service and Withdrawal one.

In a first phase, through Customer Identity Business Service invocation, a first part of several checks takes place on account holder identity, account balance, privacy preferences, joint signature, IBAN existence, paperless preference, etc. In addition to these checks, it is mandatory, according to Italian financial laws, to verify whether the customer has filled the GIANOS survey: an unfilled survey or a lack of this important document forces exit by a terminal error. [2]

After GIANOS check and according to the request type (form or check), it is done a **simulation** of the withdrawal, calling the Withdrawal Business Service in simulation mode. The simulation has in charge of obtaining other information about customer, his account and to verify if the operation could be successfully executed or not. In this phase another part of several checks takes place thanks to the Business Service which has in charge of

[2] **GIANOS** stays for "Generatore di Indici di Anomalia per Operazioni Sospette" (in English could be traslated with "Anomaly Index Generator for Suspicious Transactions") is an IT procedure for selecting potentially suspicious transactions spread among Italian banks. It was carried out by an inter-bank working group coordinated by ABI with the support of legal, IT, organizational and statistical experts.

orchestrating several business components, including those belonging to Accounts Service, Check Service and Saving Accounts Service.

These Services correspond to different areas of Bank's Mainframe infrastructure, they can be considered "Service Silos" because each service is separated from other ones and contains all resources which needs (Databases, Memory Areas, etc.).

As can be seen in the flow chart of figure 3.4, whole logical core of the withdrawal service is written in main business service which has in charge of guaranteeing the transactionality and of interfacing with all "smaller" business components. In other words, behind Withdrawal Business Service there is a huge COBOL coded procedure which orchestrate all necessary Business Component in a transactional way.

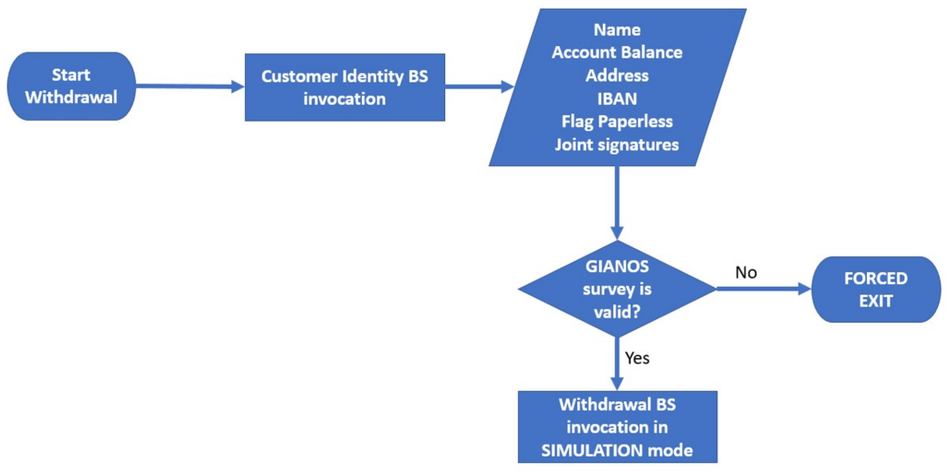


Figure 3.4. Flow chart of first "simulation" part of withdrawal operation

At the end of the simulation phase, if it is positive, takes place the second one, the **dispositive phase**. In this phase "Sportello" application receives simultation outcome and obtains also data about charges according to simulated operation.

Finally, it invokes the same Withdrawal Business Service by but in dispositive mode this time.

All checks and orchestrations are re-executed because some information retrieved in the simulation phase could be no longer valid (i.e. account balance) and then Business Service returns a "Withdrawal Result Tag".

This tag contains information about:

- Account Balance after operation
- Possible alerts
- Data for accounting printing

Last step consists in signature verification: it is necessary to check if the account has a joint signatures or not and to save the signature of withdrawal receipt.

In the past, when in branches there were not yet digital tablets, this step required an other Business Service invocation but now, with a complete digitalization of branches no invocations are necessary because the signature is registered by tablet and the check of joint signatures can be performed by first Customer Identity Business Service.

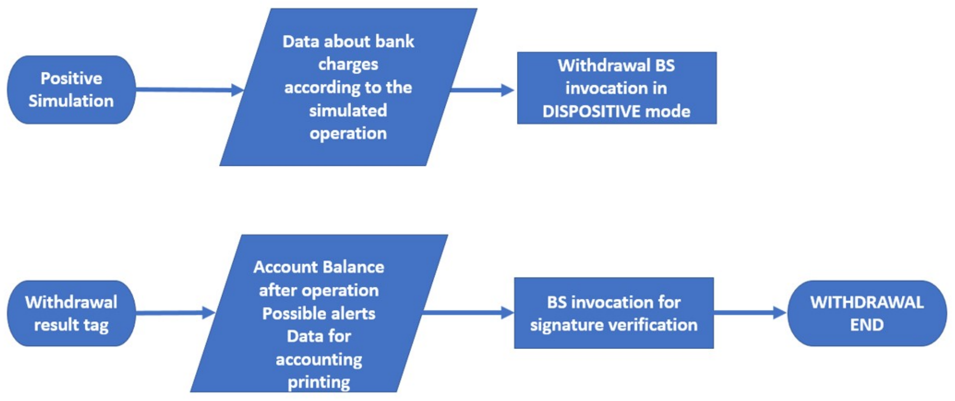

Positive Simulation
Data about bank charges according to the simulated operation
Withdrawal BS invocation in DISPOSITIVE mode
Withdrawal result tag
Account Balance after operation Possible alerts Data for accounting printing
BS invocation for signature verification
WITHDRAWAL END

Figure 3.5. Flow chart of "dispostive" part of withdrawal operation

Chapter 4

Withdrawal service: orchestration on Service Oriented Architecture

During analysis of the withdrawal service orchestrated on Mainframe, it was possible to see how this service was declined on a legacy infrastructure and how a service-oriented architecture was only partially implemented inside business services. It can be said that it is a system architecture based on vertical legacy silos that include data, business logic and (sometimes) front-end, determining:

- High effort to implement new business functionalities
- Data fragmentation, with higher complexity to have a 360° view of the customer
- Access channels with limited homogeneity

Within the bank it was necessary to rethink all the most used applications in a new perspective that would allow the reuse of the components used, a greater speed in the development of new services and a centralized view of the customer information. Also the Withdrawal Service it was re-engineered on the new customer centric architecture, which implements all principles of a Service Oriented Architecture.

4.1 Customer Centric Architecture: Service Oriented Architecture in Intesa Sanpaolo

In an overall market context in which technology evolves at a speed never observed before and where consumer habits are in continuous and rapid transformation, to meet the new challenges it is necessary to create a digital strategy integrated with the business strategy.

Led by these reasons, Intesa Sanpaolo has created a platform enabling the objectives of the Business Plan, creating an architectural model oriented to fast and full changes, designed on the customer. The value from which the project starts is **Customer Centricity**. That is the recognition of the strategic nature that the individual person, current or potential customer, covers for the bank in terms of quality of the provided and delivered services (e.g. best next product, real time marketing, time to market). In this way, Intesa Sanpaolo has created a platform which embodies a Service Oriented Architecture.

4.1.1 Principles

The architecture is based on four principles:

- **Agility**: adoption of agile solutions able to allow the integration of new custom components or products, simply, quickly and governed. The solution ensures continuity both in the innovation process and in the guarantee of regulatory adjustments, in a simple and effective way.
- **Always on**: continuous availability to offer also extended products today supplied only during work hours. Use of products and services in real time mode on the various customer contact channels.
- **Services Hub**: creation of a "service of services" ready to provide functionality to the group and outside the group. The range of services offered extends to banks and foreign branches.
- **Customer Centricity**: complete view of the customer both retail and corporate, in a centralized, univocal and updated manner.

4.1.2 Layers

The structure of this architecture is on three levels called layers:

- **"Presentation Layer"** is in charge of presenting applications on the various "channels" (Branch office, contact unit, Internet banking, Mobile, Video, ATM, SMS, ...) adapting to the peculiarities of both devices and users. Channels represent the access layer to the Bank's services, where both customers and employees access in traditional or innovative ways (mobile, social ...).
- **"Agility Layer"** contains the central business logic (visibility data, process engine, orchestration services, centralized management commercial products), for the realization of business services and processes. Inside the Agility Layer the Enterprise Service Bus, implemented by IBM Integration Bus, is fundamental; it allows in fact the design, execution, and monitoring of end-to-end processes, and permits flexibility in creating new products and in reviewing existing ones. Processes are implemented in the agility layer of long-term business, provided with status. This "key" layer includes also a "Rule engine" which allows to enable changes in real time and a product for managing long-term business processes (i.e. a procedure to take out a mortage).
- **"Core Banking Layer"** is an important element in the new architecture and represents the revision of the government systems of the Bank in order to ensure consistency and timeliness in providing information for vertical supply chains of the individual stakeholders. It is the engine of the individual elementary application services. It is the "lockbox of data" of the bank (Register, Accounts, Cards ...). This layer is

mainly formed by old Business Services and Business Components, but they have been transformed and adapted to interface with new orchestrated services.

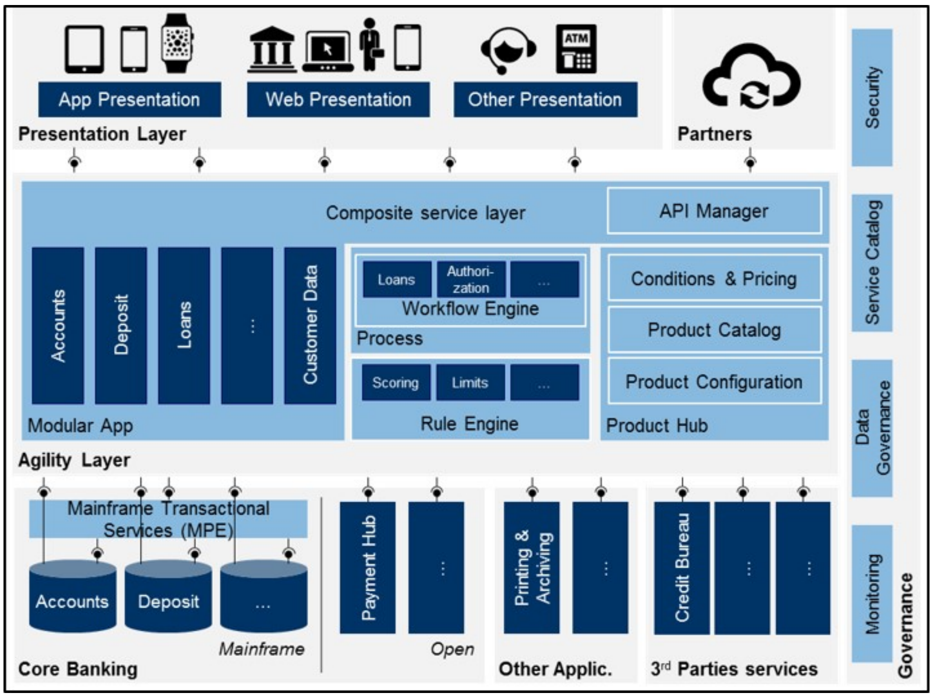


Figure 4.1. Logical model of Customer Centric Architecture

4.1.3 IBM Integration Bus

IBM Integration Bus represents the beating heart of Agility Layer. It is an Enterprise Service Bus supplying a communication channel between applications and services in a service-oriented architecture. IBM's Integration Bus is also known with its old name as Websphere Message Broker and it belongs to WebSphere product family. It allows business information to flow between disparate applications across multiple hardware and software platforms. In other words, IBM Integration Bus allows to connect applications together, regardless of the message formats or protocols that they support.

Main characteristics

IBM Integration Bus routes, transforms, and enriches messages from one location to any other location:

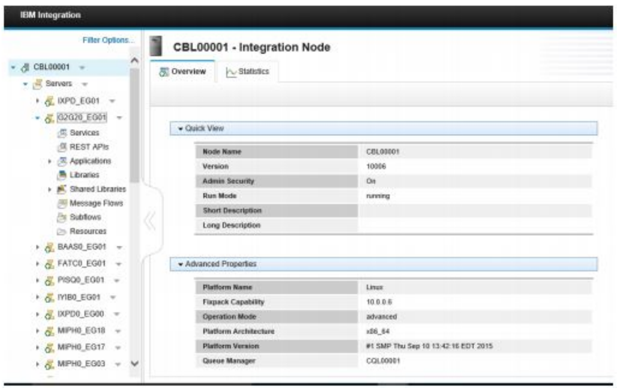
- The product supports a wide range of protocols: WebSphere MQ, JMS 1.1 and 2.0, HTTP and HTTPS, web services (SOAP and REST), File, Enterprise Information Systems (including SAP and Siebel), and TCP/IP.

- It supports a broad range of data formats: binary formats (C and COBOL), XML, and industry standards (including SWIFT, EDI, and HIPAA). You can also define your own data formats.
- It supports many operations, including routing, transforming, filtering, enriching, monitoring, distribution, collection, correlation, and detection.

IIB Tools

IBM Integration Bus consists of the following components:

- IBM Integration Toolkit is an Eclipse-based tool that developers use to construct message flows and transformation artifacts using editors to work with specific types of resources. Context-sensitive help is available to developers throughout the Toolkit and various wizards provide quick-start capability on certain tasks. Application developers work in separate instances of the Toolkit to develop resources associated with message flows. The Toolkit connects to one or more integration nodes (formerly known as brokers) to which the message flows are deployed.



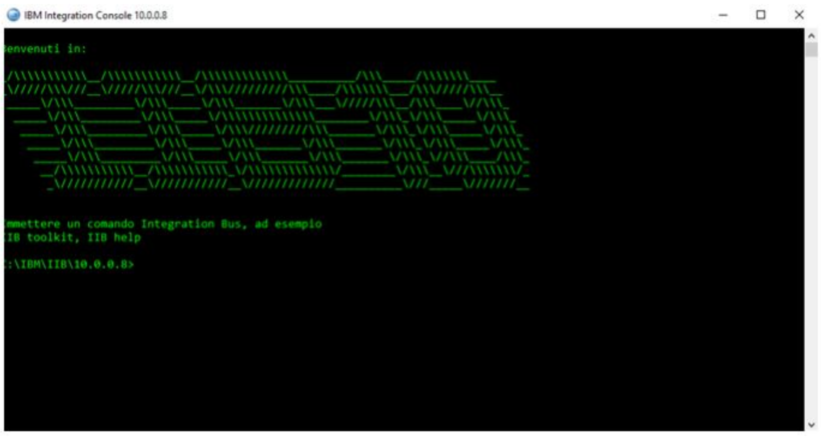
CBL00001 - Integration Node

Node Name	CBL00001
Version	10006
Admin Security	On
Run Mode	running
Short Description	
Long Description	

Platform Name	Linux
Fixpack Capability	10.0.0.6
Operation Mode	advanced
Platform Architecture	x86_64
Platform Version	#1 SMP Thu Sep 10 13:42:16 EDT 2015
Queue Manager	CQL00001

Figure 4.2. IIB Web User Interface screenshot.

- IBM Integration Bus web user interface (UI) enables System Administrators to view and manage integration node resources through an HTTP client without any additional management software. It connects to a single port on an integration node, provides a view of all deployed integration solutions, and gives System Administrators access to important operational features such as the built-in data record and replay tool, and statistics and accounting data for deployed message flows. (The web UI supersedes the Eclipse-based Explorer from earlier versions).
- IBM Integration Console is a more powerful user interface for System Administrators who can interact with the Integration Bus by command line. This console consists in a IBM customized shell different for each operating system on which the Integration Bus is installed on.



```
IBM Integration Console 10.0.0.8

envenuti in:

mmettere un comando Integration Bus, ad esempio
IB toolkit, IIB help

:\IBM\IIB\10.0.0.8>
```

Figure 4.3. IIB Console screenshot for admistration purpose.

Main components

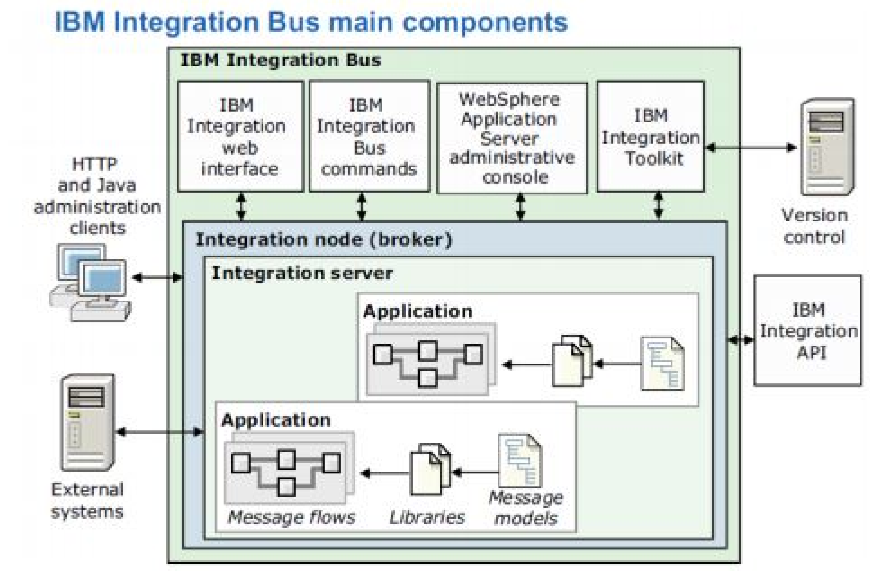

IBM Integration Bus main components
IBM Integration Bus
IBM Integration web interface
IBM Integration Bus commands
WebSphere Application Server administrative console
IBM Integration Toolkit
Version control
HTTP and Java administration clients
Integration node (broker)
Integration server
Application
Application
IBM Integration API
External systems
Message flows
Libraries
Message models

Figure 4.4. IIB Console screenshot for admistration purpose.

Working on IBM Integration Bus requires knowing some key elements:

- **Message Flow**

  Message flow describes the application from a logical point of view and defines the path that data must follow within the integration node. The message flow requires a message model that defines the structure and properties of data passing through.

- **Integration Server** Each Integration Server is a different process in the IIB machine that handles services by thread. The integration node allows to isolate message flows in distinct integration servers by ensuring that they run in separate address spaces, or as unique processes.

Each integration server is started as a separate operating system process, providing an isolated runtime environment for a set of deployed message flows. Within an integration server, the assigned message flows run in different thread pools. You can specify the size of the thread pool (that is, the number of threads) that are assigned for each message flow by specifying the number of additional instances of each message flow.

- **Integration Node**

  An integration node is a set of execution processes that hosts one or more message flows to route, transform, and enrich in flight messages. It checks integration servers are running according to the operations specified by their respective messages flows.

  Several integration nodes can operate on different operating systems so that it is possible provide protection against failure and separate business flows on different architectures. For example, you might have one integration node that handles all your financial applications, and another that handles other business processes.

  Application programs connect to and send messages to the integration node, and receive messages from the integration node. The integration node routes each message by using the rules that you have defined in message flows and message model schema files, and transforms the data into the structure required by the receiving application.

4.1.4 Process Engine

The **Process Engine** is just the component in which long-term business processes are designed and executed.

The Process Engine is therefore the owner of the process and invokes related services and the front-end applications, to allow end-to-end processes implementation and management, using rules to add flexibility.

The Business Process Manager (BPM) is a workflow engine made available by the Agility Layer in Intesa Sanpaolo, which allows definition and execution of long-lasting business processes (DLD or long running).

A "DLD stream" implements a business process, that is a state-provided functionality that typically includes human interactions. The DLD flow consists of several tasks, even if separated over time and guided by external events, each of which can aggregate the execution of multiple DLD services or sub-flows according to the process's own logic. The DLD flow is exposed by this component of the Agility Layer and has the following characteristics:

- **Status**: the concept of status is directly referred to the so-called "state machines". In a context of "workflow management" it is however possible to define a logical and "behavioral" state machine, but in general in the BPM it is preferred to say that a "state" is always linked to one or more "business entities", entities whose status is defined by specific enhancements of the expected attributes (properties). Usually the status of one or more business entities is changed in the transition from one "activity" to another in the process.

A DLD stream is provided with status indicating performed tasks, those waiting to be executed and those in progress. A set of business attributes (for example order status) that characterize the DLD flow in every single instant can be registered. Status and attribute management is delegated entirely to the Process Engine; • **Access to databases**: a DLD stream has no direct access to any application database but only to the state database for state management; • **Census**: a DLD flow must be recorded in the Service Catalog; • **Call for sub-flow**: a DLD stream can call DLD sub-flows as part of its execution; • **Life cycle, orchestration and Rule Engine**: a DLD flow comes from a startup task. It accepts and processes external events, coming both from human interactions and from signals originating from other processes or systems.

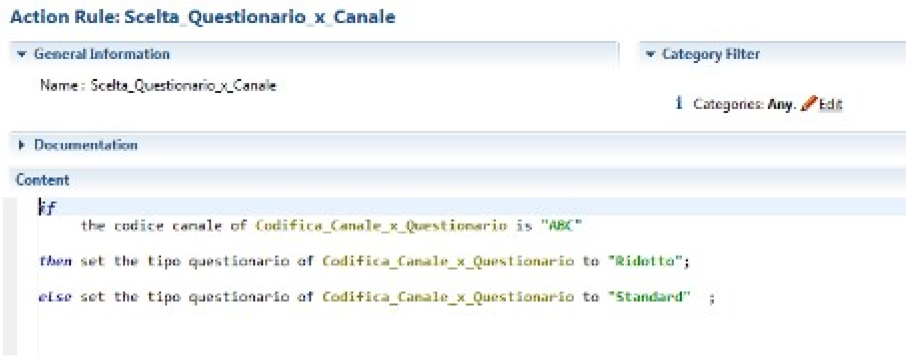
Each event is processed according to the defined orchestration logic, which may include logic of services or sub-processes, also conditional, and variations of the DLD flow attributes. As in the case of composite services, the decision rules that drive the business orchestration must be delegated to the Rule Engine component.

The DLD flow must be limited to the orchestration of the business logic realized through the services but it must not contain within it the implementation logic of the service itself. This increases the modularity of the system, the transparency and the reuse of components.

4.1.5 Rule Engine

In addition to BPM, the process management involves also the **Rule Engine**: engine dedicated to the definition and execution of the business rules, in order to guarantee their modularity.

This component allows: outsource the rules from processes and services and centralize execution and governance of themselves. In this way, for example, a new law on cash withdrawal limit doesn't requires changing in the code of service but only an update on the correct record in the Rule Engine.



Name : Scelta_Questionario_x_Canale

```
if
     the codice canale of Codifica_Canale_x_Questionario is "ABC"
then set the tipo questionario of Codifica_Canale_x_Questionario to "Ridotto";
else set the tipo questionario of Codifica_Canale_x_Questionario to "Standard"  ;
```

Figure 4.5. Example of a rule in ODM.

The use of the Rule Engine implies that the business rules of the service must be registered and implemented on the rules engine and made decoupled by the code; this guides services towards a logic of modularity and composition. IBM ODM (Operational Decision Manager) is the tool used to perform the Rule Engine functions. Main benefits of using ODM are:

- **Identification**: the transposition of an algorithm made up of if-then-else constructs in order to give it a precise functional connotation and to identify it with a name and a description;
- **Isolation**: the use of rules makes it possible to aggregate and isolate business logic from other application functionalities (for example, to distinguish rules from orchestrations). The isolation of the rules allows also the execution of unit-tests in a context independent from that of the application. The rules are collected and classified in the ODM catalog;
- **Reuse**: the cataloging of the rules makes it easier to reuse by different services;
- **Readability**: the ability to express the rules in a language and a business dictionary makes them understandable even to non-technical figures. Readability also translates into greater ease of maintenance and modification by developers;

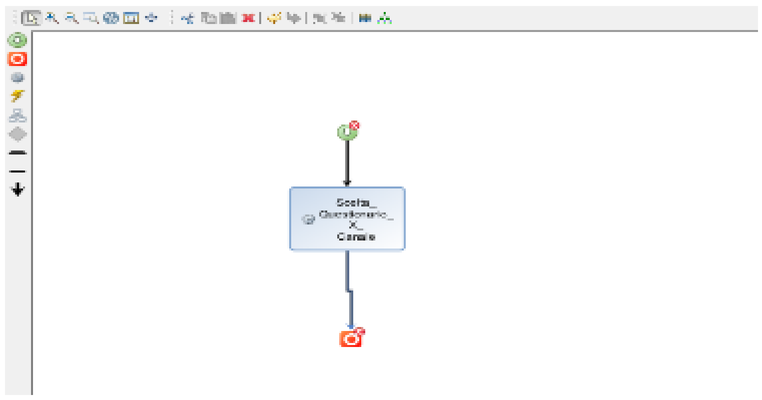


Figure 4.6. ODM rule can be developed also in a graphical way, which allows to concatenate more simple rules in order to create a new more complex rule.

The Rule Engine in the context of the Agility Layer is invoked by the orchestrator (IIB) and the Workflow Engine (BPM), which use it to elaborate business rules both at the services and the processes level. The ODM offers developers, in a native way, the possibility of implementing a simple system of business rules based on a business language (Business Activity Language) or on a series of decision tables (Decision Table) that allow to specify the criteria of evaluation of a certain policy or business rule.

4.1.6 Service Catalog

The service government represents a challenge for the control of a Service Oriented Architecture. It is necessary a focus point on which all architectural components find information about services that must be orchestrated. In Intesa Sanpaolo architecture exits a Services Catalog, currently implemented with the WSRR product (Websphere Service Registry and Repository).

The Service Catalog is used:

- in Run phase to address calls;
- in Development phase as a census and a government tool to facilitate the re-usability of services.

In other words, when an IIB flow needs to invoke a service, it can use an endpoint lookup node to dynamically retrieve the endpoint of that specific service. At run time the IIB flow can retrieve the correct endpoint without any configuration inside the code and even if a service endpoint changes, it is necessary only update it in WSRR, and the new endpoint is automatically retrieved by the IIB flow.

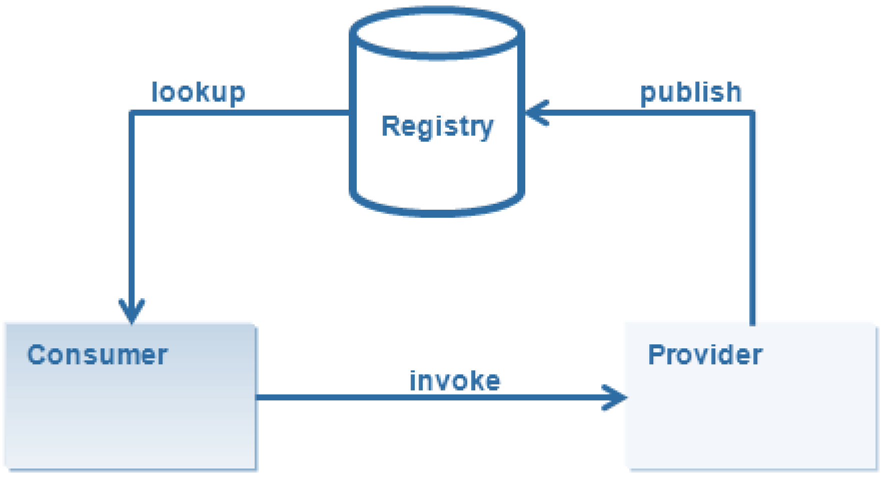


Figure 4.7. Lifecycle of and endpoint in Service Catalog.

Types of services

The catalog is the only repository of services. The following categories are available:

- **Host Services**: include the following sub-categories: Business Component, Host Atomic Services, Generalized Routines;
- **Host Composite Services**:include the following sub-categories: Business Service, MPE Services;
- **Open Services**:include the following subcategories: Open Atomic Services , Open Generic Services ('SGOPEN'), REG ('ODM Rule'), WS IIB Services for Parallelism;
- **Open Composite Services**: include the following sub-categories: IIB Modular Services.

4.1.7 Customer Data

The application solutions before Customer Centric Architecture are mainly based on the concept of product. Each legacy silos manages within it "valuable" data relating to the relationship that customer has with the bank. Making available for an access channel, this information in 'real time' mode, accessed using a single point of contact, constitutes an important value and a factor of speed in the construction of new initiatives.

Moving from product-centric to customer-centric architecture has represented a challenge that the construction of the Customer Data component has accepted with the objective of offering an aggregate and complete representation of the Bank's information assets regarding customer, retail, corporate for channels that request it.

Customer Data offers a complete view of the centralized and univocal customer, to be used as a source of interrogation for information or for commercial proposals (real time marketing).

The main purpose is to free the application teams from the need to have to census and analyze all the services of legacy data silos that provide customer information.

The responsibility of the data is the exclusive responsibility of the relative data silos (Customer Identity, Accounts, etc.), not of the Customer Data. The Customer Data can only be queried read-only, any data modification operations must be carried out directly through the data silos.

The Customer Data service can be queried in two ways:

- Query from the Presentation Layer via the Connection Manager
- Query by orchestration IIB

The construction strategy of the component has foreseen the realization a layer of «virtualization» of information through one appropriately developed modular application.

Designing services to access information more efficiently, combined with cache solutions for greater volatility information, allows to achieve greater efficiency and savings.

The next step of the solution is the integration with the data lake, under construction on the Big Data project, which will permit to obtain an increasingly more complete customer profile.

4.1.8 Product Hub

Product Hub integrates in a single layer the creation and configuration logic of a commercial product. Its main task is centralizing the sales logic of the individual legacy silos and managing the product configuration rules to ensure its governance through the Hub modules. The product / commercial component entity comes conceived as the sum of the different views and dimensions, federated and integrated with each other.

The Product Hub is the component that integrates in a single point logic of creation and configuration of a commercial product.

Its task is to centralize the sales logic of individual legacy silos of data and manage product configuration rules to ensure its governance through its own forms. It is divided into two branches: **Centralized Products Catalog** and **Conditions & Pricing**.

Centralized Single Products Catalog aims to centralizing in one single point:

- logic of creating and configuring a commercial product
- information (commercial and technical) of products
- rules for the sale of products.

Conditions & Pricing, instead, has the task of centralizing the pricing logic of the products in a single point. Each condition and price evaluation required by the Conditions & Pricing starts from the premise that the product has already been previously registered in the Single Centralized Product Catalog.

4.2 Description of withdrawal service transformation

Within the Agility Layer is described the new way of interfacing back-end functions by changing the vision of information system, considered no longer composed of individual applications but aggregation of services and components that collaborate and cooperate with each other to provide clear, simple functional solutions focused on the customer.

The services provided by new Enterprise Service Bus can be categorized into two main groupings:

- **Core system**: contains all the "elementary" services of the product, in jargon the parties, mainly related to the data to be managed and to the basic function to be supplied to the business. This group is strongly bound with Mainframe Business Components
- **Modular Apps**: contains all the business processes implemented by the information system, the result of the composition of the elementary core systems components on business or compliance rules.

An important difference between the two service groups is the frequency of change required, very low on the first group, very high on the second.

The new Custom Centric Architecture, based on this subdivision, aims to create a modeling of business services, on specific platforms, that maximize the speed of change and then aim to address the goal of "Agility".

To do this on the ESB are implemented and orchestrated:

- "Core services" called also "Atomic Services". They are the equivalent of Business Component on previous Mainframe based Architecture
- "Modular services" or "Composable Services" for fully automatic processes, made to orchestrate calls to atomic services in a stream
- "Long-term flows" for business processes with asynchronous human beings interactions which last for a long time, i.e. Mortage request procedure.
- Business rules which is a model of rules that drive the execution of a process according to the context in which it comes delivered.

The orchestration logic define the order of execution of the operational blocks and may also include business rules specifications of the requested operation, security, channel and rules deriving from legal regulations.

4.2.1 Description of new withdrawal service

In the third chapter it has been described how withdrawal service is developed and how it works on Mainframe architecture. The main attention point could be found in the role of Business Services which are in charge of orchestrating several Elementary Business Component and of guaranteeing the transactionality of each withdrawal operation. To do this, the service orchestration is split on different infrastructural layer (a part at application layer, the other one is performed by business services on Mainframe) and each one is in charge of invoke several services which belongs to heterogeneous areas of information system.

Introducing the Custom Centric Architecture, which embodies the Service Oriented Architecure principles in their entirety, it has become natural to rethink the withdrawal service in a new service oriented way. The transformation of the withdrawal service can be divided in three main operations:

- Business Components evolves in Atomic Services
- Orchestration only on Agility Layer
- Business Rules available only through Rule Engine

Business Components evolves in Atomic Services

Transformation of Business Components in Atomic Services was necessary to enable a complete orchestration on the Enterprise Service Bus. In the Business Components, in fact, are implemented particular customization which allows the correct dialogue with other important parts in Mainframe environment, as well as ISP Connector or ISP Middleware. Each Business Component required a specific invocation which cannot permit a real "service orientation" of mainframe orchestration layer. For these reasons, it was necessary re-engineer the business components according to the service oriented approach, creating new atomic services, which, for definition, have to be independent from any other service in the system.

The goal is to optimize the re-usability of atomic services which are made or modified very frequently. In other words, they should not coexist more versions of the same functionality, which, on the contrary, should be atomic and unique within the information system. Is appropriate for the customer to perform the same operation, for example a payment, in the same way even if he uses different possible connection channels (Internet, phone banking, ATMs, etc).

MPE

The "MPE" (Mainframe Performance Enhancer) component is born in response to the requirement to orchestrate the entire end-to-end service path on a single tool. Indeed, in cases where this orchestration involves more than two atomic services exposed by mainframe, to avoid that at every request of an atomic service is performed by a departmental[1]

[1] The term "departmental" in Intesa Sanpaolo IT infrastructure indicates every IT object (server, connection, layer, etc) which is not on Mainframe architecture.

connection to the mainframe, with consequent degradation of performance, becomes necessary to demand the execution of atomic services on mainframe to mainframe itself. The Mainframe Performance Enhancer is therefore characterized by:

- An integrated mainframe services development environment of the IIB toolkit, which ensures the design of the modular service with end-to-end visibility and a complete integration in ISP audit system;
- A LUW [2] management at the moment which are called multiple atomic dispositive services on Mainframe: these they must be able to be committed and rolled back together;
- Efficient performance management (response time e.g.) and consumption.

The code generated by MPE is a set of COBOL functions which call atomic services involved in the execution of simple logic and formatting data in input to services. It concentrates the execution of atomic mainframe services in a single point, ensuring through the ISP-Middleware the participation in the same LUW. In the current solution, Business Service ensured the transationality at the level of all CICS resources involved. In fact, the channel interacts only once with the Mainframe for the recall of the BS (this also induces to optimize Mainframe consumption). In the new architecture instead the orchestration is in a single point and the service on Agility Layer is used by all channels.

Orchestration only on Agility Layer

After transformation of Business Components in Atomic Services, all that remains is to orchestrated them on a single homogeneous layer, the Agility Layer. At this layer belongs the IBM Integration Bus, the Enterprise Service Bus adopted by Intesa Sanpaolo, which is in charge of orchestrating the several Atomic Services, like previously was done by Business Service with Business Components.

First of all, it was necessary analyze the whole withdrawal Business Service creating a flow chart with all business component invocations. From this analysis emerges:

- A small part of orchestration is made also by "Sportello" front-end application which invokes an entire Business Service to obtain all customer data.
- The biggest part of orchestration is made by a single Business Service which interacts with a lot of services, according to type of withdrawal request (by check or by form), belonging to different areas of mainframe. Mainframe architecture, in fact, is based on vertical legacy silos that include data, business logic and front-end for each "banking product" (i.e. Account, Loans, Credit Cards, etc.)
- Withdrawal operation, how described in the third chapter, consists in a double invocation to the same business service, once in simulation mode, once in dispositive mode. In this scenario it is very important guarantee a low latency of each atomic service and a correct transactionality.

[2] (Logical Unit of Work)

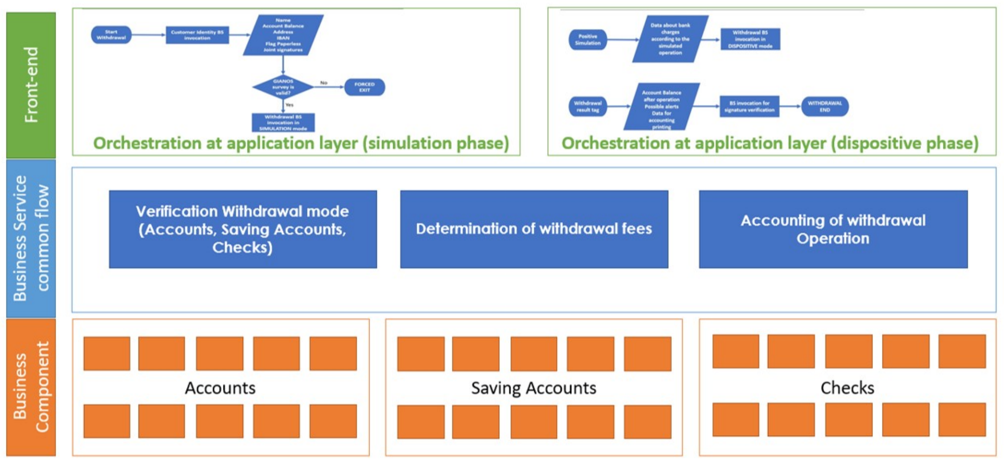

Front-end
Start Withdrawal
Customer Identity BS invocation
Name
Account Balance
Address
IBAN
Flag Paperless
Joint signatures
GIANOS survey is valid?
No
FORCED EXIT
Yes
Withdrawal BS invocation in SIMULATION mode
Orchestration at application layer (simulation phase)
Positive Simulation
Data about bank charges according to the simulated operation
Withdrawal BS invocation in DISPOSITIVE mode
Withdrawal result tag
Account Balance after operation
Possible alerts
Data for accounting printing
BS invocation for signature verification
WITHDRAWAL END
Orchestration at application layer (dispositive phase)
Business Service common flow
Verification Withdrawal mode (Accounts, Saving Accounts, Checks)
Determination of withdrawal fees
Accounting of withdrawal Operation
Business Component
Accounts
Saving Accounts
Checks

Figure 4.8. Logical schema of orchestration on Mainframe (AS-IS scenario)

- Some business rules are hard coded in COBOL procedure of several Business Components, making them not so flexible.

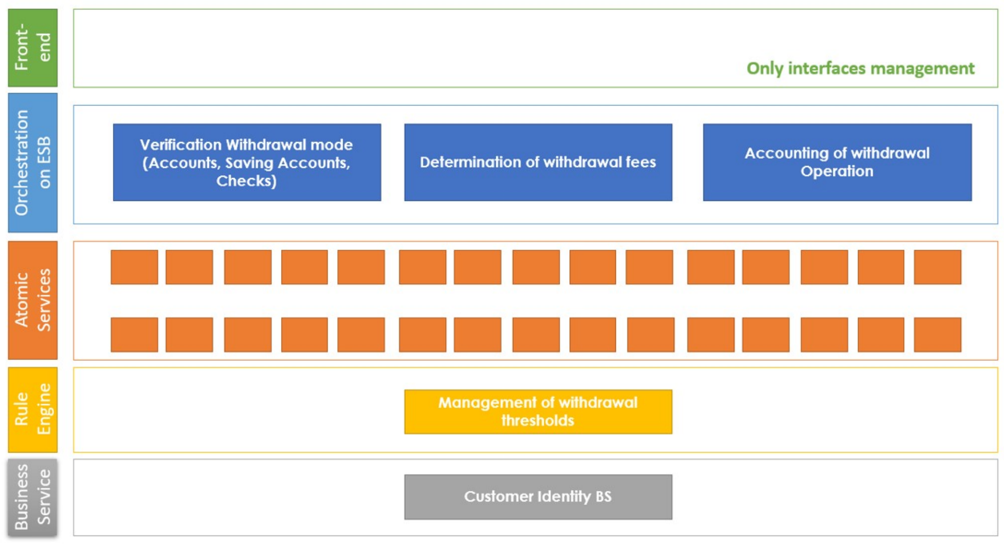

Front-end
Only interfaces management
Orchestration on ESB
Verification Withdrawal mode (Accounts, Saving Accounts, Checks)
Determination of withdrawal fees
Accounting of withdrawal Operation
Atomic Services
Rule Engine
Management of withdrawal thresholds
Business Service
Customer Identity BS

Figure 4.9. Logical schema of orchestration on ESB (TO-BE scenario)

Starting from these considerations, it was created a new flow chart in which emerges:

- The whole orchestration is now concentrated in a unique layer, the Agility Layer. The withdrawal service can be defined as a message flow on a dedicated Integration Server. Orchestration at front-end layer is not more required even if the invocation of

Customer Identity Business Service is already used because its redesign in service oriented mode does not bring any advantage in terms of functionality and optimization. It could be considered as a service which returns Customer Identity information.

- Invocation of Atomic Services now does not require to know where these services are deployed. In new flow chart are positioned in a single area because they are exposed by a MPE, which has in charge of invoke Atomic Services on Mainframe and retrieve their responses to the requesting nodes.

- All business rules are now deployed in the Rule Engine and no more hard coded rules are necessary inside Atomic Services code.

Business Rules available through Rule Engine

Withdrawal service involves different business rules which could be changed by new business organization or by new financial laws. On the mainframe architecture every business rule was written in a series of if-then-else statement implemented in COBOL code of each business component. A simple changement of a rule triggered an expensive and hard work by development teams, besides an high probability of error during changing management.

With the Rule Engine, each business rule is written once and every module which uses it can found it in the same place. In this way, if the rule must be updated, no changes can be deployed on applications which are using it.

4.2.2 Development of Composite Service as a flow on IIB toolkit

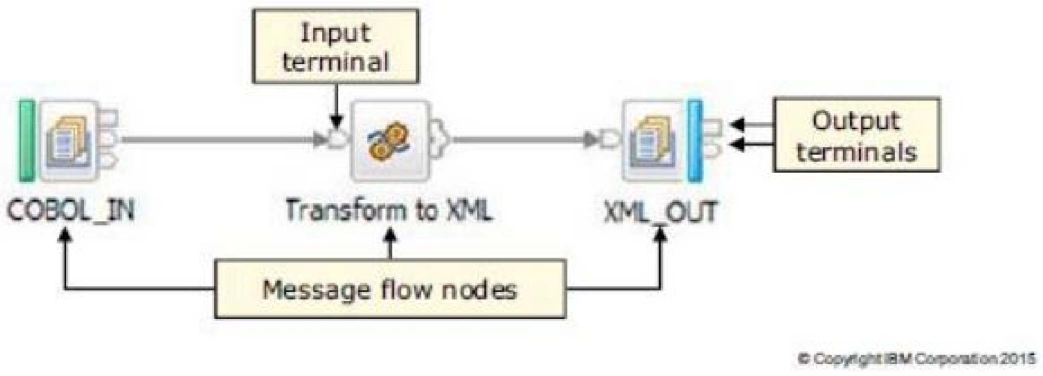


Figure 4.10. A message flow example.

Starting from the flow chart which represents the new orchestration of withdrawal service, it has been created a Composite Service using the visual Eclipse-based editor, called IIB Toolkit.

The native language with which the Composite Services must be developed is ESQL[3]; you can also use the Java language but this choice is not encouraged in developing teams because Java code is not so efficient in IIB node which are designed for ESQL code. The two main steps for the development of a service are the definition of the **interface** and its **implementation**.

Interfaces are a fundamental component in the creation of a service; they can be displayed in SOAP or REST format for synchronous services and in MQ for asynchronous services.

For the development of a Composite Service within the IIB Toolkit, **patterns** are provided.

The use of these patterns is mandatory and should not be considered as modifiable templates, as they already provide standard handling of calls, tracing and errors.

The creation of a composite service involves the subdivision into two subflows, one of interface and one of implementation:

- the interface flow includes the implementation sub-flow specifying the type of exposure that the service must have (Web Service SOAP, MQ).
- the implementation flow is technically a IIB sub-flow containing the orchestration logic provided by the end-to-end service.

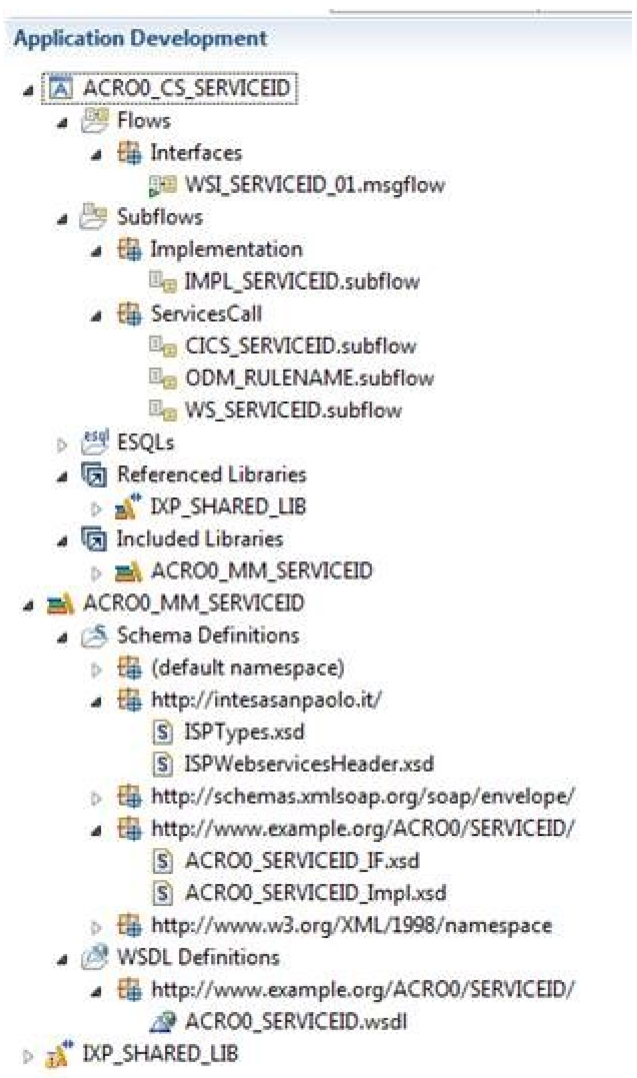
Interface flow and implementation flow are available under an unique IIB project tree which collects in a single point all resources involved in the service development.

Inside the project tree there is:

- "ACRO0_CS_SERVICEID" application contains the message flows representing the composite service
  - "ACRO0": Application Acronym[4];
  - "SERVICEID": service ID belonging to the acronym;
  - the "IMPL_SERVICEID" subflow contains the orchestration of the services;
  - subflows under the "ServiceCall" package contain the management logic of calls to atomic services or rules;
  - among the referenced projects there is the "IXP_SHARED_LIB" infrastructure library for the use of infrastructural nodes, it allows tracing, auditing and error management in a unified way for all Intesa Sanpaolo IT;
  - among the included projects there is the application library that contains the data structures (Message Model).
- Static Lib "ACRO0_MM_SERVICEID" will contain all data structures of the composite service

[3]Extended Structured Query Language (ESQL) is a programming language defined by IBM Integration Bus to define and manipulate data within a message flow.

[4]In Intesa Sanpaolo IT infrastructure an acronym represent a service identifier and all its software and hardware components are related to it.



```
Application Development
◢ ACRO0_CS_SERVICEID
    ◢ Flows
        ◢ Interfaces
            WSI_SERVICEID_01.msgflow
    ◢ Subflows
        ◢ Implementation
            IMPL_SERVICEID.subflow
        ◢ ServicesCall
            CICS_SERVICEID.subflow
            ODM_RULENAME.subflow
            WS_SERVICEID.subflow
    ▷ ESQLs
    ◢ Referenced Libraries
        ▷ IXP_SHARED_LIB
    ◢ Included Libraries
        ▷ ACRO0_MM_SERVICEID
◢ ACRO0_MM_SERVICEID
    ◢ Schema Definitions
        ▷ (default namespace)
        ◢ http://intesasanpaolo.it/
            ISPTypes.xsd
            ISPWebservicesHeader.xsd
        ▷ http://schemas.xmlsoap.org/soap/envelope/
        ◢ http://www.example.org/ACRO0/SERVICEID/
            ACRO0_SERVICEID_IF.xsd
            ACRO0_SERVICEID_Impl.xsd
        ▷ http://www.w3.org/XML/1998/namespace
    ◢ WSDL Definitions
        ◢ http://www.example.org/ACRO0/SERVICEID/
            ACRO0_SERVICEID.wsdl
▷ IXP_SHARED_LIB
```

Figure 4.11. IIB project tree used for withdrawal service development

– WSDL[5] and XSD[6] of interface;

– XSD of implementation and of the individual atomic services (XSD and DFDL[7]).

[5]The Web Services Description Language is an XML-based interface definition language that is used for describing the functionality offered by a web service.

[6]XSD (XML Schema Definition), a recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) document.

[7]Data Format Description Language published as an Open Grid Forum Proposed Recommendation in January 2011, is a modeling language for describing general text and binary data in a standard way.

Developing Interface Flow

The development of Interface mode is led by relative pattern in which the main modules ('nodes') are the following:

- SOAP Input

  Description: standard IIB node for displaying a stream as a Web Service. Type: node with wizard configuration.

- QoSP_SOAP: ('Quality of Service Provider')

  Description: Infrastructure library for exception handling, which returns any exceptions to the client in the Soap Fault structure. In the case of an exception, an Alert message is generated to the Maya application (MBRK0). Type: infrastructure node not customizable (does not have parameters to customize).

- MapSoapToImpl:

  Description: node of type compute that hooks the interface to the implementation. This node must be implemented to map SOAP input data to the internal format of the implementation sub-flow.

- <IMPL_IYBWSPREL2>:

  Description: service implementation subflow.

- MapImplToSoap:

  Description: node of type compute that enhances the SOAP response by setting all the fields. The node does not require parameters to be set. This node must be implemented to map the output data of the implementation sub-flow in the SOAP format of the interface.

- SOAP Reply:

  Description: standard node for closing a Web Service flow;

- AddErrorHandling:

  Description: allows you optionally to add a custom error handling after standard management by the 'QoSP_SOAP' node.

Developing Implementation Flow

Implementation flow contains the orchestration logic provided by the end-to-end service and, like Interface flow, can be developed by a pattern.

The calls described are linkable to each other by connecting the terminal node with the initial node of the segment of interest. In addition, calls are 'clonable' by copy-paste operation. It is also possible to introduce conditional orchestration logics (for example: if, loop constructs, etc.) using standard IIB nodes that do not modify the message (for example: nodes like Database, Filter and RouteToLabel).

The following guidelines must also be taken into consideration during the design phase:

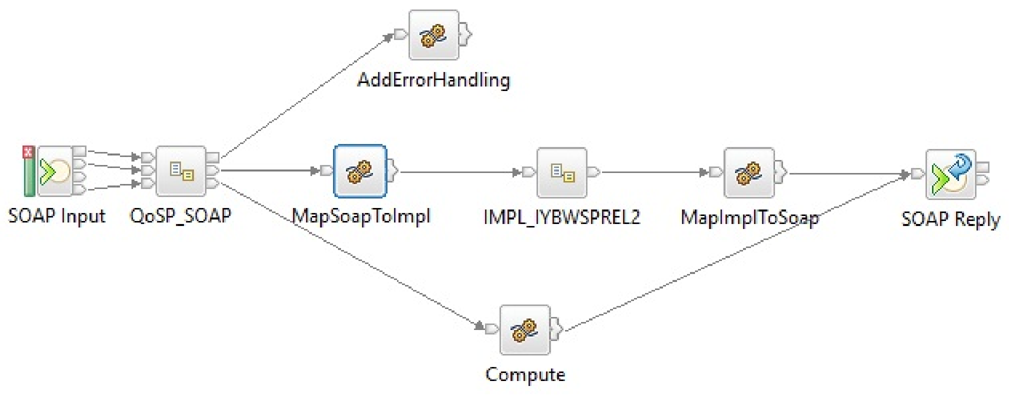


Figure 4.12. Development pattern for Interface flow

- All I / O paths must be in a static library
- Each application can contain multiple streams
- Each application can contain multiple static libraries and can refer to multiple external libraries.

There is currently no hierarchy of the IIB that maps the concept of 'acronym'. In other words, it is not possible to create a set of IIB flows that includes all the services of an acronym. It is instead possible to create multiple Integration Servers belonging to the same acronym, logically linking them according to the following naming convention:

- <ACRONYM>_EG01;
- <ACRONYM>_EG02;
- etc.

The sample pattern provides the possibility of making three distinct calls:

- the call made to an atomic Web Service Soap service;
- the call to the atomic service HOST through the Channel & Container protocol of the CICS node;
- the query to the ODM rule through a REST call

The implementation flow of withdrawal service, which appears as IMPL_IYBWSPREL2 node in interface flow, is formed by:

- An INPUT NODE which receives the input message

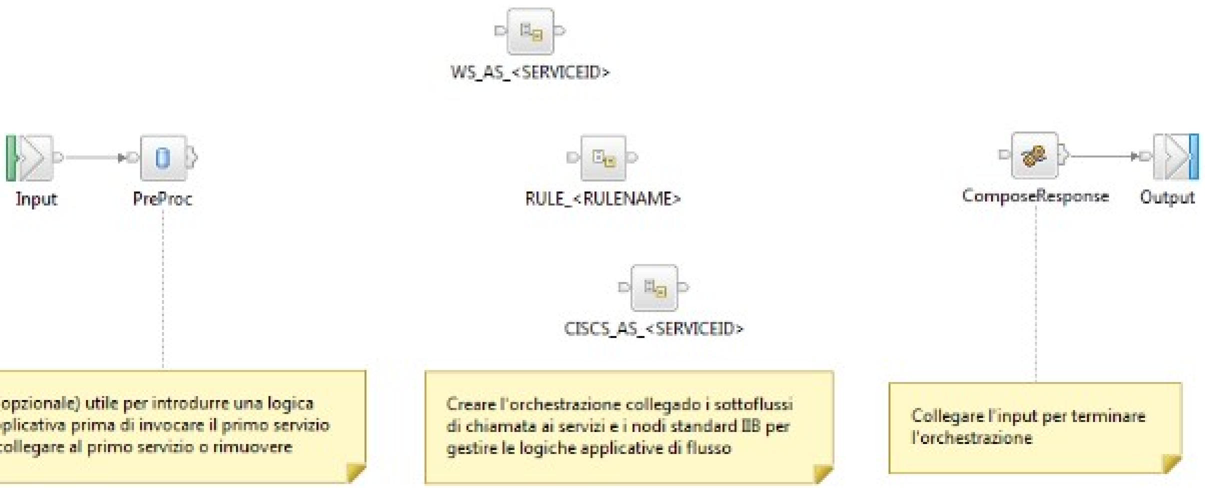

WS_AS_<SERVICEID>
Input
PreProc
RULE_<RULENAME>
ComposeResponse
Output
CISCS_AS_<SERVICEID>
- (opzionale) utile per introdurre una logica applicativa prima di invocare il primo servizio
- collegare al primo servizio o rimuovere
Creare l'orchestrazione collegado i sottoflussi di chiamata ai servizi e i nodi standard IIB per gestire le logiche applicative di flusso
Collegare l'input per terminare l'orchestrazione

Figure 4.13. Sample pattern for Implementation flow

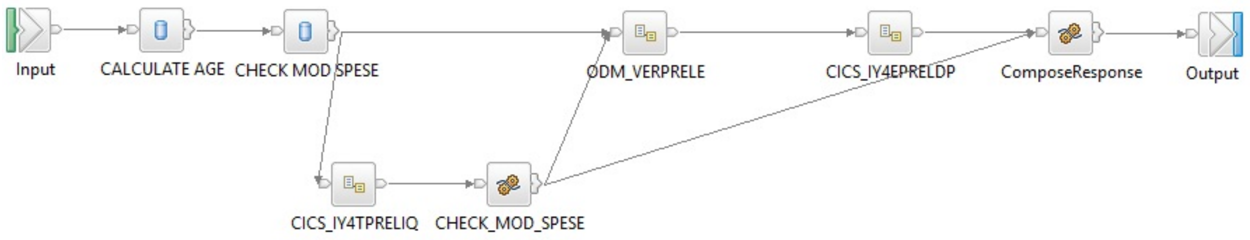


Figure 4.14. Development of Implementation flow

- A Database Node with label CALCULATE AGE which calculate if customer has 18 years or not
- A Database Node with label CHECK MOD SPESE which calculate the amount of charges for the operation
- A CICS Node with label CICS_IY4TPRELIQ which perform a CICS call to atomic services for simulating withdrawal operation
- A CICS Node with label CICS_IY4TPRELDP which perform a CICS call to atomic services for dispose withdrawal operation
- A Compute Node with label CHECK_MOD_SPESE

- A ODM Node to perform a query toward ODM in order to obtain correct threshold for current withdrawal operation
- A Compute Node with label ComposeResponse which allows to compose the entire SOAP message that is retrieved to calling service
- An Output Node which retrieve message previously composed by ComposeResponse node.

4.3 Re-usability on ESB: cardless withdrawal service

One of the most important aspect of customer centric architecture is the modularity: a service, for its definition, is "a logical representation of a repeatable business activity that has a specified outcome". In other words, once a service implements a function which is repeatable, it can be invoked by a lot of other components without any problem.

Starting from this principle, in Intesa Sanpaolo it was developed a new cardless withdrawal service that reuse many components of standard withdrawal with great savings in terms of time to market, development agility and costs.

4.3.1 Description of cardless withdrawal service

The goal is to allow ATM withdrawals without using chip card. To do this, it was decided to use the user's mobile phone and the App of the bank installed on it as the primary vehicle for the withdrawal. The whole phase of authentication and inquiry with host takes place through the mobile APP, at the ATM layer remains the physical management phase, that is the only cash dispensing for the withdrawal.

Prerequisite is the signing of a digital identity contract and the "Your bank" App installed on the customer's mobile phone.

4.3.2 Integration and implementation

To perform a cardless withdrawal, customer has to:

1. Press the function key on ATM to trigger the QR code generation function that will be shown below on the screen.
2. Frame the QR code with his smartphone and wait a few seconds: the app after capturing the QR code image through its back-end calls the QR code verification services exposed by the ATM back-end. Then user has to confirm with fingerprint or with his PIN.
3. Choose the account on which to charge the sum and the withdrawal amount, check that the information entered is correct and confirm to withdraw. After QR code verification the Mobile App through its back-end makes the call to IIB that orchestrates the atomic service which locks the requested amount on the user's account. Once the block result is received, IIB invokes the operation confirmation service exposed by the ATM back-end to complete the ATM withdrawal operation.

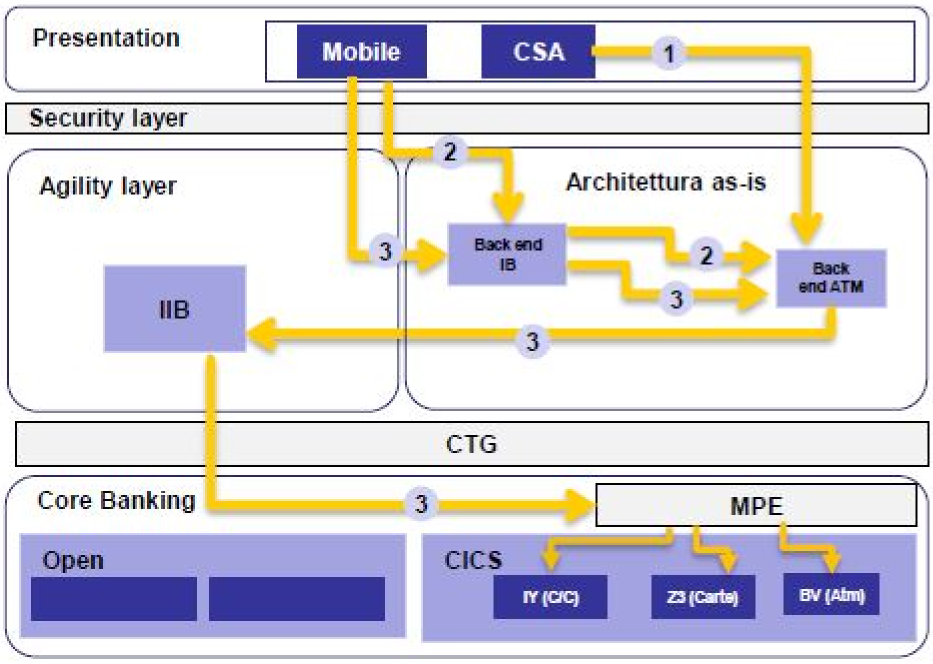


Figure 4.15. Architectural schema of cardless withdrawal.

Cardless withdrawal service is the concrete example of a widespread integration on Intesa Sanpaolo information system. In this single service cooperate back-end services (like Internet banking and ATM), CICS transactions on Mainframe coordinated by MPE for guaranteeing transactionality and IIB flows.

The development of this service required less time of other services because its logic core is entirely written in a IIB flow which re-use atomic services previously transformed for standard withdrawal service.

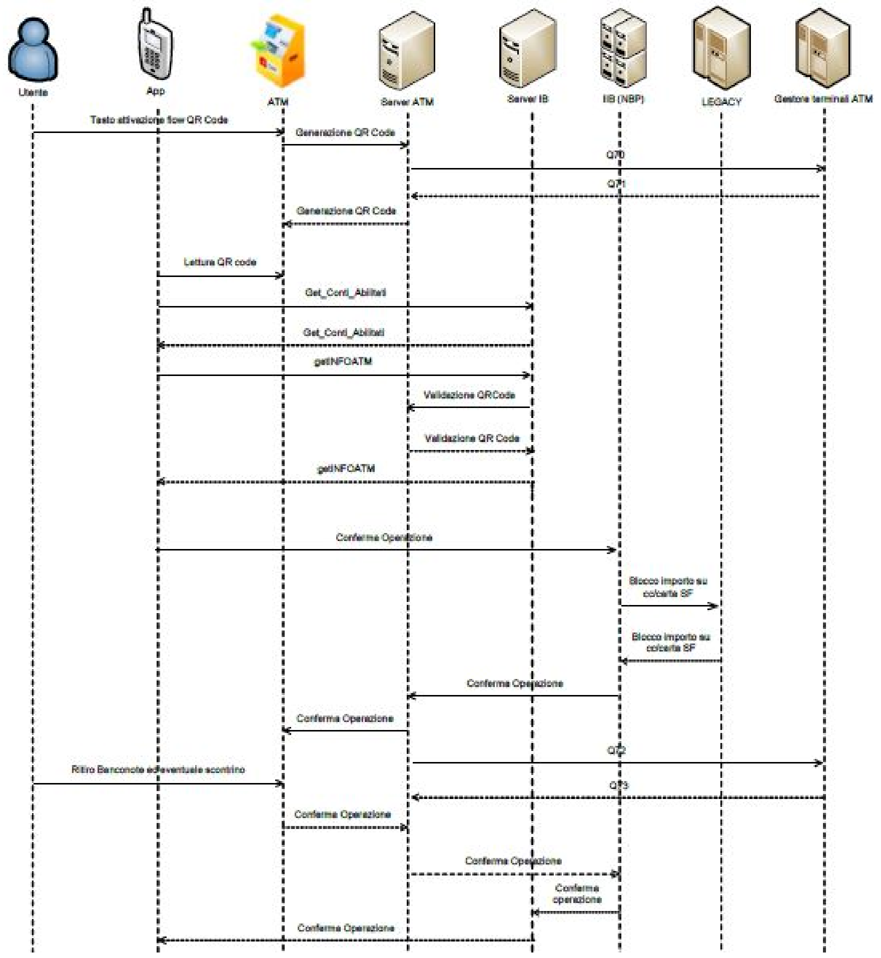


Figure 4.16. Use case schema of cardless withdrawal.

Chapter 5

Evaluation of withdrawal service transformation

5.1 Requirements

Before each transformation or project it is necessary to identify the functional and non-function requirements. In the following paragraphs there are requirements which led to withdrawal service transformation from Mainframe to Enterprise Service Bus orchestration.

5.1.1 Functional Requirements

Among function requirements of withdrawal service:

- The withdrawal operation allows cash withdrawal at bank office from the customer's bank account or from savings account. The operation can be performed in four different ways:
  - by a form
  - by a check
  - "Third Party Withdrawal"
  - Simple Credit
- Withdrawal application must be used by bank employees with specific security and authorization profile.
- Customer who wants do a withdrawal must have previously filled a GIANOS survey otherwise operation cannot be executed.
- Withdrawal service needs to interact with different bank services like Customer Identity Service, Check Service, Saving Account Service, Account Service.

5.1.2 Non-Functional Requirements

- Performance: Response Time of front-end application, which include also network response time, must be lower than 3 secs for each page loading.
- Availability: Withdrawal service must be available during branches worktime. Some branches are opened also on Saturdays.
- Modularity: new service must use modular services which allows to reuse them by other new services.
- Interoperability: must be used standard protocols of exposure of the services despite of technical reference platforms.
- Simplicity: must be used common templates for development and test because they allow to concentrate development only on the service and not on the infrastructure aspects
- Standard: must be adopted a common development standard which increases the knowledge and management of the entire ESB architecture.

5.2 KPI

5.2.1 Reuse rate

As first approach to evaluate impact caused by transformation of withdrawal service, it has been analyzed reuse rate of the old components in the Customer Centric Architecture.

By convection, the orchestration of the service on the Mainframe through Business Services and Business Components will be considered as AS-IS scenario, while the TO-BE scenario will be the one that sees the service orchestrated by the ESB.

	AS IS	TO BE			DELTA
COMPONENT		Reused	New	Deleted	
Business Service	1	0	-	1	-1
Business Component	26	10	-	4	-16
Programs/Routines	0	0	-	0	0
Atomic Services	-	-	9	-	0
Rules	-	-	1	-	0
IIB flows	-	-	1	-	0
MPE	-	-	2	-	0
Total	27	10	13	5	

Table 5.1. Old and new objects involved in To-Be Scenario

First of all, as shown in table 5.1 we proceed to count the components belonging to the two scenarios. For each component in the TO-BE scenario, it will be indicated how many of the old components have been reused and how many have been removed. Reading the column "DELTA" it is immediately visible that many of the old components have been removed from the new implementation of the service because more components have been incorporated into a single Atomic Service or in a single ODM Rule. In general 17 old components, which appears in delta column have been transformed in 13 new components.

In the lower part of table 5.1 you can see how many new components have been implemented in the new scenario. Consider that these new objects replace the deleted objects (not necessarily in a one-to-one relationship): for example, the withdrawal service business service has been replaced by a IIB flow. Starting from this component census it is possible to calculate a first approximation of the percentage of reuse of objects in the TO-BE scenario.

- Total number of involved objects in TO BE architecture: 23
- Total number of new objects: 13
- Total number of old objects: 10
- Total number of deleted objects: 5
- Percentage of re-utilization: 43,48%

This first percentage can not be defined as reliable since it has been hypothesized that each reused component has the same weight as a new component.

	Amount of objects in to be scenario								
	Reused			New			Deleted		
Complexity	Low	Medium	High	Low	Medium	High	Low	Medium	High
Business Service	-	-	-	-	-	-	-	-	1
Business Component	2	4	4	-	-	-	-	-	4
Programs/Routines	-	-	-	-	-	-	-	-	-
Atomic Services	-	-	-	4	4	1	-	-	-
Rules	-	-	-	-	-	1	-	-	-
IIB flows	-	-	-	-	-	2	-	-	-
MPE	-	-	-	-	-	1	-	-	-
Weighted Total	2	8	12	12	36	98	-	-	5

Table 5.2. Weighted components in To-Be scenario

To obtain a more truthful KPI, it was decided to create a weight table (Table 5.3) in which to each object is assigned a different weight based on the technology implemented and its complexity.

Thanks to the weight table it is possible to recalculate the weight of each component of the TO-BE scenario obtaining in Table 5.3 an overview of the impact of the re-use of old components on the new architecture. It is natural to associate the components belonging to the new architecture with a higher weight because these objects meet many more requirements than those discussed in the previous paragraph. Finally, a new percentage of the re-use rate of the old components is obtained, compared with the use of the new ones.

It appears evident from the graph in figure 5.1 that the reuse of the old components is very limited and has been applied only for business components that are not very relevant within the withdrawal service flow.

	Weights of objects in to be scenario								
	Reused			New			Deleted		
Complexity	Low	Medium	High	Low	Medium	High	Low	Medium	High
Business Service	2	4	6	-	-	-	1	1	1
Business Component	1	2	3	-	-	-	1	1	1
Programs/Routines	1	2	3	-	-	-	1	1	1
Atomic Services	3	9	20	3	9	20	1	1	1
Rules	1	2	3	1	2	3	1	1	1
IIB flows	10	15	25	10	15	25	1	1	1
MPE	-	-	-	10	25	15	-	-	-

Table 5.3. Weights table

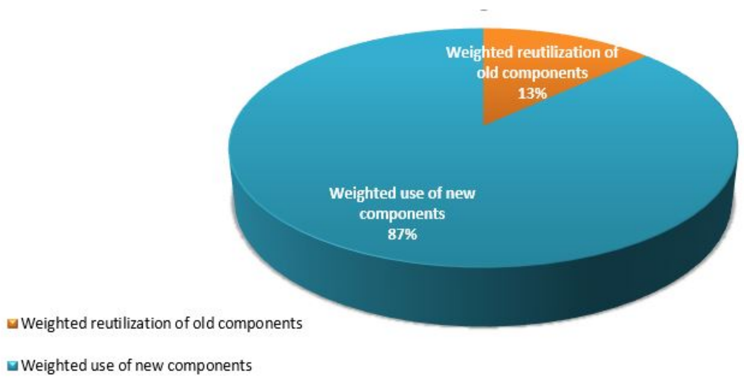


Figure 5.1. Reusability graph of old components in new withdrawal service.

5.2.2 Performance Times

To perform an assessment of the performance caused by the transformation of the withdrawal service, several tests were conducted in a dedicated environment.

These tests and the relative measurements were carried out on architectures that are more similar to those used in production.

It should be noted, in fact, that in the Customer Centric Architecture scenario (figure 5.2) the security layers were considered both towards the outside and between the departmental environment and the mainframe environment. Similarly, the impact of the ISP Connector and communication through MQ was considered in the scenario on the Mainframe (figure 5.3).

The performance test involved the execution of 5000 withdrawals in the Customer Centric Architecture environment and the same input was entered as a load of Business Services

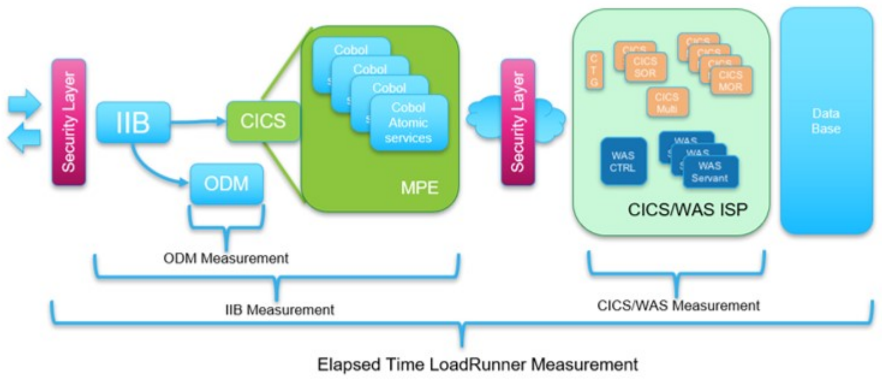


Figure 5.2. Schema of ESB orchestration environment used in performance test.

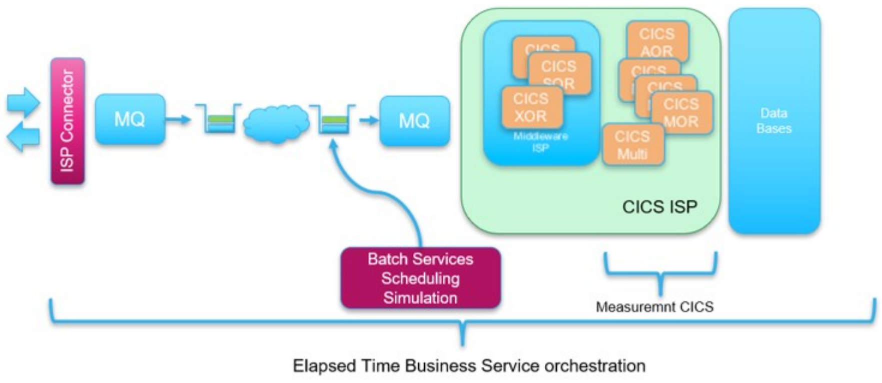


Figure 5.3. Schema of Mainframe orchestration environment used in performance test.

on the old system.

Subsequently, various measurements were made and in particular were taken into consideration:

- CICS Response time in the orchestration by IIB flow compared with that in the orchestration on the Mainframe
- CICS CPU times for the execution of a transaction of the old architecture against the new one.
- Communication times introduced in the new architecture between the ESB and the Mainframe: in fact ESB is attested on a different network than the Mainframe's one.

Unfortunately for reasons of confidentiality it is not possible to report the absolute values of the measurements but we can state that:

- CICS Response times of the orchestration (considering the two invocations: one of simulation and one in dispositive mode) in the new architecture are greater than those on mainframe of about 20%.

  The call of the withdrawal service entirely done on the mainframe environment owes its speed to the fact that all elaboration is managed by a single point, that is on the mainframe and, although many business components must be orchestrated by a business service, it does not need to go out on the network as well as with the ESB, which is installed on a different network than the mainframe's one.

- CICS CPU times for the execution of a new architecture transaction are lower than those on the old infrastructure of around 11%. This result can be justified by the fact that in the Customer Centric Architecture, the dispositive functions were divided from the simulation to manage the functional flexibility of the process, thanks also for MPE business components aggregations. In this way it is performed a better and more efficient CICS call which does not needs to invoke each single business component but, exploiting MPE features, groups them in a single call. In the old Business Service they were managed by the same multi-functional transaction.

- Communication time for calls to Mainframe from the ESB between different networks is approximately 60-70 ms per request. In this way in new architecture it is introduced a latency that does not exist in old environment.

5.2.3 Scalability

The scalability test involved performing multiple tests in an Independent-Test environment by varying the number of threads that enter workloads on the new system with the ESB.

Using as a reference the number of withdrawals actually made by customers in the peak hour, this test was carried out loading the system with an ever increasing number of requests.

In the first instance, a number of requests equal to those recorded during the ordinary operating hours has been entered into the system, thus obtaining an estimate of the systems' basic response time.

Subsequently, a number of requests equal to the peak requests was entered. In this step the response time had an insignificant increase that did not differ so much from the base response time.

A first increase in response times was recorded when the number of requests entered into the system was equal to one and a half times those of the peak time. In this case there was an increase of about 20% of the response time compared to the base one.

The response times then recorded a linear growth trend according to the number of requests.

The breakpoint, in which the response times began to be double or triple with respect to the base response time, was reached when loaded operations were equal to 3 times the number of operations in the peak time.

5.2.4 Time distribution in elapsed time in ESB orchestration

Another important KPI for the evaluation of the process of transformation of withdrawal service is the time distribution in elapsed time in ESB orchestration.

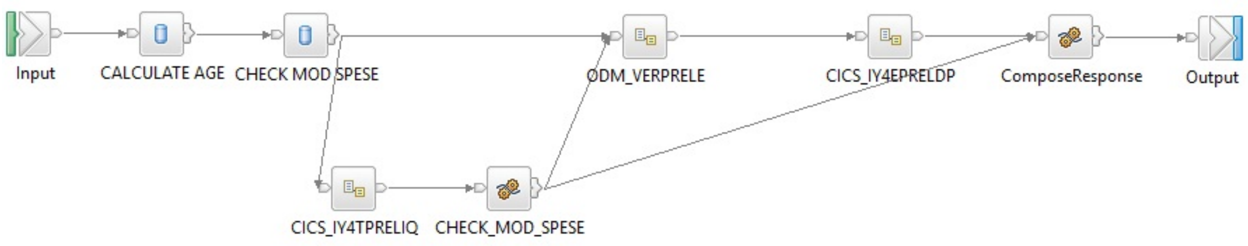


Figure 5.4. Withdrawal implementation flow

The execution times of the IIB flows have the following elapsed time distribution:

- Over 80% of the time is used to wait for the response from the MPE/CICS transactions and in particular for the IY4D (dispositive) transaction
- 2-4% for requests to ODM services, where over 70% of the time is charged to IP communication between servers
- Less than 7% of the time is to perform the choreography logic on IIB

5.2.5 Final KPI considerations

At the end of all these tests it is possible to obtain a fairly clear picture of the effects that the transformation of the withdrawal service has brought. In general, there has been a slight increase in some service response times due to the more complex technological infrastructure on which the new Customer Centric Architecture is based, compensated however by a more efficient management of CICS calls and by a lower application complexity. In a large company, like Intesa Sanpaolo, introducing a more complex technological architecture, made up of many more layers that need to connect to each other, always entails greater efforts in terms of both economic resources and technical skills.

The question that arises, therefore, is: "Why do I invest in a new orchestration with ESB when the mainframe is still able to provide the same service with comparable response times and with an architecture that requires less infrastructural layers?"

The first is an economic driver: the mainframe is not like ordinary computers, which companies pay and then use. The cost of running a mainframe depends on how much storage space is needed (let's not forget that the application DBs also reside on the mainframe) and how much processing it does. In other words, users pay both for capacity and

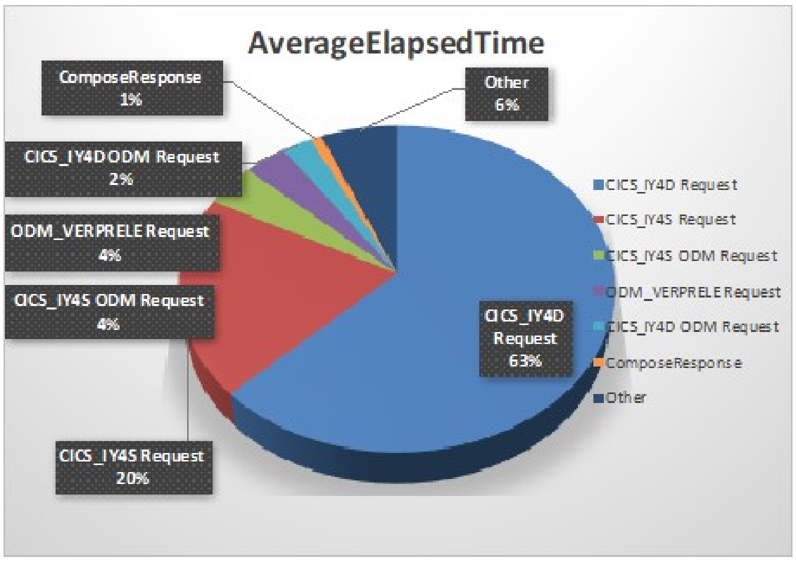


Figure 5.5. Average Elapsed Time distribution in ESB orchestrated withdrawal flow.

for consumption when using a mainframe. The unit of measure that allows programmers and system administrators to determine the performance of a mainframe is the MIPS that stands for Milions of Instructions Per Second.

It is clear that the transformation of a service totally developed on Mainframe in a service developed on open infrastructure that recalls the mainframe only for a part of the functionalities allows the company considerable savings, considering that it is a service widely used by customers.

The second driver, more important than the first one in some respects, is that this transformation of the service enabled a much faster and more efficient development pattern. The creation of Atomic Services to replace several Business Components has made these services available to new functionalities, allowing faster development and fewer errors. In this way, when developing an atomic function, it is made available to application development groups "functional blocks" ready to use that can be handled as a black box.

Chapter 6

Future evolution of Intesa Sanpaolo ICT: New Digital Architecture

In previous chapters we have seen how it was possible to transform the information system of Intesa Sanpaolo by integrating core services on legacy technologies with new services through the use of ESB.

The next step for the bank to keep on this continuous integration will be to the Custom call into question Centricity Architecture to allow a new integration with the latest development patterns, such as those oriented to microservices. To do this was born the New Digital Architecture, which is still focused on customer centricity but keeping in mind that customer can interact with bank in a more and more pervasive way.

6.1 Objectives

New Digital Architecture's targets are:

- Enable the evolution of new digital channels to ensure a uniform user experience on all "touch points" and facilitate the rapid introduction of new products
- Ensure "Always On" by limiting the use of batches in favor of synchronous logic and exploiting the opportunities made available by Cloud solutions
- Reverse the growth trend of infrastructure consumption (mainframe, open and storage) induced by the customer-centric evolution of digital channels (i.e. Mobile and advanced ATMs)
- Ensure the quality, consistency and coherence of data between systems, respecting standards of security, privacy and compliance
- Improve software quality through the adoption of new development and delivery techniques (i.e. DevOps) together with automation and simplification of management and operational processes

6.2 Logical model and main components

The Logical model of New Digital Architecture in Intesa Sanpaolo is the obvious example of a continuous integration of a information system. Legacy technologies and the most recent IT patterns come together in an unique information system which is tailor-made on the needs of the bank. Inside this model it is possible to recognize five macro cluster which

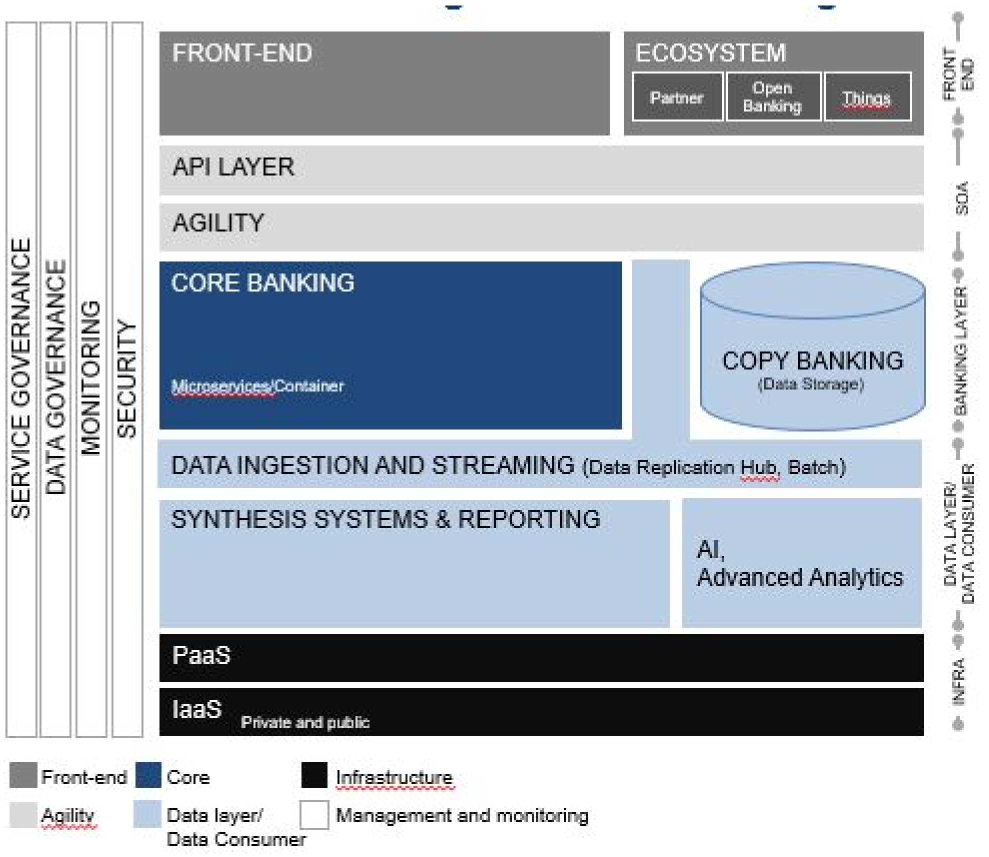


Figure 6.1. New Digital Architecture logical model

represent fundamental elements of information system:

- **Front-end** is the first user approach to the system: so it is important to guarantee consistent and natively omnichannel user experience also through different touch-points. It does not manage business, process and product logic and does not include replicas of business data.

  Web applications are implemented with a single framework and with a single application architecture while mobile and tablet ones will be implemented, based on functionality, as a native or hybrid App.

  At this layer are necessary omnidevice solutions which use "unique APIs" (no business data, no business logic) which allows to communicate with external ecosystem made by partners, third-part applications and IT objects.

- **Service Oriented Architecture**, which belongs to previous Customer Centric Architecture, finds in this new architecture the right place for accelerating information system transformation.

  In facts the decoupling layer for the orchestration of complex transactional logic and long-running processes allows to maximize the flexibility and reuse in development of new applications.

  If in Customer Centric Architecture Web Services can be considered one of the most used methods of service invocation, now New Digital Architecture is focused on API as a paradigm to access or provide services, both internal and partner ecosystem.

- **Banking Layer** has two souls: the first is the traditional one made by Core Banking which is the layer that provides the resources (i.e. data) on which to apply the business logic, the second one is the Copy Banking and its Data Storage of replicated data (Near Real Time or Batch) oriented to inquiry access, synthesis systems and new digital services.

  In other words at this layer there are transactional bi-modal systems that, depending on the context, dynamically invoke the Core or the Copy Banking.

  On the Core Banking, Mainframe disposition operations expect invocation of CICS transactions orchestrated via ESB on Agility, while for informative operations there will be a transformation into microservices. Nevertheless microservice transformation is applicable to the whole architecture, evaluating the case studies.

- **Data Layer/Data Consumer** with its Data Ingestion and Streaming component allows to replicate and exchange data in order to avoid direct communication between applications and the redundancy of the master data. The introduction of a Dataflow system will make bank's data architecture open, known and reusable thanks to standardization and centralization of management and data exchange between systems.

  This infrastructure enables massive data processing (batch re-platforming) through a gradual transition from a batch logic to an event logic that enables updates almost in real time.

- **Infrastructure** will able to manage the service and microservices life cycle and their horizontal scalability thanks to the management of "Cloud-native" solutions implemented by Platform as a Service and Infrastructure as a Service solutions.

6.3 Microservice centrality

6.3.1 What is a microservice

Microservices are a software development technique in which a single application is seen as a suite of small services, each running in its independent process and communicating with lightweight protocols, often an HTTP resource API. One of the main benefit of use

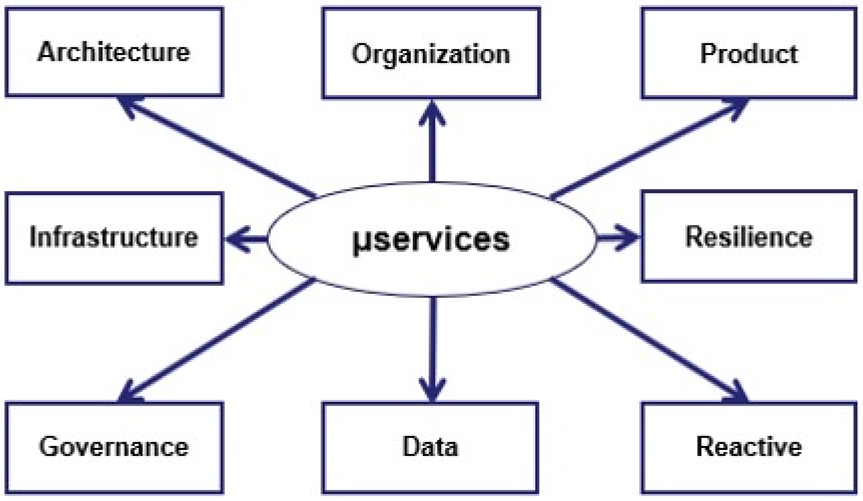


Figure 6.2. Microservices influence many aspects of an information system

microservices is that they improve modularity. This makes the application easier to understand, develop, deploy and scale respective services. Microservices-based architectures enable continuous delivery and deployment.

The most interesting characteristics of microservices include:

- Componentization: the ability to replace parts of a system independently from the others.
- Organization around business capabilities instead of around technology.
- Decentralized data management with one database for each service instead of one database for a whole company.
- Infrastructure automation with continuous delivery being mandatory.
- Independence brings benefits on many aspects about the life cycle of the software.

6.3.2 When and how to use microservices

Microservices are characterized by agility and scalability advantages, unreachable for any other paradigm, so consider adopting for applications that have extreme requirements in these areas. All IT actors are developing frameworks and platforms that helps in adoption of the microservices architecture.

Creating a complete microservice infrastructure requires composition and integration of a number of supporting tools to create what Gartner calls the “outer architecture.” It delivers the platform capabilities needed to help microservices work together to deliver flexible development and deployment.

For this reason, if do not appear agility and scalability requirements for an application, it is not convenient adopt microservices.

There are other ways to increase agility. Focus on development and deployment agility of larger components before attempting to do it for a swarm of dynamically evolving microservices. Most organizations will find microservices too complex, expensive and disruptive to deliver a return on the investment required.

Meanwhile, organizations can derive tremendous value from the mini-services model, which advocates a more pragmatic, coarse-grained services approach with relaxed architectural constraints.

6.3.3 Microservices in Intesa Sanpaolo

The use on microservices in Intesa Sanpaolo can be displayed in three different scenarios, each one specific for particular application requirements.

- **Front-end in a single component** - Applicable in contexts where continuity in the user experience is required

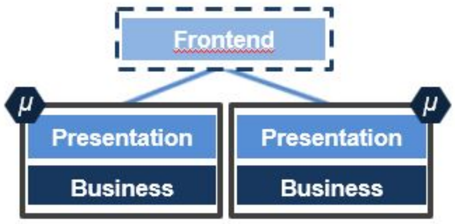


Figure 6.3. Front-end in a single component scenario.

  Main characteristics:

  - Functional decomposition applied to the business logic
  - Consumer-oriented reuse / exposure of REST services
  - Horizontal scalability
  - DB is not included for functional and technological limits.

- **Front-end in each microservice** - Applicable in contexts of modular applications

  Main characteristics:

  - Functional decomposition applied to an entire functionality
  - Consumer-oriented reuse / functionality exposure
  - Coupling between front-end and business logic
  - DB is not included for functional and technological limits.

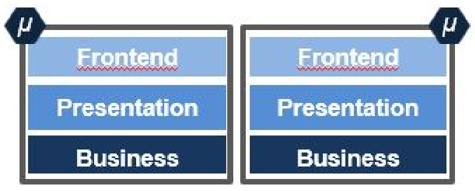


Figure 6.4. Front-end in each microservice scenario.

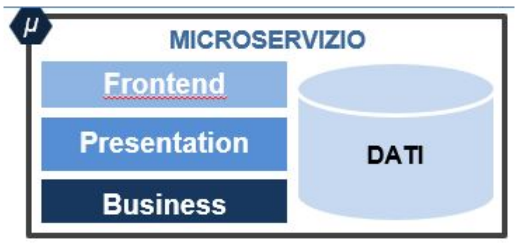


Figure 6.5. Full microservice scenario.

- **Full microservice** - Applicable in contexts where it is possible to isolate the data model and include the database in the microservice

  Main characteristics:

  - Suitable for new isolated features to be developed on the Open world
  - Independent
  - Individually scalable
  - Limited portability for large applications

Conclusions

In this thesis, it was shown how a complex information system like that of a large company, such as Intesa Sanpaolo, can continue to integrate with increasingly more current and complex technologies. In the last few years, as already stated in the first pages of this work, it becomes more and more important for companies to be able to offer services to customers as sensitive to their needs, and this inevitably goes through the use of the latest technologies.

Therefore, the approach Intesa Sanpaolo has adopted to update its infrastructure, strongly focused on legacy solutions (Mainframe), has been shown. It is unthinkable in this context that an entire infrastructure is abandoned to make room for a new technology, for this reason, the only viable approach remains that of continuous systems integration.

The ESB can therefore be considered the keystone that allows a "mainframe-oriented" information system to evolve towards a real service-oriented architecture that does not exclude the legacy component, but that instead manages to exploit it at its best (think about the improvement of the CICS CPU time analyzed in chapter 5) alongside the more advanced products. Integration through ESB has allowed to realize also a new development framework able to simplify the work of the application groups, enabling them to realize advanced services, such as the cardless withdrawal illustrated in chapter 4, exploiting the modularity of the components and the independence of services.

All these advantages, listed so far, have costs not only in economic terms. As illustrated in the last part of chapter 5, the new Customer Centric Architecture has involved a greater complexity of the infrastructure that has seen involved: the ESB, the Rule Engine, the Service Catalog, the MPE, etc. : all components that were not present in Intesa Sanpaolo's Enterprise Architecture and therefore need not only to communicate with each other in order to work but they also need people able to manage and maintain them: this affects management costs and systems response times that need more resources to guarantee SLAs typical of the system information system of a systemic bank such as Intesa Sanpaolo.

Naturally, the system integration process can not be considered concluded with the introduction of the ESB, but we must continue in this direction by developing increasingly scalable and reliable solutions. A valid help in this area can come from architecture oriented to microservices and containerizable infrastructures.

Bibliography

[1] Anthony, R. N. , *Planning and Control: a Framework for Analysis*, Cambridge MA, Harvard University Press, 1965.

[2] Douglas K. Barry, https://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html , 2018.

[3] KPMG, https://home.kpmg.com/it/it/home/insights/2017/07/KPMG-CIO-Survey-2017.html, 2017

[4] VV. AA., *Enterprise Service Bus*, https://www.techopedia.com/definition/24746/enterprise-architecture-ea, 2018

[5] Intesa Sanpaolo, *History*, https://www.group.intesasanpaolo.com/scriptIsir0/si09/chi_siamo/eng_storia.jsp#/chi_siamo/eng_storia.jsp, 2018

[6] IBM, *IBM MQ (già IBM WebSphere MQ)*, https://www.ibm.com/support/knowledgecenter/it/SSFKSJ/com.ibm.mq.helphome.doc/product_welcome_wmq.htm, 2018

[7] IBM, *IBM Integration Bus introduction, Message flow nodes*, https://www.ibm.com/support/knowledgecenter/en/SSMKHH_10.0.0/com.ibm.etools.mft.doc/ac12640_.htm, 2018

[8] Mike Ebbers, John Kettner, Wayne O'Brien, Bill Ogden, *Introduction to the New Mainframe z/OS Basics*, IBM RedBooks, 2011

[9] Barbara Calderazzo, *Enterprise Service Bus: tutto quello che devi sapere*, https://www.interlogica.it/insight/enterprise-service-bus-cos-e-benefici/, 2018

[10] VV. AA., *Microservices*, https://en.wikipedia.org/wiki/Microservices, 2018

Ringraziamenti

I miei ringraziamenti vanno:

A mia madre, a mio padre, a mio fratello, al loro supporto morale e materiale, alle loro chiamate nei momenti più bui e più luminosi, alla distanza fisica che è cresciuta tanto quanto il senso di appartenza ad una vera Famiglia. A voi posso solo dire grazie.

Alle zie, agli zii, a quelli di sangue e a quelli acquisiti, a chi è qui oggi, a chi non può esserci, a chi è sempre con me. A voi che mi avete insegnato l'umiltà, il coraggio, l'intraprendenza, la caparbietà.

A mia nonna Mafalda, al suo eterno preoccuparsi per il mio benessere, alle sue attenzioni che resteranno sempre impresse nei miei ricordi. A tutti i miei nonni che nell'eternità del tempo sono sicuro hanno assistito e assisteranno ad ogni attimo della mia vita.

Alla grande famiglia allargata in Intesa Sanpaolo ed in particolare a Ezio, a Federica, a Liliana, a Michela, a Manuel, a Nicola, ai colleghi del gruppo Data Communication e a tutto il gruppo del "Supporto Open". A voi che mi ha insegnato tantissimo e mi insegnerete ancora tanto. Alla "Progettazione DB" che mi ha "adottato" negli ultimi mesi. A tutti voi che avete contrubuito in tanti modi a questo lavoro di tesi con la conoscenza, con l'entusiasmo, con l'incoraggiamento.

Al professore Maurizio Morisio, per avvermi accompagnato in questo lavoro finale del mio percorso accademico.

Ad Armando e a Claudio, i compagni di questa avventura qui a Torino che ci ha regalato momenti gioiosi e quelli in cui ognuno ha avuto bisogno dell'altro, alla grande tribù del Sangiovannino capitanata dal mitico Don Natale che mi ha accolto quando ero solo un ragazzino spaesato, a Michele e a Leonardo, portatori autentici dell'Irpinia più vera anche lontano da casa. Un grande grazie a Gianluigi per esserci, anche stavolta.

A tutti quelli che ho dimenticato, per pura distrazione sappiatelo, a chiunque anche incosapevolmente mi ha donato qualcosa permettendomi di essere ciò che sono.