Giovanni Henrique Silva Oliveira

DEVELOPMENT OF A MESSAGE BROKER

1 2 9 0

UNIVERSIDADE Ð COIMBRA

Giovanni Henrique Silva Oliveira

DEVELOPMENT OF A MESSAGE BROKER

VOLUME 1

Internship Report in the context of the Masters in Informatics Engineering, Specialization in Engenharia de Software advised by Professor Vasco Pereira and engineer Hugo Duarte Fonseca and presented to Faculty of Sciences and Technology / Department of Informatics Engineering.

January, 2021

Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

DESENVOLVIMENTO DE UM BROKER

Giovanni Henrique Silva Oliveira

Dissertação Desenvolvimento de um Broker no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia Informática orientada pelo Professor Vasco Pereira e Engº Hugo Duarte Fonseca e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Janeiro de 2021

1 2 9 0

UNIVERSIDADE Đ
COIMBRA

Abstract

Nowadays a growing number of systems and applications are being decoupled from on-premises infrastructure and developed and/or transitioned to the cloud. This supports cloud computing and heterogenous and separate systems that can be acessed online and via APIs. The growing number of applications and systems and their presence on the cloud raises new requirements and opens new possibilities concerning integration. It is becoming common practice to depend on different distributed systems with an increasing number of partners. Direct integration calls are an option for smaller systems with a decreased number of integration systems, however not appropriate for more complex systems and numerous systems. The use of Messaging Integration has been seen as a solution for those integrations scenarios and for the past decade, the use of a separate Messaging Oriented Middleware System has also gained popularity to separate the integration layer, where integrations are located, from the different systems, and with this option, new services have emerged to meet the demand.
In this report it is described an implementation of a generic integration system that allows users to integrate heterogenous cloud systems with the use of APIs without need for coding. The implemented solution has the purpose of tackling hardships that come with manually coded integrations and associated costs.

Resumo

Hoje em dia, cada vez mais, as aplicações e sistemas estão a ser separados de infraestruturas on-premise e transitadas e desenvolvidas para a nuvem. Este movimento suporta cloud computing e sistemas separados e heterogéneos que podem ser acedidos online e via APIs. A crescente presença de aplicações e sistemas na nuvem levanta novos requerimentos e abre portas a novas possibilidades relacionadas com integração. Está a tornar prática comum depender de diferentes sistemas distribuídos com um número crescente de parceiros. Integração através de chamadas diretas são uma opção para pequenos sistemas com um baixo número de sistemas a ser integrados, no entanto não é apropriado para sistemas mais complexos com números elevados de sistemas a integrar. A utilização de Integração por mensagem tem sido vista como uma solução para esses cenários de integração, e durante a década passada, a utilização de um sistema Middeware Orientado a Mensagens tem ganhado popularidade de modo a separar a camada de integração, onde as diferentes integrações estão localizadas, dos diferentes sistemas, e com esta opção, novos serviços tem emergido para acomodar a demanda. Neste documento está descrito a implementação de um sistema de integração genérico que permite aos utilizadores integrar sistemas heterogéneos na cloud através da utilização de APIs sem necessidade de codificação. A solução implementada tem o objetivo de reduzir as dificuldades que surgem com integrações codificadas manualmente e os seus custos associados.

Keywords

broker, brokering, message, messaging, message-oriented middleware, distributed systems, heterogeneous, heterogeneous systems

Acknowledgments

Contents

Acronyms

API – Application Programming Interface
B2B – Business to Business
CLR – Common Language Runtime
CTS – Common Type Specification
ERP – Enterprise Resource Planning
IL – Intermediary Language
JSON – JavaScript Object Notation
MOM – Messaging Oriented Middleware
XML – Extensible Markup Language
XSD – XML Schema Definition
XSLT – Extensible Stylesheet Language Transformations

List of Figures

Table List

Chapter 1
Introduction

This report presents an overview of messaging concepts along with a few technologies that support the messaging paradigm and a few solutions created by competitors.

It starts with a brief contextualization on what integration is, and what caused and brings the need for integration in the first place. Followed the contextualization, the document refers to a few solutions developed to meet the requirements raised by this need to integrate systems and their context, this will be explained with the use of a few virtual examples. It will then explain why those systems and solutions are no longer appropriate in certain and specific scenarios followed by explaining what messaging and message brokers are, and how they can fulfill several aspects and needs that previous solutions cannot.

A few developed message broker solutions are then studied and analyzed along with a few technologies they use or are appropriate for message brokers and a few of their functionalities, followed by comparisons of the different technologies and systems. The requirements are then stated to retrieve the different functionalities needed for the system to be developed, and the implementation is then reported.

1.1 Motivation

With the introduction of the concept of globalization in the 20th century and the widespread and expansion of it in the 21st century, enterprises have been met with several opportunities to expand their business and have been looking to innovate ever since to stay above water when it comes to competition and move their business abroad. This includes expanding premises, opening new hubs, partnering with several partners and use of external services. These processes include the need for integration.

Integration is becoming more complex due to the number of partners and extent of enterprises, and therefore, harder to accomplish, develop and maintain when done manually case by case through programming and direct integration.

This report proposes a system that works as the base of an integration middleware system that allows its users to create different integrations for their needs and scenarios. The system characteristics and features will be based on the evaluation of the different currently existing solutions. It will be a system that meets the integration requirements of B2B, cloud systems with the use of APIs.

1.2 The Problem

The project is being developed as a dissertation proposal by MAEIL, a company that provides integration services to select partners called Orchestra, to support different enterprises to integrate their data with one another.

Integrations are however made manually through coding and require direct action and work from MAEIL. Clients need to contact the company, describe their integration needs, and the different data types and formats are then specified for an integration to be manually developed on a personalized way. Currently the different connectors and business logics are created to order and in this current state it requires a great amount of human intervention for this process.

So instead of providing integration services this way, the system proposed by this report presents an integration system that can be later used as Software as A Service. This system is a piece of software that includes several different generic integration elements that can be used together by the customer to define and setup integration sequences. It will be designed in a way that it can be created not only by developers but by customer who have no expertise in programming as the element-based system should be as codeless as possible. A codeless element-based system makes it easier to create integrations, allowing customers to create integrations of their own and accessible to a greater number of enterprises, be it smaller or bigger.

The element-based design allows the application to be incremented in terms of functionalities without affecting existing ones.

1.3 Project Team

This project comprises of the following members:

Vasco Pereira – Supervisor at DEI

Hugo Duarte Fonseca – Supervisor at MAEIL

Giovanni Oliveira - Student

1.4 Development Methodology

For the development methodology to be used for the development of this piece of work, an iterative approach does seem more appropriate. The modularity and different functionalities that the Message Broker has makes iteration a beneficial option as the different parts and functionalities can be added and tested over time one by one as required. This methodology also allows an easier control over possible risks and adjust as needed.

This iterative approach consists of initial requirements and architecture design, followed by incremental development of the system parts.

Firstly, the functional and non-functional requirements are taken to understand what the needs and restrictions are for the design of the system.

A second step consists of the design of the system architecture and the different components to meet the necessary requirements.

After the system design has been completed, the application is developed. The system is developed incrementally, meaning small functional parts of the system are developed and tested at a time, and later incremented to the final product.

1.5 Planning

In this section, it is documented the planning process and the expected division of tasks.

First Semester

This project has had it start on the 20th of September when I have been introduced to the current way integrations are being done. Integrations I have documented.

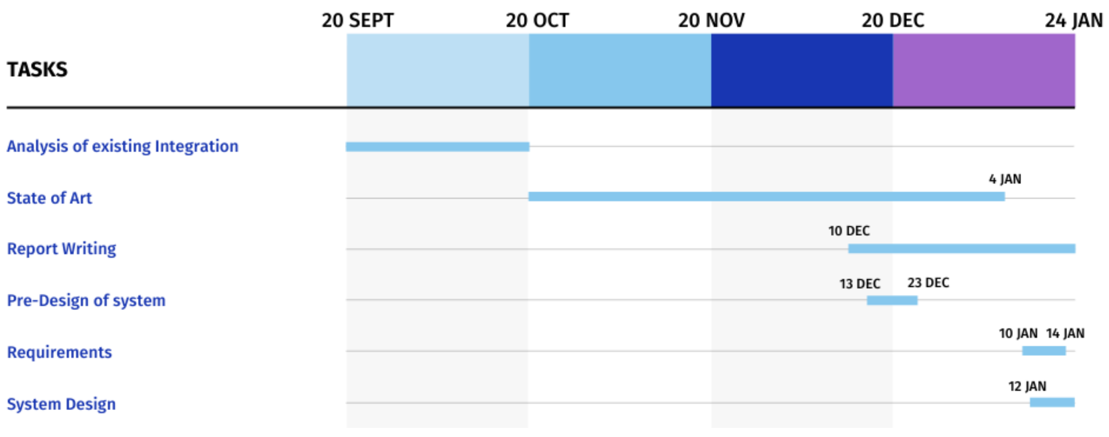


Figure 1- First Semester Gantt

The first four weeks, I have dedicated my time to analysing how some integration services are currently done in terms of its architecture, technologies used, and packages and frameworks integrated to support the solutions. This researching about these different tools for two projects and the documentation of them.

Following these tasks, I have started the research related to the State of Art. While at first, I was having difficulties orienting myself, after a conversation with the supervisor I went back on track. This phase included research of different existing solutions for generic integration applications and their possible architecture in order learn possible ways to implement mine and a few technologies along with it.

While I was researching about the current solutions under State of Art, I was taking notes separately and writing bullet points, I then decided that would be better and more beneficial if I started writing into the report directly as I was researching and learning and therefore, the Report Writing task overlaps the State of Art and drags along until the end of the first Semester.

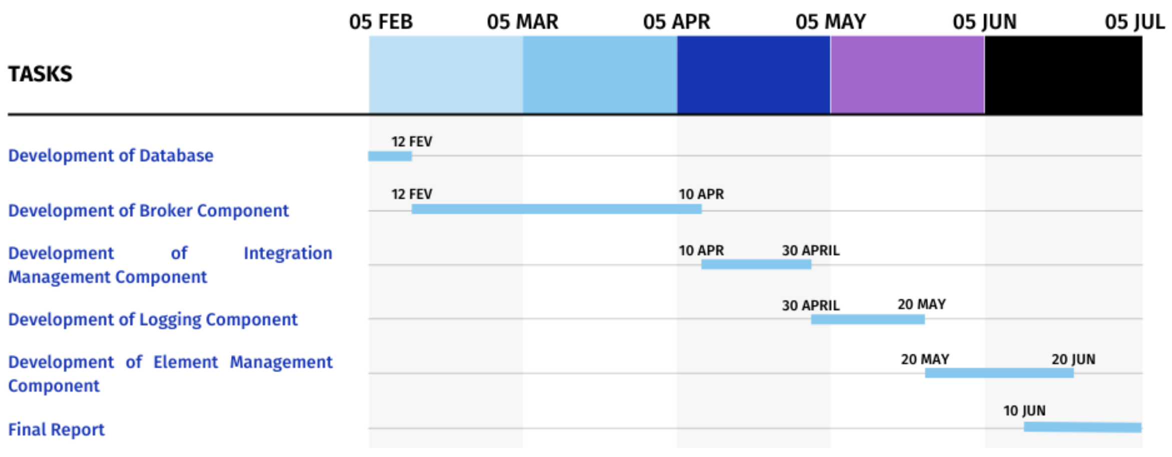
After some research has been done, I started having in mind a few possibilities and structure design of a possible system, and therefore I have made a few component diagrams to share ideas with the supervisor, as possibly taking the functional and non-functional requirements, to adapt to a final possible design. This corresponds to the Pre-Design of System task.

It was only possible to extract the requirements after the New Year, and so this task has been done close to the end of the Semester to be able to design a possible final system.

Finally, the last task done was related to the design itself as specific above.

Second Semester

For the second semester, the planning is set to cover the development of the System using the chosen software development methodology. The planning of the Second Semester has been formulated at the end of the First Semester and is used as guidance for workload division. It can be re-adjusted and reformulated as needed based on the divergence of expected outcomes.



The first week has been assigned for the development of the different components to be stored in the database and creation of the database itself.

From the 12th of February to the 10th of April, it is expected the development of the base of the Broker component which would include the creation of the system responsible to accommodate different elements, and the initialization of integrations made of them.

Followed would be the development of an Integration Management Component that is made of a REST API responsible for initiating and stopping already created integrations. This task should take around 20 days.

After this task, a Logging Component is planned to be developed from 30th April to 20th May, which is responsible for the storage of integration logs.

The last Development task is designed to be an Element Management Component, which is also a REST API responsible for managing the elements from a user. This should take roughly 1 month.

The final Task is the writing of the Final Report, which is set to start around 10th June and continue until delivery.

1.6 Risk Analysis

A few prominent risks have been acknowledged and evaluated along with ways of dealing with them.

The risks are mainly focused on the lack of experience with certain technologies and frameworks, including the use of services, such as cloud databases and the integration of those elements in the project.

Condition	Consequence	Probability		Impact		Risk		Action	How?
Lack of experience with programming language	Hardship and development delay	Medium High	3	Serious	2	Serious	5	Mitigate	Spend extra time in research regarding the language
Difficulty integrating technologies, framework and services	Hardship and development delay	Medium High	3	Critical	3	Critical	6	Mitigate	Spend extra time in research regarding the technology, framework or services
	Impossibility of development using the technology	Low	1	Catastrophic	4	Serious	5	Avoid	Fully invest in the resolution of the condition by focusing solely on the integration of the required services
Difficulty integrating with services already in use	Development delay	Low	1	Critical	3	Serious	4	Mitigate	Research and request help on information about integration to existing systems/services.

1.7 Objectives

The objective of this internship is the study of different integration techniques and different integration concepts that support those same techniques to understand why integration is important and a modern requirement in certain settings. It also has the purpose of exploring different message broker solutions developed and currently available, to better understand their differences and how a software service can be developed to meet integration requirement needs of enterprises including functionalities and structure. A few technologies will also be analysed to understand what technology is more appropriate.

The final objective is, however, the development of a functional Messaging Broker system that can be deployed into the cloud and later implemented as a Software as a Service. The system should provide generic functionalities that allow incremental features for both users and developers. When it comes to developers, the system should have a modular approach in a way that new features/elements can be added without directly impacting previous features, and each feature work as an addiction to existing others, having similar interactions with the system by using common interfaces. From a user point of view, the system should support a modular approach to integration, allowing users to easily integrate cloud APIs with the use of the different generic elements the application has available.

1.8 Results

No results at the moment.

Chapter 2
Background Review

Before researching and entering a more in-depth analysis about different Messaging Brokers, their functionalities, and what they allow users to do, it is important to understand what they are, why do they exist, and what problems could have raised the necessity to implement these solutions. This chapter exposes a brief explanation of the development of enterprise information systems, and how certain problems started arising with the growth of companies and the subsequent incrementation of enterprise systems. It then follows with the concept of a Broker and Messaging Broker and how it is becoming a requirement for certain branches.

2.1. Evolution of Integration Requirements

In this section it will be discussed and exemplified how the continuous development and expansion of enterprises have brought new challenges in each stage and what solutions have been developed to meet the everchanging integration requirements.

Early System Requirements

Despite how good a solution is, it may not be a good fit for all enterprises, as not all enterprises have the same problems. In the early days, in the USA for instance, companies were small enough that very few businesses needed the services of a full-time administrator [1], the scale of business were so small, that it was easy to administrate manually. When a customer, for instance, purchased a good or made a reservation, it would be registered locally on paper, and that would suffice as the flow of requests were small enough and the overall customers were local and manageable. After WW2, the economic ‘Boom’ accelerated the flow of business and customers, and the completely manual operations started to become inefficient, and the aid of enterprise systems was needed. Small local systems were used, usually, a simple application that would meet a specific department’s needs, helping manage operations performed and improving performance, bringing a smaller rate of manual error, and increased economic benefits.

Enterprise Growth & Integration Requirement

With the economic and customer flow increase, companies grew as a consequence. This growth forced companies to expand, be it locally within the company, or externally, by opening new hubs elsewhere. This introduced a set of problems that needed to be resolved.

Due to complexities, enterprises needed to separate inner responsibilities across different departments, such as accounting, IT, sales, inventory, among others. And so, when, for instance, a sale was completed, by the sales team, the purchase information needed to be reported to both the inventory and accounting departments. This created a dependency between departments systems which could often cause problems such as outdated data between departments, incoherent data due to human error, inefficiency, scalability issue, communication complexity, amongst others, and brought the need for integration systems such as ERP systems (Enterprise Resource Planning).

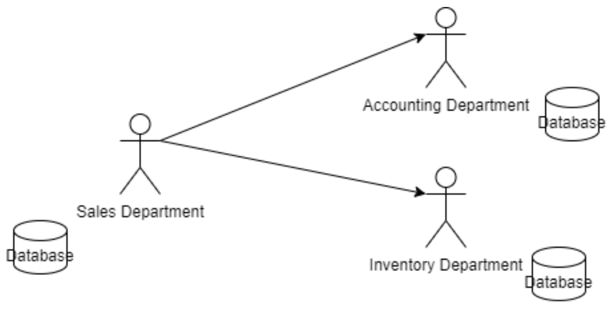


Figure 2- Coupled Manual Data Sharing

Enterprise Resource Planning

ERP is the name given to a system or software used by an organization to facilitate the management of daily activities of the company through centralization. [2]

It is, fundamentally, a centralized system with a single database and data structure that provides integration for the different departments within a company, allowing the access of updated data between all departments, reducing human error and increasing the level of automation.

So, for instance, if the Sales Department wanted to add a Sale into the system, it would use common tables amongst all departments in the shared Database and insert the sale. The accounting department would then have access to updated, recent data, and would be able to perform operations on that date and save the results in the same database, since it knows what data format and structure to expect.

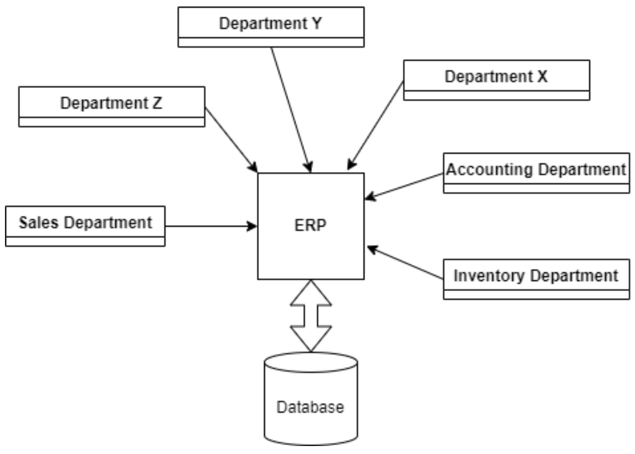


Figure 3- Centralized Data Sharing

Current and Future Enterprise Integration Requirements

In a modern and recent setting, with the introduction of globalization and redefinition of the meaning itself, different, yet similar needs are surging. The ever-growing amount of data, department size, and its systems' operations complexity require a greater level of decoupling for the sake of management. The introduction of new technologies for the different needs of the different departments pushes these departments to opt for heterogeneous systems using different technologies, which makes coupling and the use of a single Database unfavourable, and so the need to integrate these systems becomes a high priority. This is a case example of heterogeneity within a company. A possible solution would be messaging integration.

Messaging

Messaging is the act of communicating through the exchange of messages. It allows asynchronous, fast multi-system communication, working on the basis of message sharing to transfer data. At first glance it may seem like an HTTP request, but with the difference there is no need to wait for the whole process to take place and wait for an answer, simply sending the Message (data) itself and 'forgetting' about it (send and forget). There are possible analogies that help understand better what Messaging is. Take an example of a phone call. When a person wants to communicate with another, they would grab the phone and call the desired person, and the conversation could only take place if the other person was present and near the phone to answer and talk, and the conversation would only be over when both parties ended the conversation subject. This is an example of a synchronous communication. Now, for instance, consider that another person also wants to communicate with their friend, however this time, they choose e-mail as the mean of communication. They can simply write an e-mail on their computer or mobile phone, and send it, without waiting for a specific reply for the message, and whenever the friend has time and is ready to read the e-mail, they can do so. This is an example of asynchronous communication, and this is what messaging in based on.

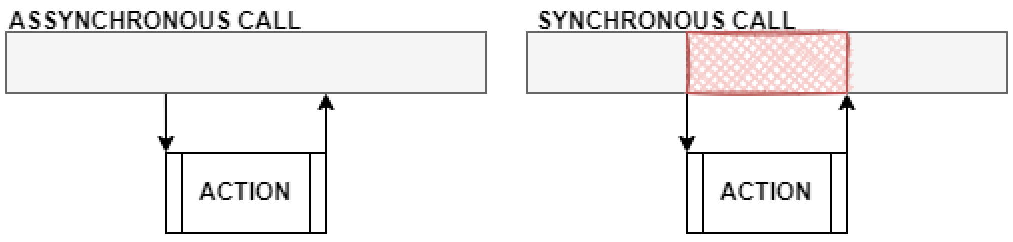


Figure 4- Synchronous and Asynchronous calls

The messages shared are basically some data structures like JSON or XML [28] that are technology independent, therefore allowing heterogenous systems to communicate and share data with each other. The nature of using messaging to communicate between heterogenous systems brings a few limitations, however.

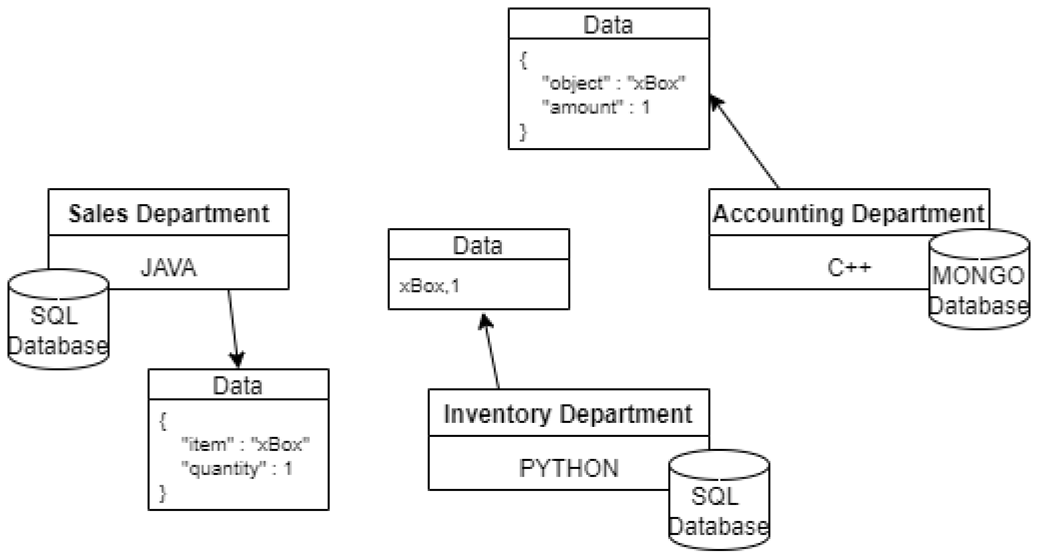


Figure 5- Different Systems with different Data Types

As visible in Figure 5, the Sales Department has a system running on Java and using a SQL database, and when it comes to data type it expects to receive and send, the data type is JSON, and the format is Item and Quantity.

On the Accounting Department however, they are using C++ along with a MONGO DB and expects Json as well, however with a different format of Object, Amount.

Finally, the Inventory Department has its systems developed in Python and a SQL Database and expects to receive and send CSV formatted data.

For data to be shared between these departments it would be necessary the creation of one endpoint in each system, for each action, (such as sale, renting, etc…) that would convert and translate the different message data types and structures. This may be appropriate and easier for smaller distribution of systems but bring certain problems such as code duplication and complexity and difficulty of integration when new features are added within a bigger distribution of systems.

Similar cases can happen externally such as a company expanding overseas, for instance, may have difficulties sharing the same database. The same applies to a case of integration with external entities to the enterprise. When a company receives an order, it may need to communicate with several different systems, to process that order, that may not belong to the organization itself, such as a delivery company or other partners. In this case, depending on the number of partners, the complexity and number of endpoints needed would be extremely high.

One possible solution is the use of Message Brokers.

2.2. Message Broker

Message Brokers are middleware, also referred as MOM (message-oriented middleware), that facilitate the connection of several different systems together. It is a software that allows application, systems, and services to share information with each other by validating, translating, and transforming messages between messaging protocols, and filtering, selecting, and routing with the support of its routing services and business rules functionality[3][4]. This allows heterogeneous systems and services to communicate with one another even in the event of being written in different languages and needing not to know anything about each other. It also allows messages to be shared between more than one system. For instance, if a Sales Department registers a sale, a message can be sent to the Broker, and then sent by the Broker to all Departments whose information is appreciated.

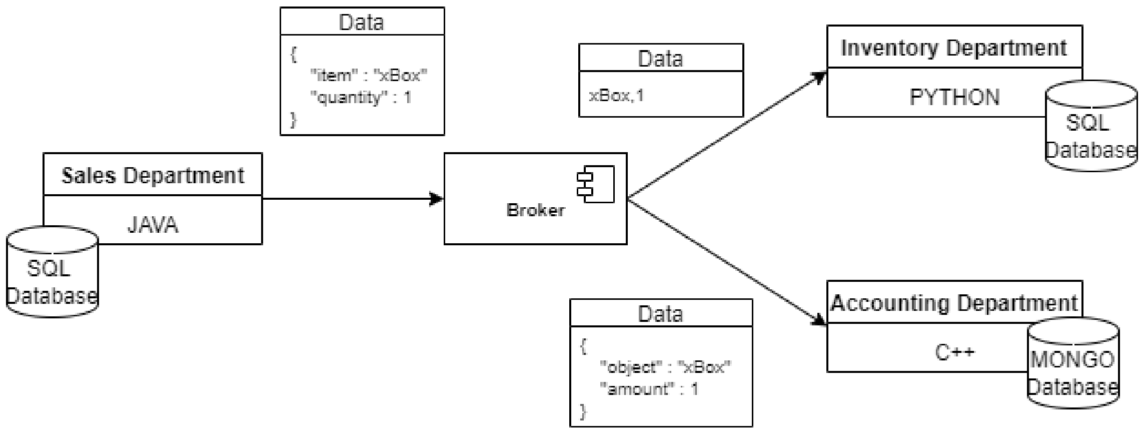


Figure 6- Visual Broker Example

Following the last example, and as it is visible in Figure 6 the Sales department sends the sales information as json, and the Broker then, for each department previously set as destination, translate, and transforms that data type into the required data type of the destination department based on pre-set schemas for the effect.

Validation, Transformation and translation

One of the essencial functionalities of a Message Broker are validation, Translation, Transformation and routing of messages. This is the essence of a broker. Firstly, upon receiving a message, the broker needs to verify and guarantee that the message is in fact, contructed with the expected elements and in the correct format. This is to filter wrongly formulated messages and avoid future data corruption. This can be done with schemas such as XSD (XML Schema Definition) and JSON Schema. Then translate and transform the data into another format using, for instance, XSLT. (Extensible Stylesheet Language Transformations) Following, then, to the routing of the transformed message to outter destination systems.

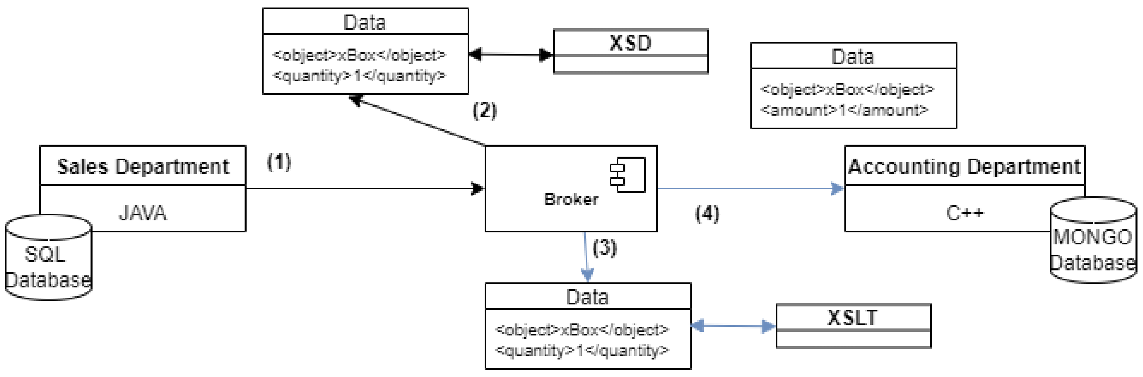


Figure 7- Validation and Transformation

Message Patterns

Message Brokers often use queues and topics in order to distribute and route messages to their destination and allow asynchronous communications. There are two basic distribution patterns.

Publish/Subscribe Messaging: As referred before, one advantage of Message Brokers is the fact that it removes the need for different systems to call several systems and one by one by their endpoint to send the same message to all of them. The Broker can receive a single message and send that same message to all necessary destinations, by possibly, using a topic. A Topic is a message container that receives messages, from one or more publishers, these can be clients or for instance, interfaces, and then the same message published into the topic can be read by one or more consumers, which would be the destinations. This can be referred as the publish/subscribe paradigm.

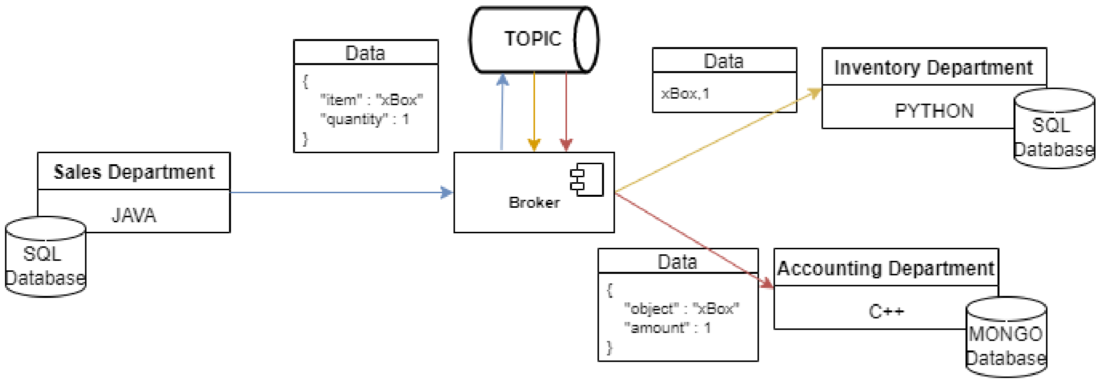


Figure 8- Example of Topic

Point-to-Point Messaging: Point-to-Point Messaging usually uses Queues. Messages can be published into queues by one or more publishers but can only be read once by a single subscriber. It works in a similar way a REST API would work, in that it has a one-to-one relationship, the message sender, that sends a message and the receiver, that reads/consumes that message. It has the benefit, however, of allowing asynchronous messaging, that being, if the receiving end is unavailable, messages can be read later, and the sender does not need to be waiting for a response.

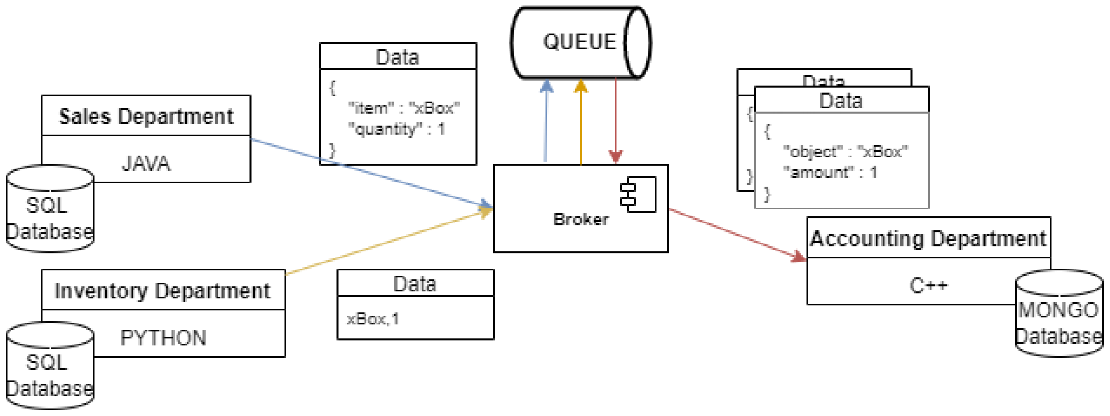


Figure 9- Example of Queue

Business Processes

Message Brokers usually do have a business process component. This component is responsible for containing the business logic to be applied to the incoming messages.

Business logic is the segment of the Message Broker that applies logic to messages, this can be used for filtering, routing, and monitoring/notifying. For instance, in a scenario of business logic applied to routing, when receiving a message from the Sales department, it may check for certain characteristics of the message, this could be message type, content, attribute, among others, and based on that characteristic, can either route it to the Accounting Department only or route it to the Account Department and Inventory Department instead. This could also be used to validate purchases if and only if there is inventory for the items in question. Routing is part of the Orchestration, which controls the flow of the messages.

2.3. Technologies

For the development of a system, the usage of different technologies is unavoidable. The following are technologies that will be used in this project as they are required.

C#

C sharp is an object-oriented programming language developed by Microsoft in 2000 with the goals of being a simple, general purpose language for developing software components while providing support for software principles . C# is run on the .NET Framework and was introduced alongside .NET Framework and Visual Studio. C# programs are run on .NET. [30]

.NET

.NET is a Virtual execution system, named common language runtime (CLR), and a collection of class libraries developed by Microsoft with the objective of code interoperability. The .NET Framework compiles code written in the C# language into an intermediary language (IL). That code and resources are then stored in an assembly either with an extension of .exe or, usually, an extension .dll. When a C# Program is run, the CLR loads the assemblies and perform compilations to convert the intermediary language code into machine instructions [30]. Because the compiler conforms to the Common Type Specification (CTS) and uses intermediary language, code compiled with .NET can interact with each other, even if written in different programming languages.

Chapter 3 State of Art

In this chapter there will be an exposition of different existing brokers in the area of data integration and different competitors' related work and offers. A few solutions will be shown and explained to be later compared and analyzed to understand the differences between them and how the different characteristics can be applied to the solution to be developed.

3.1. Microsoft BizTalk

Microsoft Biztalk is a middleware that facilitates the connection of several different systems together. Modern organizations are usually modularized, or for instance, divided into departments, such as the Finance Department, the Sales Department, Data department, amongst others. When, say, a sale is accomplished via the company's website, it may need to communicate with all these departments, and traditionally, in this case, the website would have to call all these endpoints to send the purchase request. This can bring complexity and data integrity problems. There is also the problem of technology use. The different departments may use a wide range of incompatible technologies.

The BizTalk Server follows the publish/subscribe architecture, and therefore a message is published into the system and received by one or multiple subscribers. Biztalk implements the 'content-based publish/subscribe' model. [10] Content-based publish/subscribe model refers to a model where subscribers state what messages they want to receive based on a set of criteria about the message. When it comes to BizTalk, the message is assessed the moment it is published and all subscribers that are looking for messages with specific details met in the receiving message receive the message.

Biztalk uses XML as the inner message language, to functions in terms of validation and transformation to referencing data.

BizTalk is comprised, from a simplistic view, of Receive Ports, a Message Box, Orchestration and Send Ports.

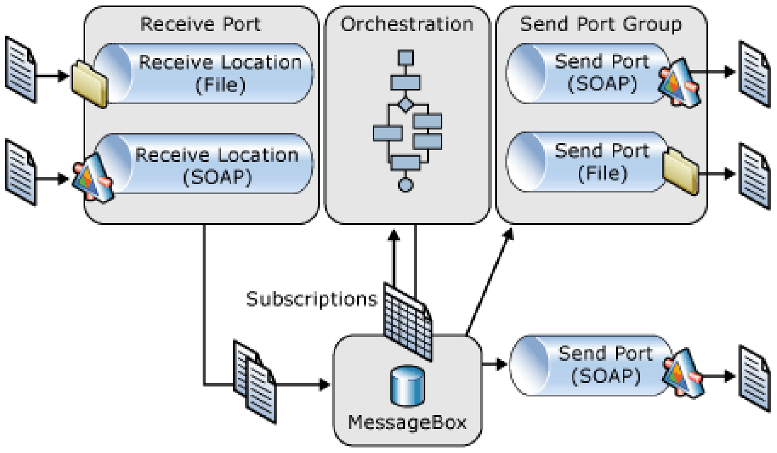


Figure 10- Biztalk Simplified View [12]

Internal Message Access

In this section, it will be discussed how Microsoft Biztalk Server shares data amongst internal elements.

BizTalk Message

Messages in BizTalk Server are structured in a multi-part format. The moment a message is received by Biztalk to the moment they go out, they are processed as a class named Biztalk Message. They are made of a Context and zero or more parts. Messages with several parts have one of those parts set as the body part. Each parts comprise of data, be it a XML document, flat file or other binary stream of data. The body part is used to identify the type of message for purposes of routing. [12]

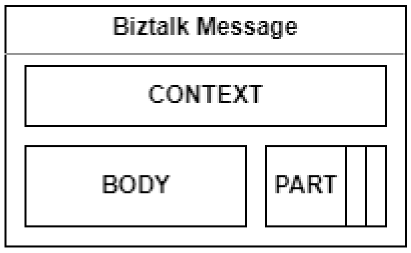
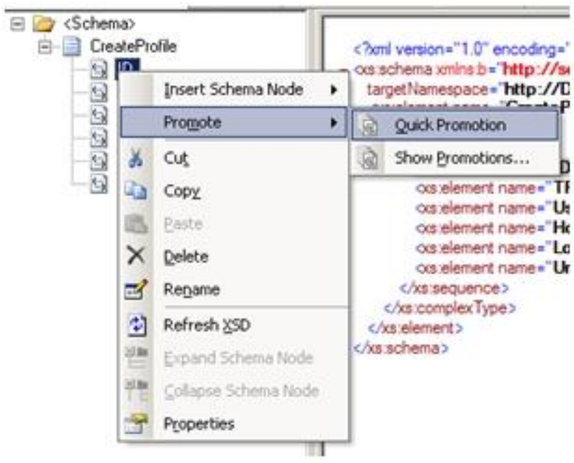


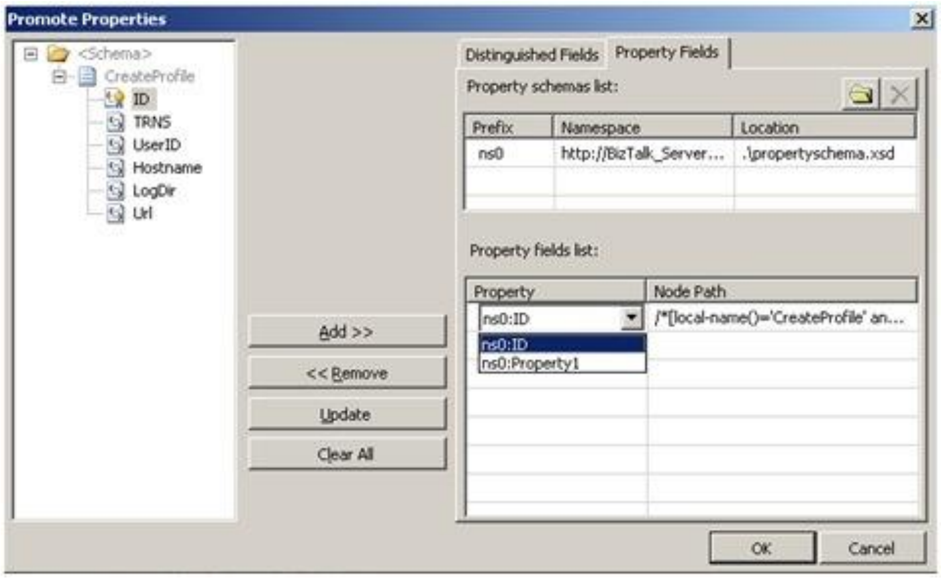
Figure 11- Biztalk Message

Message Context hold a set of properties about the message, values can be drawn out from the message or generated by BizTalk. To draw properties from the message, Biztalk uses a concept of ‘property promotion’. Promoted properties are held in a property schema of the schema, in a shared location, so that it can be referenced by others schemas. Property schema has the name and type of each element that is defined as promoted.



```
<?xml version="1.0" encoding="
<xs:schema xmlns:b="http://s
  targetNamespace="http://D
    <xs:element name="TF
    <xs:element name="Us
    <xs:element name="Ho
    <xs:element name="Lo
    <xs:element name="Ur
  </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 12- BizTalk Promote Functionality [29}



Prefix	Namespace	Location
ns0	http://BizTalk_Server...	.\propertyschema.xsd

Property	Node Path
ns0:ID	/*[local-name()='CreateProfile' an...

Figure 13- Promote Property Fields [29]

When it comes to values generated by Biztalk into the Context, it can be added, for instance, by the adapters themselves. They put properties into the Context about certain message details like the location the message was received and the type of adapter used to receive the message.

Biztalk Architecture

In this section the different components of the Message Broker are broken apart and analysed in terms of functionalities and structure.

Endpoints - Ports

Ports are the endpoints of BizTalk. Is through ports that messages are received and sent to outter systems. Ports consist of adapters, pipelines and optionally a Mapper.

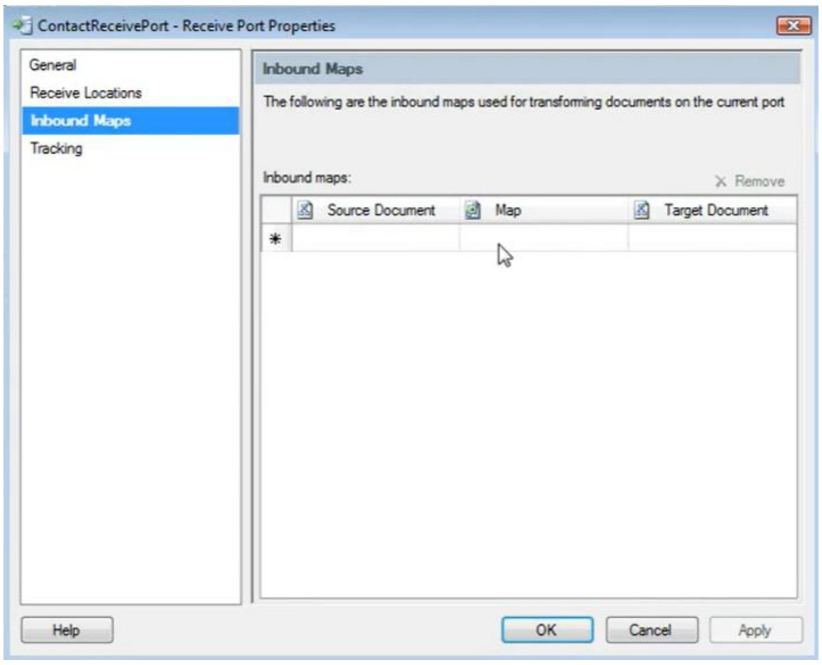
With all the components, the Port is able to receive data, validate data, encrypt or decrypt data if needed and convert the data into a BizTalk Message.

Adapter: The adapter is responsible for extracting, receiving and sending the data from and to a transport.
Pipelines: Pipeline is responsible for inducing the message received from the Adapter into a sequence of stages, as there are reasons to prepare and transform the messages. It is in the pipeline that operations like Decoding/Enconding, Dissassemble/Assemble and validation happen.
There can be more than one component associated with each stage. It is here that message types such as Flat Files are converted into XML, via a parser, so some operations in Biztalk can occur, such as validation.
Mapper: After the above processes, Biztalk can Map the XML data into a different XML schema, using XSLT, that may be used internally for that type of data. Biztalk has a tool named Biztalk Mapper which facilitates the creation of XSLT files. In case there is a complex XML translation, such as a Contact with multiple different address types, it has a 'functoid' visual functionality to facilitate the process.

There are two types of Ports: Receive Ports and Send Ports.

Receive Ports

Receive Ports are collections of one or more ***receive locations*** and a possible InboundMapper.


ContactReceivePort - Receive Port Properties

Inbound Maps

The following are the inbound maps used for transforming documents on the current port

Inbound maps:

Source Document	Map	Target Document
*		

Figure 14- Receive Port

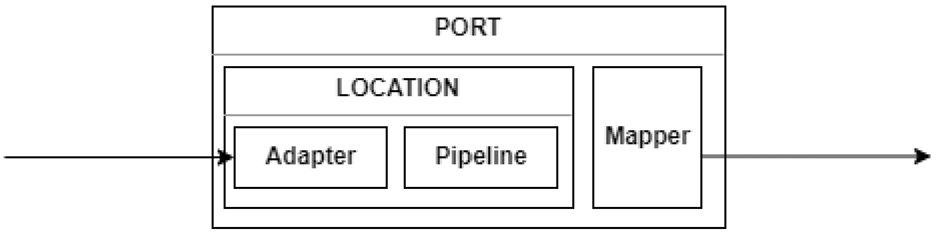


Figure 15- Receive Port

Receive Locations contains and are made of a Receive Adapter and a Receive Pipeline and is the configuration needed by Ports to make use of these two elements, representing the configuration of a single endpoint to receive messages.

In case of a Receive Port, the **receive adapter** listens or looks for data wherever it is configured to look at, and upon receiving new data, it converts the data received into a Biztalk Message and places a set of data attributes into the Context of the Biztalk Message.

The **receive pipeline** receives the messages from the adapter and make them go through the possible stages:

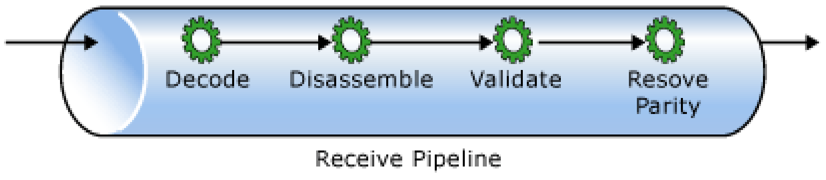


Figure 16- Receive Pipeline

Send Ports & Send Port Groups

Send Port Groups are multiple Send Ports and behave similarly to a distribution list. Once a message is sent to a Port Group, the message will be redirected to all Ports linked to that Port Group.

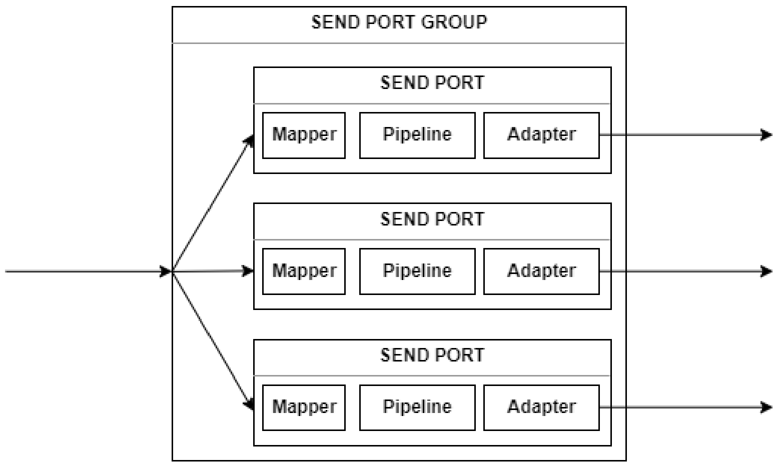


Figure 17- Send Port Group

Send Ports are a composition of a send pipeline and a send adapter.

Both send ports and send port groups have three different states: Bound- Send port and port groups are bound to an orchestration. Started- It means they exist and are active, when in this state BizTalk will deliver messages to the port or port group. Stopped- Port or group Port are not running. When in this state, new messages are sent to a queue of the host where a send handler is running.

Because Biztalk separates the messaging layer from the business process layer, a port or port group need to be bound to at least an orchestration to receive messages, therefore needing to be Bound to be started.

In case of a Send Port, the **send adapter** receives the Biztalk Message and adapts the message to be sent to the destination defined.

In case of a Send Port, the **send pipeline** receives the message from Biztalk and makes it go through different stages, so that it can, for instance, convert the XML into a different data type such as flat text. The different stages are as follows:

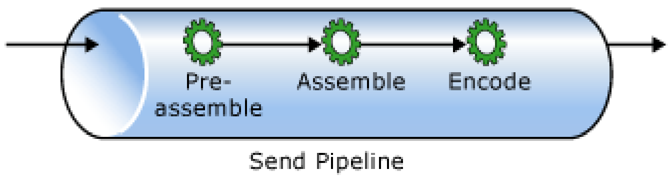


Figure 18- Send Pipeline

Data Storage/Sharing - MessageBox

The MessageBox is the base of the publish/subscribe engine in Biztalk.
The MessageBox is made of two components:

- One or multiple Microsoft SQL Server databases
- Messaging Agent

The SQL databases allows persistence for several things such as messages, message properties, subscriptions, orchestration state, message parts and so on.
The Message Agent is the component that encapsulates and abstracts the database component and works as the interface used by BizTalk to communicate with the MessageBox [13]. It supplies interfaces for actions like subscribing to messages and message retrieval, so that, for instance, orchestrations, can receive the messages.

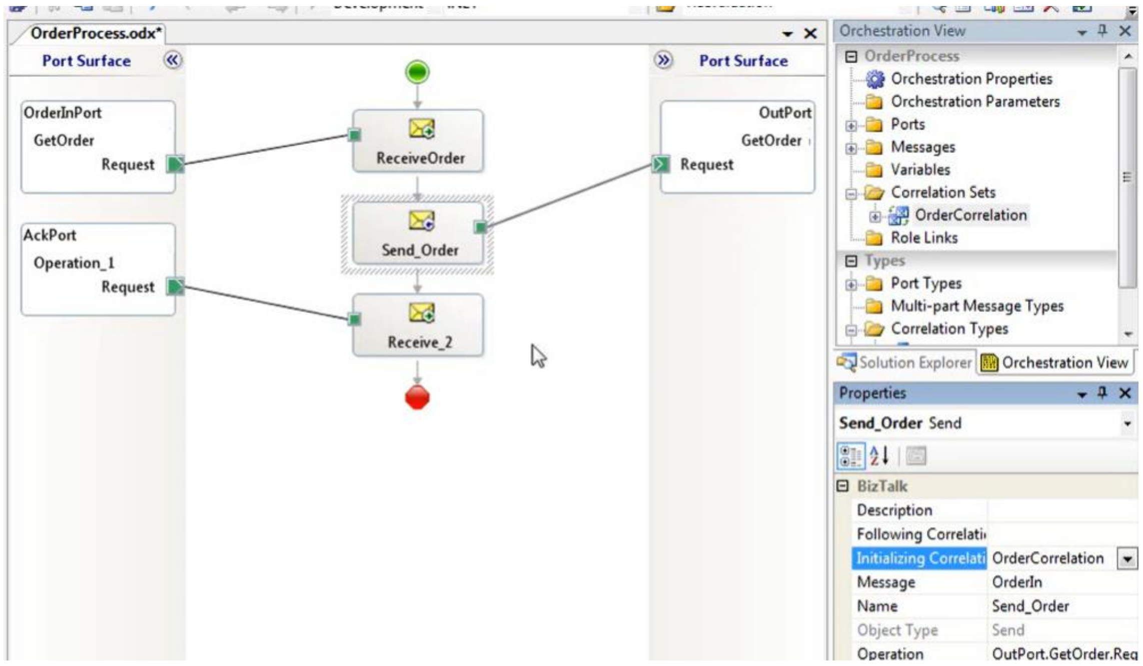
As previously stated, messages inserted into the database come as Biztalk Messages. Biztalk Messages have a Context and is through that Context's promoted and predicated properties that business processes verifies if it subscribes to that message. If it does, the MessageBox sends the message to the business process, which then, after processing, if the business process is bound to a send port, it returns a message to the MessageBox.

Business Process/Rules - Orchestration

Orchestration is a mechanism for defining how messages are exchanged in a business process. It provides functionalities such as semantics for modelling exceptions, synchronization, parallel processing, and conditions.

Orchestration can be activated by subscribing to messages published to the MB OR explicit call from another orchestration. They are often long time running (waiting for an answer) and its process state must be saved in the database. Uses Types such as port types and Correlation Types to support correlation (through context variable). Correlation Type and Correlation Sets are used to provide correlation between a message sent asynchronously and an acknowledgment received later by a system related to that request.

Correlation type is a collection of context properties, that are also used for routing purposes. The properties are selected from XSD schemas created.
Correlation Sets are sets of properties that hold actual data and are instances of the Correlation type they point to. This is held in an XML format.



Send_Order Send

BizTalk

Description	
Following Correlati	
Initializing Correlati	OrderCorrelation
Message	OrderIn
Name	Send_Order
Object Type	Send
Operation	OutPort.GetOrder.Req

In the example above an order is received at the first step, and then, at the second step it is sent via a send Port. In the Send Shape, it is possible to define an Initializing Correlation, for when the objective is to set the values of the Correlation Set based on the Context attributes of the current message at the Shape.
At the third shape, a receive shape, the ‘Following Correlation’ property is set to be the same Set as the previous Initializing one. This basically defines a subscription of the next message received that has all and the same values currently at the Correlation Set. This allows correlation on characteristics like IDs.

Orchestration has a flow that can be created by the user. It is here that business rules are implemented, whiles, if/else and decision making.

Logging

No Logging information has been found for Microsoft’s Biztalk Server.

SWOT Analysis

Strengths	Weaknesses	Opportunities	Threats
-Provides a lot of functionalities for developers -Good for Message widespread as it is fully based on subscriptions -Good Variety of Technology Adapters	-Complex compared to alternative -Is on-premises only -Complex structure for simpler integrations -Expensive Solution	-Ease of access to local structure	-Unfriendly Interface -Forced technologies like XML -Lack of media documentation -Only on Windows

3.2. Boomi AtomSphere

Boomi Atomsphere is like Microsoft's Biztalk Server in terms of purpose and in what it allows to be done. It has, however, a different architecture and different characteristics to its competitor. Boomi works as an integration Platform as a Service (iPaaS) and allows customers to integrate diverse composition of cloud and on-premises applications with no software or coding. This is one advantage point in that AtomSphere is 100 percent cloud native, allowing users to integrate application at any time and any place, being able to choose between downloading the integration solution created on cloud to run on-premises or using Boomi's cloud platform to run the integration Solutions.

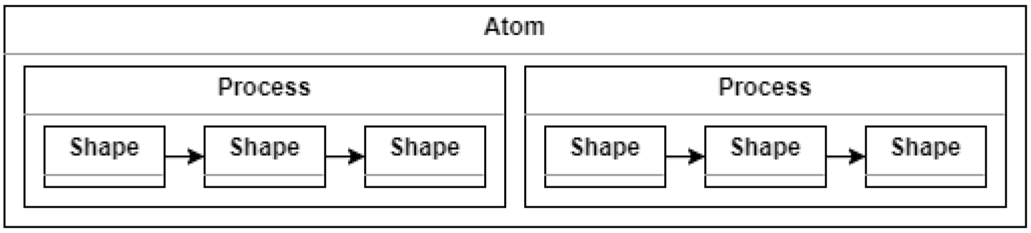


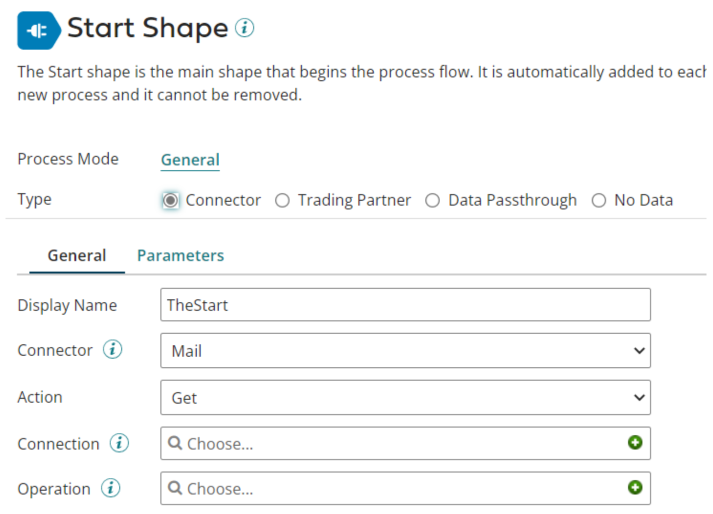
Figure 19- Boomi Atom

Boomi's Atom

Dell's Boomi has a patented lightweight, dynamic and self-contained runtime engine called Atom. The Atom contains one or more Integration processes that are initiated when the Atom is. They run Virtually on any Server (supported o windows and Linux operating systems and require Java and Java Runtime to function).

Boomi Processes

Processes is the name given to a sequence of integration actions. They contain A sequence of **Shapes** and components that are connected to one another, to create an integration flow. Processes have a start and end point and start when their start trigger (Start Shape) is activated. Processes can hold variables that are accessible to all Shapes. The act of running a Process is called 'Execution'.


Start Shape

The Start shape is the main shape that begins the process flow. It is automatically added to eacl
new process and it cannot be removed.

Process Mode	General
Type	[✓ Connector] Trading Partner Data Passthrough No Data

General Parameters

Display Name	TheStart
Connector	Mail
Action	Get
Connection	Choose...
Operation	Choose...

Figure 20- Start Shape

The Start shape can be of 4 Types: A connector, Trading Partner Connector, Data Passthrough and No Data. This means a Process can start with data from personalized Connector or a pre-defined connector, from a process call or start with no data, from a timer trigger.

Internal Message Access

In this section, it will be discussed how Dell's Boomi Atomsphere shares data amongst internal elements.

Documents

Documents are data that goes through a Process. Documents can be single records, groups of them, an EDI Transaction or an entire file. Documents, however, are normally individual files read in. A point to be taken into consideration is the fact that some files may represent more than a record such as a csv file with several rows, therefore, if the User needs to process them individually, it is needed to use the Split Documents function from the Data Process Shape

Document Flow

Boomi does not have a specific internal data type. Documents flow from Shape to Shape from the start of the process to the end of it as a way of sharing data between Shapes.

Boomi supports five raw document types: XML, JSON, Flat File, EDI and Database, and allows the creation of Profiles for each of these. Documents are presented in 4 different formats:

- Records: for Flat Files and Database Types.

- Transactions: for XML and EDI.
- File Instances: does not require structure analysis (email analysis/exporting to disk).
- Empty: for simple triggering of subsequent shapes within a process.

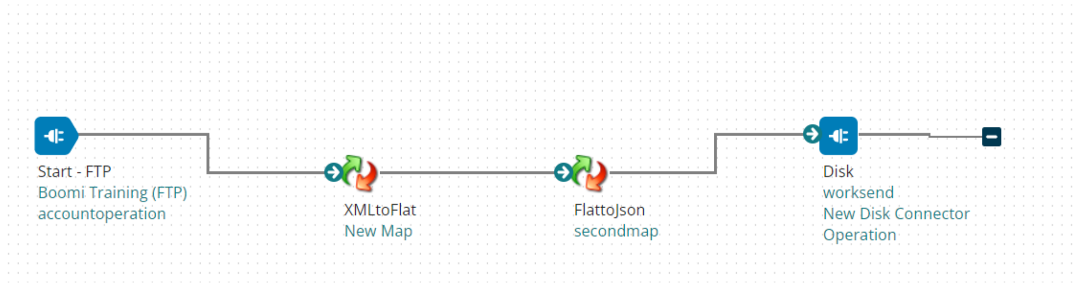


Figure 21- Process Example

Document Properties

In Boomi it is possible to set Document Properties. Document Properties are similar to Microsoft's Biztalk Server's Message Context, possibilitating metadata associated with each Document. There are two types of Document Properties: Document Property and Dynamic document Properties.

Standard Document Property contains run-time specific data such as connector type.

Dynamic document Properties are properties that can be used or created to be used temporarily by Shapes.

Boomi Architecture

Boomi Shapes

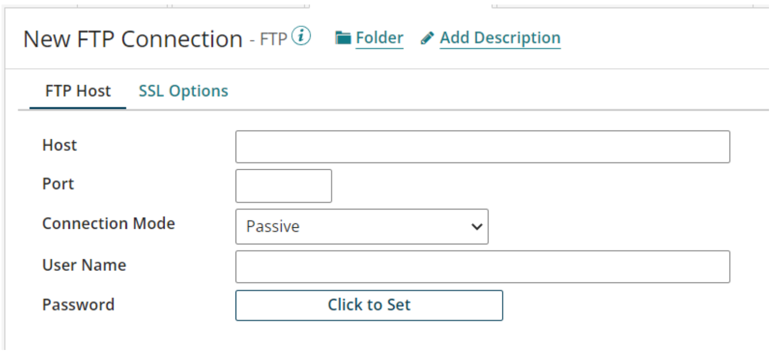
Shapes are a bundle of pre-defined available tools with their specific code that perform specific tasks. They are separated into three categories:

- Connector Shapes: Connectors are used to get data into the process, and send data out of the process.
- Execute Shapes: Execute shapes are used to manipulate and transform data.
- Logic Shapes: Logic shapes are used to control the flow of the process. This includes Business Rules. (such as if conditions)

Boomi Shapes use Components to configure their behaviour. Shapes may have some configuration attached to themselves but more detailed configurations such as an URL to access are made via specific components for the Shape. Components in a real setting could be either a file or data stored in database columns.

Boomi Components

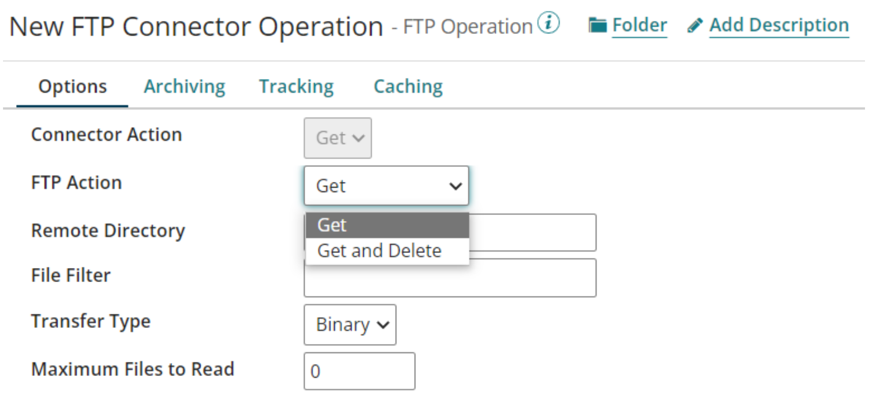
Components are reusable configuration objects, this includes apis, certificate connections, connector operations, cross reference tables amongst others. As per the nature of reusability, it can be created once and referenced by multiple Shapes. For instance, the components Connector Connection and Connector Operation can both used to configure one or more Connectors. The following are examples of a connection and an operation component:



New FTP Connection - FTP

Host	
Port	
Connection Mode	Passive
User Name	
Password	Click to Set

Figure 22- FTP Connection



New FTP Connector Operation - FTP Operation

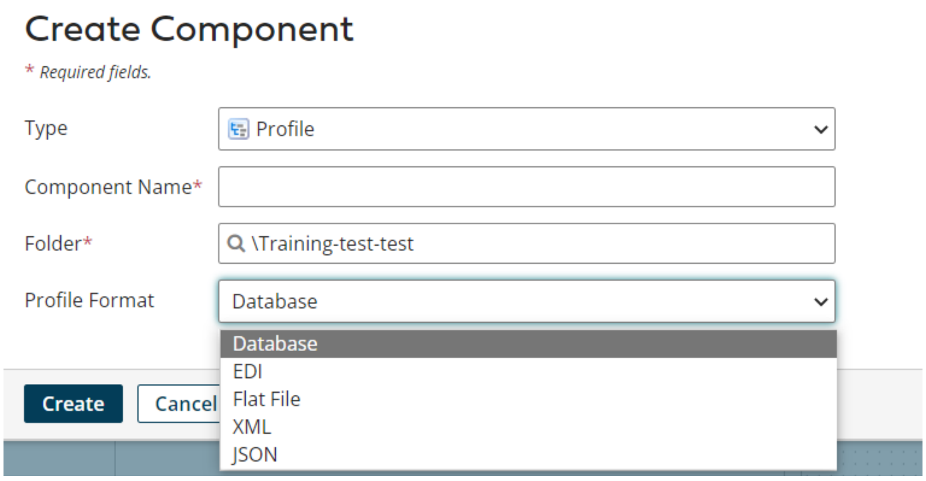
Connector Action	Get
FTP Action	Get
Remote Directory	
File Filter	
Transfer Type	Binary
Maximum Files to Read	0

Figure 23- FTP Connector Operation

Profile

Profiles detail the layout and format of different documents read into and out of AtomSphere. Profiles do not fall into the Shape category, as they are not usable elements in the process, but a component, therefore a configuration some shapes use. It works similarly as an XSD would, in the sense that it defines a group of data elements, that a document can have, and with that, allow for data transformation and validation. While Microsoft's BizTalk receives the messages and has a parser to convert the different data formats into XML in the pipeline, and then validate it against an XSD, in Boomi, messages are parsed using Mappers, which use Profiles to extract data, and if validation is needed, Boomi has a logic Shaped named 'Cleanse' that also use a Profile to check for mandatory data.

Boomi allows 5 data formats to have profiles: Database, EDI, Flat File, XML and JSON.

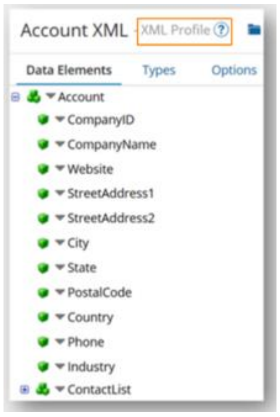


Create Component

* Required fields.

Type	Profile
Component Name*	
Folder*	\Training-test-test
Profile Format	Database

Figure 24- Profile Component Formats



Account XML

- Account
  - CompanyID
  - CompanyName
  - Website
  - StreetAddress1
  - StreetAddress2
  - City
  - State
  - PostalCode
  - Country
  - Phone
  - Industry
  - ContactList

Figure 25- Profile structure

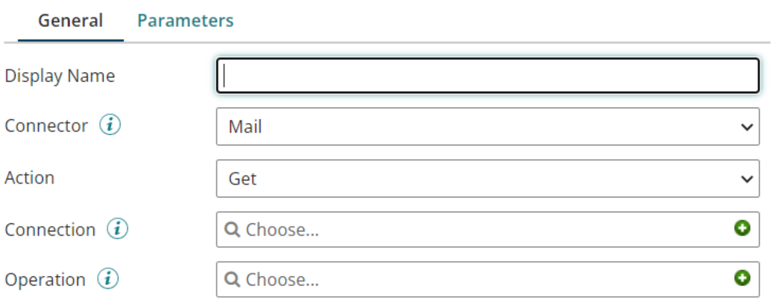
Different Shapes

Endpoints - Connector Shape

The Connector shape is the main element containing all the information and operations to connect to a data source by using referenced Components to setup a connection. There are available out-of-the-box technology Connectors ready to setup and use. The technology connectors available to use can be found at [15].

Connector Shapes contain two components: A connection and an Operation.

- Connection: Data referring to the physical connection information. (the 'Where') Such as URL and Port.
- Operation: Data referring to what function to call, or file(s) to act upon. (the 'How') This includes setting up sub-directories to look for, operation such as (GET/ GET and DELETE) file filter, transfer type and maximum files to read.



Display Name	
Connector	Mail
Action	Get
Connection	Choose...
Operation	Choose...

Figure 26- Connector Configuration

Business Process - Execute Shapes

Execute Shapes are Shapes that support data creation, transformation and retrieval and document properties modifications. There are 12 different execute Shapes that provide different functionalities:

Table 1- Boomi Execute Shapes

MAP	Used to transform data from one format to another, with the use of a Map component and two Profiles (source and destination).
SET PROPERTIES	Used after of before a connector. Sets connector-specific properties (like file name or metadata, like promoted elements in Biztalk) for documents
MESSAGE	Generates text messages, from static and dynamic content.
NOTIFY	The Notify Shape allows you to create custom execution logs. It does not create a notification per document but aggregate all notification texts into a single message. Can be sent to an email alert event. Does not modify received documents (inbound data)
PROGRAM COMMAND	The Shape allows the user to call system commands, SQL statements while using a Database Connection component or a stored procedure. A stored procedure is a user-defined function which includes a set of SQL and procedural statements (including assignments, declarations, loops amongst others) [19]
PROCESS CALL	Process Call Shape allows the user to reuse processes for repetitive tasks/processes like initialization or logging. Sub-processes are called once for every document. Documents can be transferred to sub-processes.
PROCESS ROUTE	
DATA PROCESS	Data Process Shape grants several options for manipulating document data it allows to: • Decode (BASE64)

	• Encode (BASE64) • Character Decode • Character Encode • Combine documents into one • Custom Scripting • Search and Replace using regular expressions • Split Documents • Map Json to MIME • Map MIME to Json • Encrypt (PGP) • Decrypt (PGP) • XSLT Transformation • ZIP • Unzip
FIND CHANGES	Allows the Process to Track changes made to a document
ADD TO CACHE	Adds Documents to a Document Cache, using a Document Chache component, allowing to store documents to be later on retrieved by a Retrieve from cache
RETRIEVE FROM CACHE	Retrieves Documents from a specified Document Cache Component, allowing to select those with specified Keys
REMOVE FROM CACHE	Remove documents from Cache

Business Rules - Logic Shapes

There are also Logic Shapes. Boomi's Logic Shapes are responsible for controlling the flow of the Process. Some can function as Business Logic. The Shapes and their functionalities are as follows:

Table 2- Logic Shapes

BRANCH	The Branch Shape allows the user to receive a group of documents or document and execute the paths in sequential order. Allows the User to virtually set more than one Output for a Shape.
ROUTE	Route send Documents into different paths based on a value.
CLEANSE	The Cleanse Shape works as a validation tool, validating, repairing or rejecting documents based on a profile.
DECISION	Decision Shape allows the user to route Documents based on a comparison between two values.
EXCEPTION	Exception Shapes allow the user to conclude the integration flow and create a custom error message. It is possible to simply interrupt the current document processing instead of the whole process.
STOP	Similar to exception but without creating and logging an error message.
RETURN DOCUMENTS	Shape that returns the documents to a parent process.
FLOW CONTROL	Allow to define if documents are processed in batches, Shape by Shape, or if documents go through the whole process before the next received document start processing. Defines how documents are processed.
BUSINESS RULES	Business Rules Shape contains a set of business rules that are validated in sequence. If all Rules come back as True, the Documents go down the Accepted path, else go down the Rejected Path. Created Map Functions can be used.
TRY/CATCH	The Try-Catch Shape sends Documents down a 'Try' path and in case a document fails to be processed, it will be added to the group of Documents to go down the Catch path.

Data Storage/Sharing – Not Specified

Boomi does not specify what kind of database it uses, but despite the lack of information on the matter, it is likely it uses an SQL database to store different settings as it is the most fitting for the ocasion.

When it comes to Data Sharing, as mentioned beforehand, the data is shared directly from Shape to Shape as a Process is a sequence of Shapes. It can also be shared using queues and publish/subscribe topics with the use of the AtomQueue Connector.

Logging: Process Reporting

Boomi Process Reporting stores all the executions' logs on a .log file [33]. This specifies what actions have been taken in each Shape as informative strings. But there is no way to return back and verify what data has been passed from Shape to Shape. The only Document Data stored that can be review is the Messages/Data received and sent from Connectors.

SWOT Analysis

Strengths	Weaknesses	Opportunities	Threats
-Simpler Structure -Linear Processes -Bigger control over data, data flow and path -Good Variety of Technology Adapters	-Not the best for Message broadcast as it is linear -Logging is based on simple strings -No data Standard within process	-Cheaper Option -Friendly Interface -Can run on both cloud and locally with patented software -Run on most servers: windows and Linux	-Bad customer Service -Can broadcast messages using topics but not built for it

3.3. Workato

Workato is an enterprise automation integration platform, that is versatile in a way it allows users to address all core use-cases across businesses. It was founded in 2013 by Gauthan Viswanathan, Harish Shetty, Dimitris Kogias and Vijay Tella [20]. Built on a foundation if iPaaS, it can automate end-to-end workflows, deploy bots as well as utilize artificial intelligence and machine learning.

Workato provides functionality to synchronize data and integrations between different cloud and on-premisses applications and endpoints, to have workflow automation with the use of business processes across multiple applications and business rules.

Workato also focus efforts on the ease of use, with the goal of having the same ease-of-use for both 'citizens' and developer/IT integrators by having a component-based interface with minimal settings and programming [19].

Recipes

Recipes work similarly in a way Boomi's Processes work. They are automated event-based workflows that can be built by users, they are a container for other elements and are composed of a trigger and one or more actions. When run, recipes will run in the background waiting for the trigger event to trigger and when stopped recipes stop looking for triggering events. When a trigger receives events, a Job is created per event (message) received, which consists of an integration flow for a specific event. Recipes have an ID, and can be reused and shared publicly.

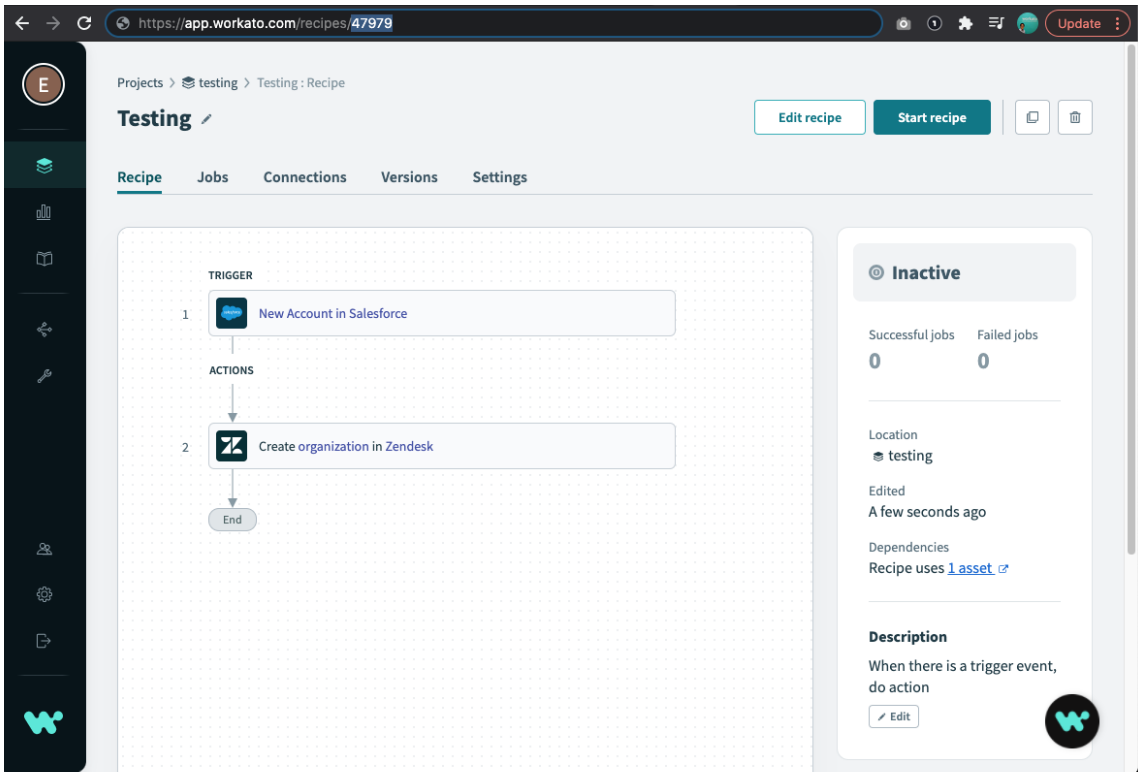


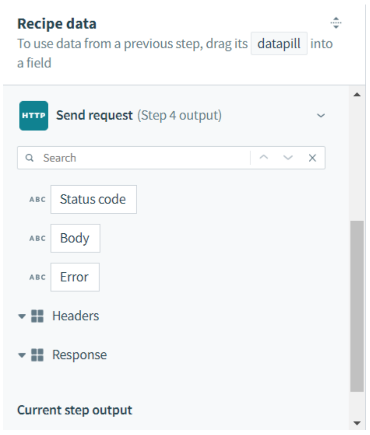
Figure 27- Workato Recipe example

Internal Message Access

In this section, it will be discussed how Workato shares data amongst internal elements.

Recipe Data and Datapills

Similar to how Biztalk works with XML to perform data operations inside the broker, Workato also has an internal message format called Datapills. Datapills are JSON-like key-pair values that are able to hold data and are generated by the different Actions and trigger outputs, these are stored in the Recipe Data, at Job level, meaning, every time an event is received by a trigger, it starts a job with that event that will flow over the business integration process, and for every data producing or receiving action in the process, a datapill is created by the action and will be stored in the Job's recipe Data to be used and referenced by later actions on that Job.



Recipe data

To use data from a previous step, drag its datapill into a field

Send request (Step 4 output)

Search

Status code

Body

Error

Headers

Response

Current step output

Figure 28- Recipe Data showing Datapills from step 4

Datapills are identified by steps, meaning the different Datapills are separated within the Recipe Data by the step they were produced in. Due to the structure on how data is pushed and accessed between actions, actions can only reference data from previous steps. Mapping can be done with both constant data and/or from variables. What this means is, that values for variables such as connector settings can be drawn from either both variables from a data pill and/or static inserted values. When it comes to Data transformations, it falls into the assignment and formulas category.

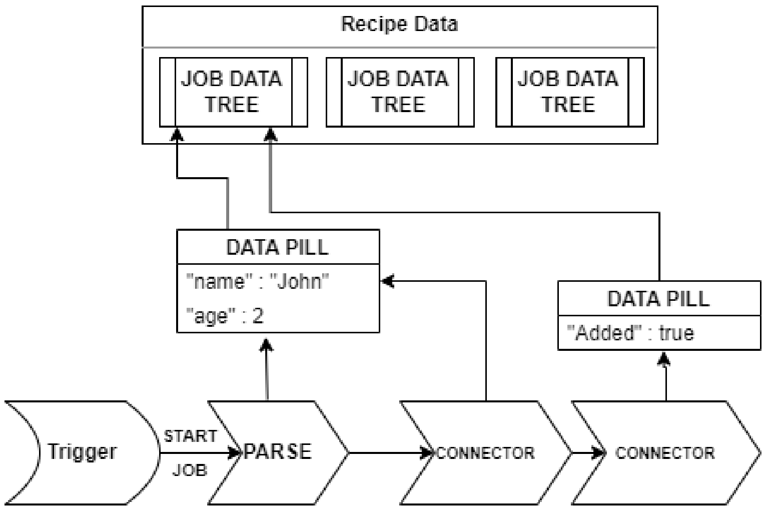


Figure 29- Data Access and Storage

Datapills can be produced, as mentioned above, by actions, but can also be produced by the User. Workato has an Action named Variables By Workato, and what it does is allow the user to create a DataPill. It differs from other Datapill in the sense that the variables defined in Variables are mutable. So the data can not only be acessed but can also be modified. Datapills support variable of the following types:

- String
- Float
- Date
- Boolean
- Integer
- Object
- Date time
- Array/List

Workato Architecture

In the Following Segments will be shown Workato's elements and functionalities.

Triggers

A trigger is the first element of the Workato recipe and is responsible for scanning or listening to an event to trigger and execute the following actions described within the recipe workflow. Triggers can be activated by app triggers, customized APIs, or schedules. Depending on the available API, they can be categorized into different types depending on when they check for new events and how the events are handled (single event or event batch)

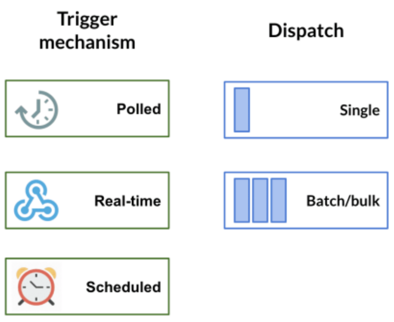


Figure 30- Trigger Mechanism & Dispatch types [21]

When Recipes receive events (data), it queues them in sequence to process them in order, therefore, an event goes along the Recipe workflow, and only once the event completes the processing, the next event starts processing. The process maintains a cursor position at the Trigger event it is processing, saving its position, meaning, if the Recipe changes its state from a running state to a stopped state, whenever the recipe changes back to a running state it can return processing all the remaining trigger events in the list.

Polling triggers

Polling Triggers check events periodically as configured and can be as low as every 5 minutes depending on the Workato Plan. It saves the last time it has fetched so it only fetches from that date onwards.

Real-Time Triggers

Real-Time Triggers usually require registration in the application to connect. They work on the basis that the application will send a notification or message to the broker, and therefore, the broker does not need to be constantly polling the application. Real-Time Triggers at Workato are usually Webhooks supported by regular polling, with generally, lower polling frequency.

Scheduled Triggers

Scheduled Triggers are enacted on specific days and times. Be it hourly, daily, monthly, amongst others. They also fetch all events matching specified criteria, in other words, the trigger can fetch repeated events on two different calls.

Single Triggers

Single Triggers retrieve one event at a time. This is used, for instance, when a sale happens, and the sales company sends a message or calls an API to inform other endpoints of the sale. It is beneficial for real-time synchronization and is usually asynchronous.

Batch Triggers

Batch Triggers retrieve a list of events at a time. This is used, for instance, when there is a need to move sets of data from one endpoint to another, hence why it is used in scheduled and Polling triggers.

Trigger Filters

Trigger Filters are a set of conditions that are used to filter what events are meant to be processed and continue down the workflow. Triggers still fetch all the events that meet the temporal criteria from the source, however, events are filtered afterward before jobs are created, and therefore no log information is created. The fact filtering can happen right at the entry, Trigger, is a positive in terms of performance, as no further processing is needed.

Conditions

Conditions consist of 3 parts in Workato, data, condition, and value. Several conditions can be coupled together with the use of AND or OR operators. Conditions are used in Trigger Filters as seen before and in condition actions. There are 14 conditions the user can choose from [22]:

- Contains
- Starts with
- Ends with
- Does not contain
- Does not start with
- Does not end with
- Equals
- Does not equal
- Greater than
- Less than
- Is true
- Is not true
- Is present
- Is not present

Business Process - Actions

Possible Actions fall into two categories: Action in an app and control flow statements.

Action in an app includes application calls.

Control Flow Statements are part of the business logic. There are 6 control flow statements:

- IF condition
- IF/ELSE condition
- Repeat action
- Call recipe
- Stop Job
- Handle Errors

IF Conditions

If conditions allow to Shape the flow of the Recipe by verifying a condition or a set of conditions using either one or more AND operators or one or more OR operators. If the result is true, the Job will flow the extra steps defined by the if condition, if not, it will ignore those steps.

IF/ELSE Conditions

If/Else Conditions also Shape the Flow of the Recipe by verifying a condition or set of conditions, however, the flow can only be directed to one of the paths instead of both.

Repeat Action

There is a Looping functionality within the possible actions. Loops allow a portion of Actions to iterate of a List variable, a variable that can be extracted from a Datapill from a previous Step or created empty simply to iterate a specific number of times.

Call Recipe

Call Recipe allows a Recipe to call another callable Recipe whose Trigger is a Callable Recipe.

Stop Job

The Stop Job Step ends the currently running Job.

Handle Errors

Handle Error Step Monitors several Steps and actions for Errors, in case an error occurs, it can either retry up to 3 times, depending on what the User previously setup, or can skip the job flow to the On Error path. It is like the Try/Catch concept.

Endpoints - Connectors

Connectors are responsible for connecting to external applications to either request, receive, send or modify data. The different Connectors have their own conections types. A complete list of Connectors can be seen at [28]. There are pre-set App Connectors for several different apps with functionalities set for those applications specifically, and technology connectors such as FTP, HTTP and SFTP.

Connections

To Communicate with external Apps Wokato uses Connections. Connections are a set of configurations, pre-defined for each tipe of connector, that must be included in the connector to connect to an external endpoint. Connections are re-usable and can be used in multiple Connectors at a time.

Data Storage/Sharing – Not Specified

Workato, similarly to Boomi does not specify what kind of database it uses. But since Workato store a json representation of the output data of each Job's Actions, SQL may be an option.

When it comes to Data Sharing, as previously stated, the data can be referenced from previously data generating Actions. It can also be shared using publish/subscribe topics using specific connectors.

Logging: Jobs Reporting

When a trigger event retrieves messages, each message will be executed by the recipe flow, therefore generating a Job per message. All Jobs contain information about themselves such as JobId, date and description, and are stored in a database and the metadata stored about each Job can be seen on [24].

When a message flows from Shape to Shape in a Job, Shapes produce datapills, which are the outputs of the processing done by the Shape, and those outputs are stored in a database, and reference the Job they belong to. As it is possible to see on Figure 29, during each Job, Actions produce outputs, their Datapills, and those outputs are stored in a Job Data Tree which is then stored somewhere, possibly in a SQL Database, referencing the Recipe Data.

With these elements, Workato Job Reporting provides the functionality of retrieving the data produced by each Job from a data source and reconstruct the output of each Shape to visualize the Jobs success and progress. This is Workato's system for data tracking and integration error analysis.

Workato Tools

Workato has also a set of tools that can be created to be used on multiple Recipes. These are linked to the User's account. These Tools include Common data Models, Lookup Tables, Message Templates, Pub/Sub and Properties.

Common Data Models allow the user to create Data Models to use as reference for Actions like Mapping. Lookup Tables work similarly to cross-reference tables in a way that they allow the user to look for frequently used data and extract a row of data based on a match against one or more columns.

Lookup Tables are organized like a databased and can be imported from a CSV file or created manually, and are a good feature as it remves the need to query from an external database or perform more expensive operations.

Message Templates allow the user to create dyamic message templaces based on variable values.

Pub/Sub allows the user to create topics that have a schema, to be user, later on, by publisher Actions. Publisher actions publish the schema variables into the topic, so that Subscribers, which need to be Recipe Triggers, can listen to and trigger the Recipe they are associated with.

Properties allow the user to set a list of key-values that can be referenced by any Action from any recipe via the data tree, just like any other data pill data. These property values are looked up each time a job is executed.

SWOT Analysis

Strengths	Weaknesses	Opportunities	Threats
-Simpler Structure -Linear Processes	-Not the best for Message broadcast as it is linear	-Friendly Interface	-Expensive Option -Can broadcast messages using

-Bigger control over data, data flow and path -Good Variety of Technology Adapters		-Can run on both cloud and locally with Agents -Agents can run on all major OS	topics but not built for it

3.4. Technologies

This project has a set of technologies that have been defined as a requirement, which were stated previously. This topic, related to the Technologies under State of Art, is related to a set of different technologies considered to be used in the project, that may perform or serve similar purposed and may be compared with one another to reach a decision on which one is best.

XML (Extensible Markup Language)

Extensive Markup Language is a simple and flexible text format derived from SGML [5]. XML defines a set of rules for encoding documents in a format that is both human and machine readable. Its usage and specifications are open-source and the main objective of its creation was to facilitate, and simplify the sharing of data over the internet. It uses Tags and Elements to define data. It is extensively used among enterprise.

XSD (XML Schema Definition)

XML Schema Definition is a document with the objective to formally provide descriptions of the elements of an Extensive Markup Language (XML). It is used to verify if the content of a XML document is according to a specific format and follows a set of rules, in order to classify the document as valid o invalid.

XSLT (Extensible Stylesheet Language Transformations)

Extensive Styleshheet Language Transformations is a language meant for transforming XML documents into other XML documents with a different structure or/and model, through the definition of rules. The transformation can also occur from XML to other formats such as plain text and HTML. It makes use of Xpath to query and extract data from input’s nodes.

XPATH

Xpath is a language used to specify parts of a XML document. It is based on node querying through paths. (ex. /bookstore/book) and brings a set of expressions and predefined functions. Usually used along XSLT to support data tree transformation.

JSON

JSON stands for Javascript Object Notation and is a lightweight data format. JSON was developed on early 2000s with the objective to solve a problem of transferring Javascript data through and HTML document, that are both easy to read and write by Humans, and easy to parse and generate by machines. [24][25][26]. It has since gained worldwide interest by companies and creators due to its potential for stateless integration and data exchange between systems to the point of replacing XML. In October 2013, Ecma issued the first edition of its JSON Standard ECMA-404.

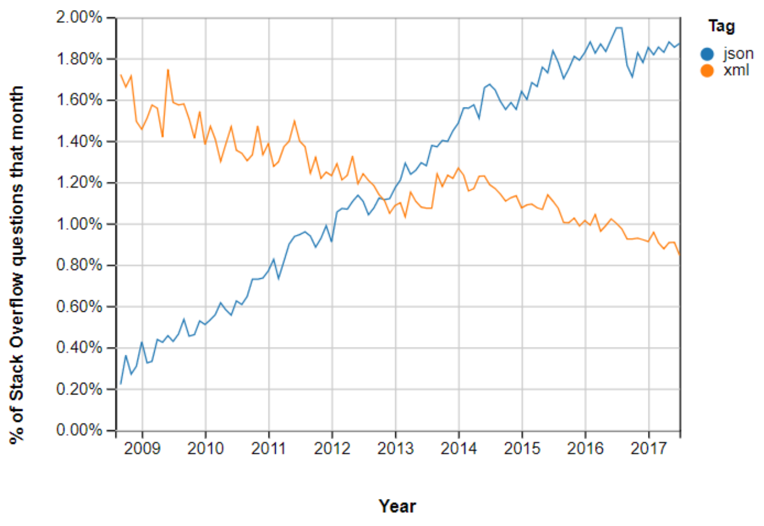


Figure 31- Stack OverFlow Trends - json vs xml

JSON Schema

JSON Schema is the equivalent of XSD for JSON. It allows to describe JSON data formats, using an easily readable JSON file, allowing JSON data validation. This is useful to ensure that data instances are following a certain structure with the correct data types, in order to avoid poor data quality and errors during processing.

Service Bus/Message Broker Implementation

Implementations of message Broker, or service buses, support the messaging patterns explained beforehand, point-to-point and publish/subscribe, and are used to share messages between systems asynchronously and a few examples are RabbitMQ, Amazon SQS and Azure Service Bus. Despite having their difference, they share most concepts [31].

Service buses do have a queue concept, which is based on a first-in first-out basis, are optimized to read from it. This fits under the previously noted point-to-point connection, under Background Analysis, in which there can be multiple publishers and a subscriber. Multiple subscribers can also be set up to compete for the messages.

Data is temporary under these implementations which means, they are stored for a specific, limited time or until read, without any purpose of long-term storage.

Service buses also have a publish/subscribe concept, that may be implemented in different ways, consisting of a temporary set of messages published by one or more published, and read by one or more subscribers. Subscribers do not compete against each other to read messages from the same topic, each one holding a copy.

Service Buses do support routing.

Event Streaming Platform - Kafka

Kafka is a scalable publish-subscribe messaging system design surrounding a distributed log. It is focused on a high throughput where data is stored in log files [32].

Kafka logs store all the data received by appending it and have its data stored permanently, allowing for multiple reads

Kafka does not have a concept of queue or point-to-point messaging and relies solely on a publish-subscribe paradigm which is named topics. Topics are made of zero or more partitions which are setup when the topics are created. Partitions are sets of data received by the topic and are meant to allow parallelism as each consumer reads from a partition, however, a partition can only be read by one consumer by each consumer group at a time. Each time partitions are modified, or new subscribers are introduced, Kafka readjusts which subscriber consumes from which partition on a round-robin basis to readjust load within consumer groups.

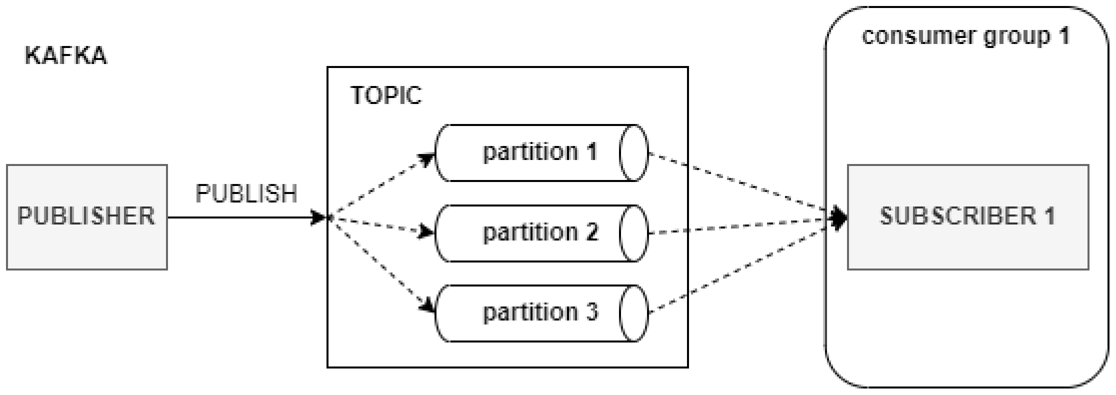


Figure 32- Kafka publish/subscribe with one subscriber

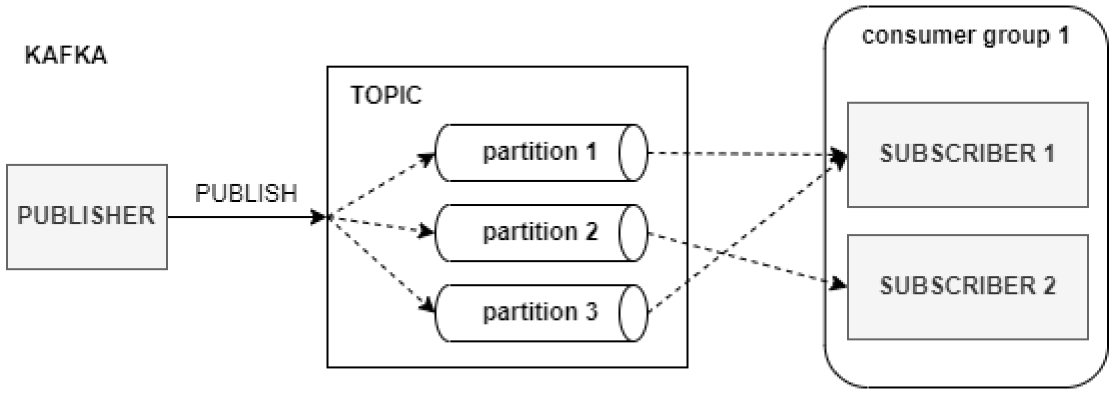


Figure 33- Kafka publish/subscribe with two subscribers

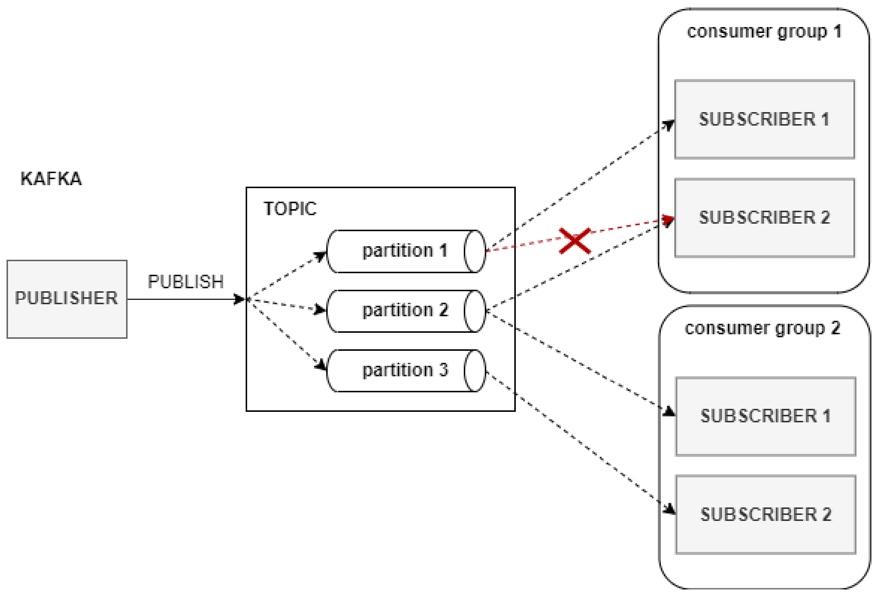


Figure 34- Kafka publish/subscribe with two consumer groups

One feature kafka lacks is the ability to support routing.

3.5. Overall Analysis

This section will be used to provide a brief overview of the different solutions and technologies and a few of their benefits and disadvantages.

	Microsoft Biztalk	Boomi AtomSphere	Workato
Functionalities	Integration of Different Systems	Integration of Different Systems	Integration of Different Systems
Cloud Functionality	on-premises	On-cloud with on-premises extension	On-cloud with on-premises extension
Language/Framework	.Net	Java	-
Compatibility OS	Windows	Windows, Linux	Linux, Windows, Mac OS, Windows Server
Internal Messaging Language	XML	-	JSON-like Datapills
Connection Types Allowed	[10]	[15]	[27]
Allows Business Rules?	Yes	Yes	Yes
Document Type Supported	XML, JSON, Flat File, Database	XML, JSON, Flat File, Database and EDI	XML, JSON, Flat File, Database

Data/Messaging Sharing

The moment data gets inside a Message Broker through an endpoint, it needs to share data across the different elements of the system such as Shapes. The way the data is shared between those elements may vary and depend on the way the system is structured. For instance, Microsoft's Biztalk Server separates the system in 3 different parts or layers, the Messaging Layer, the Orchestration Layer and the Databases Layer, and the way these separate layers communicate with each other is via a publish/subscribe paradigm and a common language. This methodology is positive when it comes to modularization, as it allows different parts, for instance, to be run on different systems, and communicate via messaging, is less positive, however, in situations we want to track or predict the progress or path of a message as it does not have a fixed flow or path and implementation can be complex.

Boomi has a more Linear approach. Since the different integrations are modularized in smaller, self-contained processes and the different Shapes are connected to one another, it passes the documents (data) from Shape to Shape. This allows a more predictive flow of documents and eases tracking and logging. Boomi has no internal data language and therefore passes the documents from Shape to Shape in the data format of the previous Shape it came from, this increases the likelihood of error as the user must remember to always parse into the data format necessary before any processing.

Workato has a json-like internal language, datapills, and has also a Linear Approach when it comes to Actions Flow. Modularized and adept of smaller integrations (Recipes), workato Actions have data outputs, that are stored as their datapills. This is better than Boomis' as it allows Actions to refer to data from any previous Action, making it easier and error prone for the user to reuse data and making it better for the logging system to store the data flow and contents of each step.

Data Consistency

First point to consider is data flow consistency. Both Microsoft and Workato use a common internal data type to communicate between their inner actions. For instance, when Biztalk receives messages from an endpoint, it parses the message into XML, and XML is then used for transformation and other features. Workato, once it receives data by a connector, it automatically parses the received data into an expected json-like common data structure, the datapill, and the data is then all accessed and referenced via those datapills. This makes easier for the user to know what data type and format to expect, and an universal way to reference it. Boomi, however, once it receives data it does not parse the information and the data flows from Shape to Shape with the exact format from the previous one. This increases the likelyhood of human error as data has to be parsed manually on the flow, and be remembered Shapes ahead when data needs some kind of processing.

Data Language

As seen above, it is beneficial to have an internal language as having a common data language allows for easier data referencing and mapping as it is known what data is expected. Two possible language choices for data storage and usage are XML and JSON.

XML was designed to be a markup language, and despite being used for data exchange it is not the most efficient way of doing so as it is more complex and due to its nature of tag usage, data redundancy is vastly present and it can rapidly increase the size of each message. JSON on the other hand is a lightweight language designed for data sharing. Easily parseable and human readable, JSON is the best approach.

Deployment

This second point to consider is deployment. Microsoft's Biztalk is designed to be an on-premise solution only, which implies the owner/user of the product has to install, maintain, configure and update all the infrastructure with it. This can be expensive, and time-consuming. On-premises, allows however, for access to local applications and databases without having to have an API open for external access. Dell's Boomi is cloud native, meaning the users can create integrations from anywhere on the go on the cloud without using local resources, via their application, causing the data storage and Processing are Boomi's responsibility. Boomi offers, however, an option to download the patented Atoms to premise, and run them locally, so users, can access local applications in their integrations via their processes. Workato also provides both Application integration on cloud and on-premises. Workato uses a concept of Agents, that can be downloaded and run on-premises to create a connection with the cloud integration, so that the company can access internal applications without opening ports for that.

On-premises functionalities may be a plus, however, for cases that companies use cloud storage and services, and for integration between business APIS, hosting a cloud service may be beneficial.

Scalability

Scalability is an important matter when it comes to integration. Over times companies increase their partners and information exchange and amount of data shared increases proportionally with the partners integrated. Being scalable is the ability to deal with increasing integrations and data associated. When it comes to Biztalk, Biztalk can use a concept of Host, in which a host contains/is associated with the different elements such as adapters and orchestrations and can be run separately on different machines.

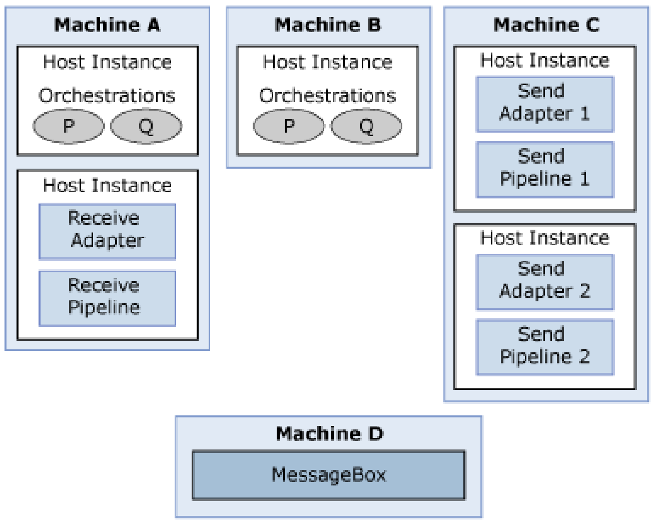


Figure 35- Host Concept

The example shows how orchestration instances and adapters are run on different host instances and different machines. Host instances are Windows Services.
Boomi's processes can be run on Atoms, their patented lightweight, dynamic runtime engine, and the nature of small integrations allow breaking several business integrations into smaller ones, and then run separately. Not to mention from a user point of view, they do not need to run locally all Atoms.
Workato is based on cloud and the on-premises connection is made using Agents. Agents provide a WebSocket tunnel to the workato platform, to access local

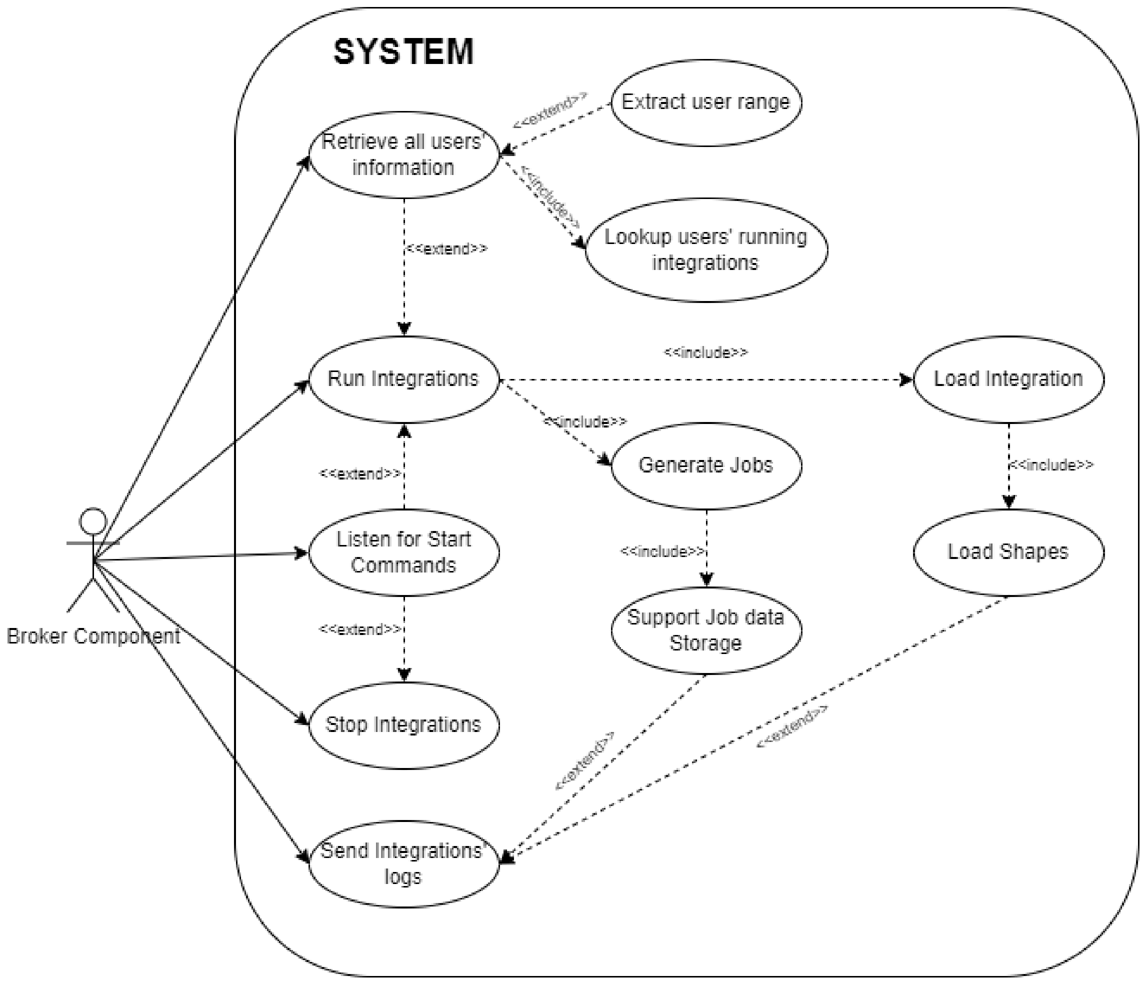
applications, allowing multiple Agents together into on-prem groups to share/balance load.

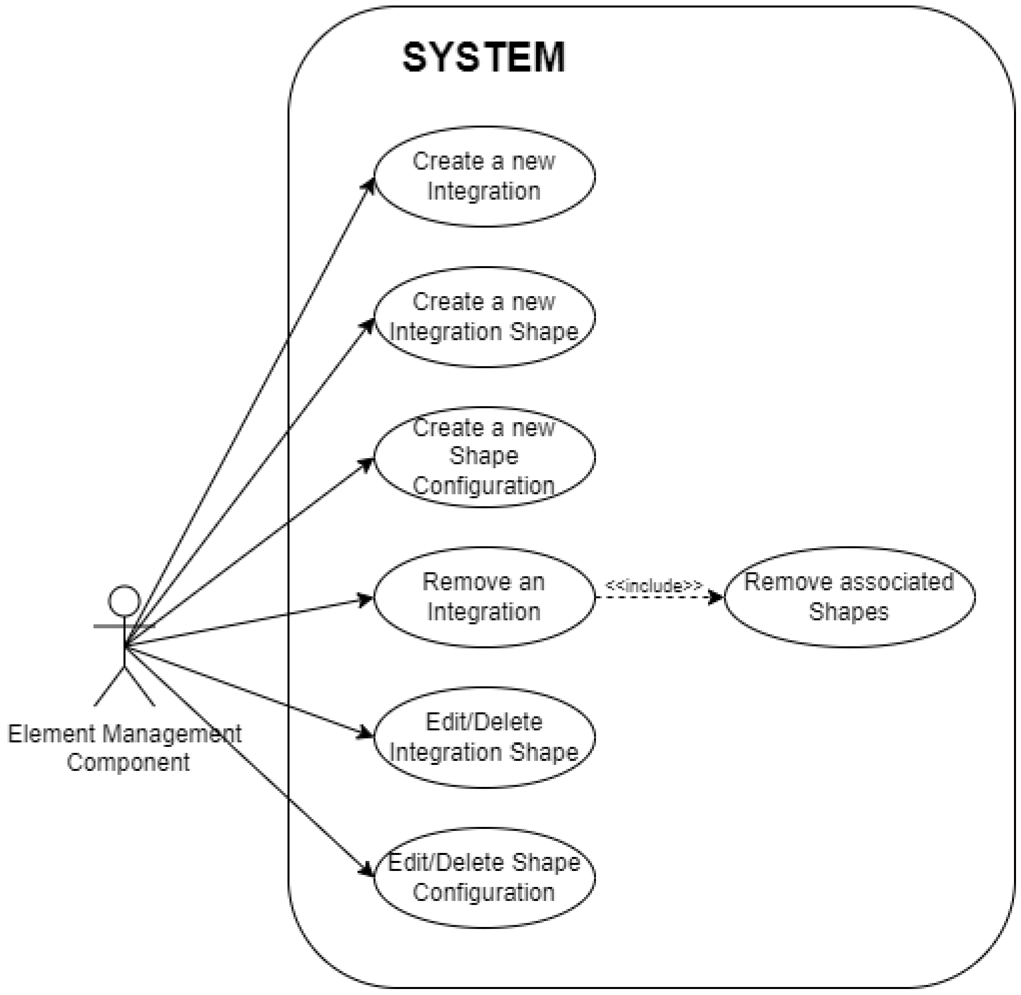
Broker Messaging Support

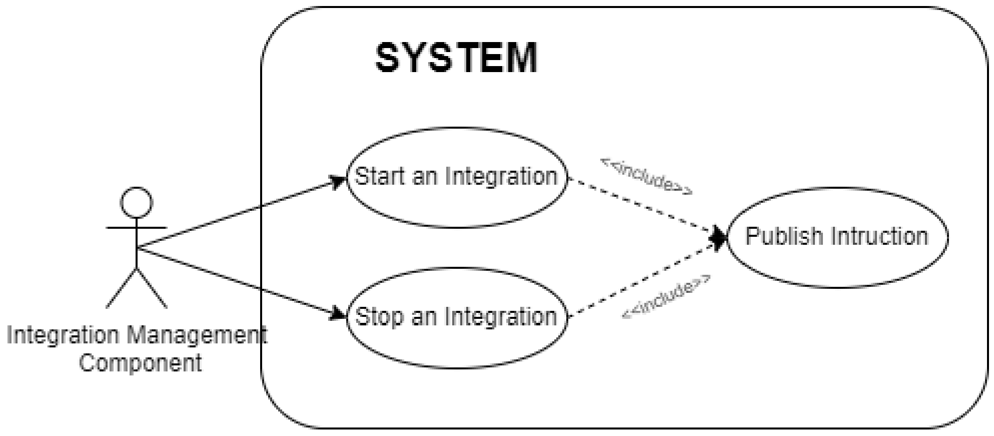
When it comes to developing a broker’s architecture one must consider possible messaging systems to support it. Service buses have point-to-point implementations which are an important when considering a few features. For instance, messages may need to be consumed only once even when there are more than one consumer, which is applied, for instance, in load balancing, as multiple consumers, in different systems can be reading from the same queue, sharing processing needs. Point-to-point implementations also allow data to be processed in a ordering manner as long as there is only one consumer, as the moment consumer number increases above one, if one consumer fails the processing or takes long enough processing a message, it may not be processed in an orderly fashion. Also limiting one consumer may cause a performance drop and incapacity to grow and expand the system, meaning it may not be ideal where ordering and scalability is fundamental. Despite kafka not supporting queueing, ordered processing of the messages can be accomplished, equally, by using a single consumer and topic, which is not ideal, as it has the scalability drawback, but scalability can be assured if we consider the increase of partitions number along with consumers number within a consumer group, needing however a hashing mechanism to guarantee that all related messages are sent to a single partition based on a characteristic such as id but complexity and maintenance makes it also not ideal. Another point to consider, for instance, is the fact that, updating the number of partitions to increase scalability, affects the hash partitioning and therefore may cause a partition to have data that should not belong there anymore.

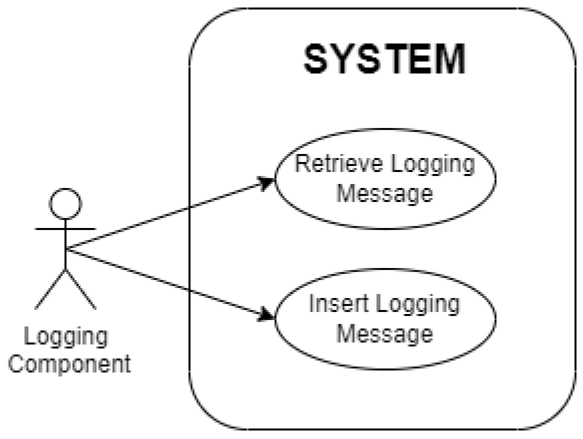
A point to consider is the fact Kafka was also developed to store all the data received permanently, and more appropriate for data revisit and real-time processing and analysis, which is not the most essencial feature for a message broker, on the other hand a service bus holds data temporarily and is naturally consumed when sucessfully read, this makes the broker more of a transient message delivery system rather than a message storage. Information that requires storage can be stored in an SQL database, such as logs.

Chapter 4 Requirements

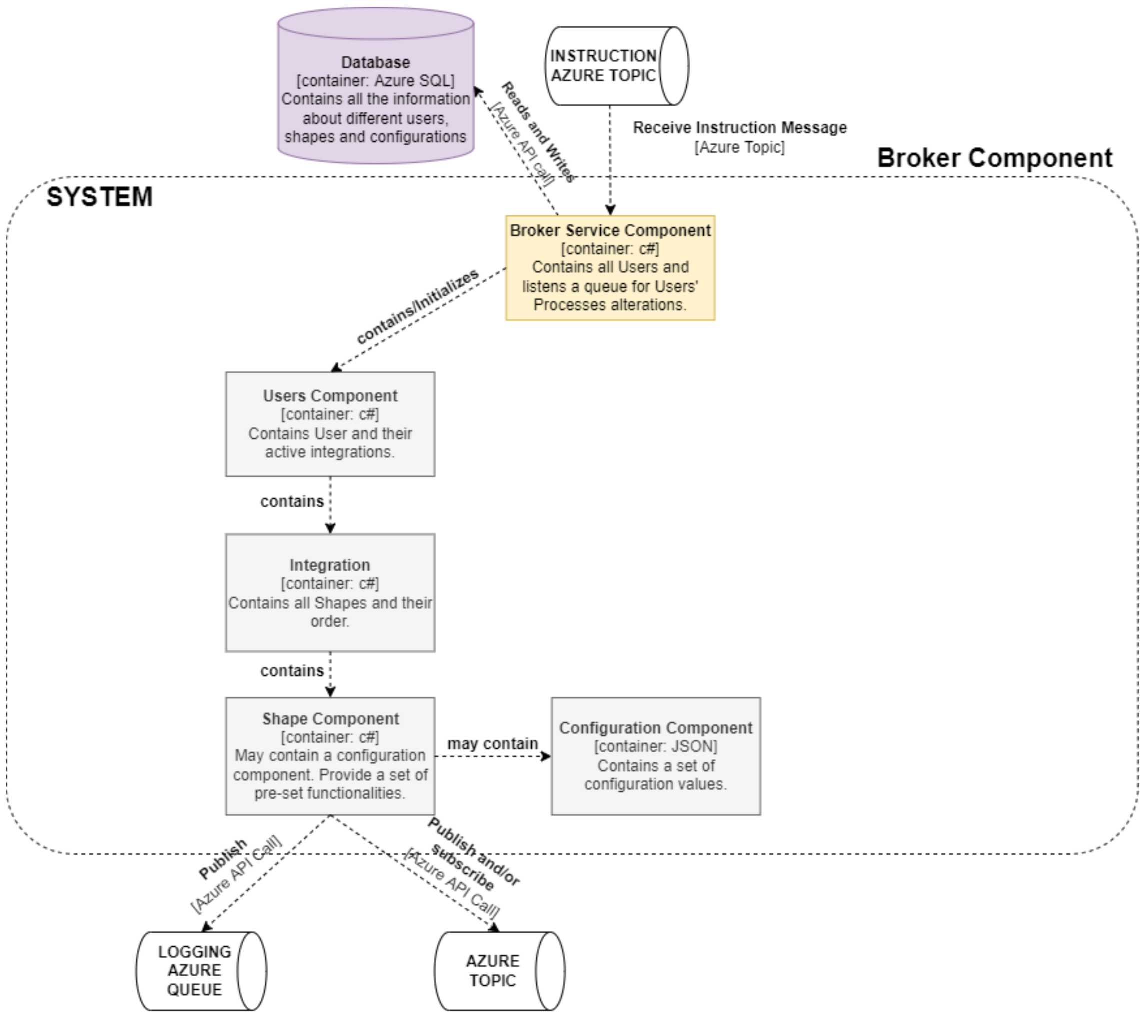


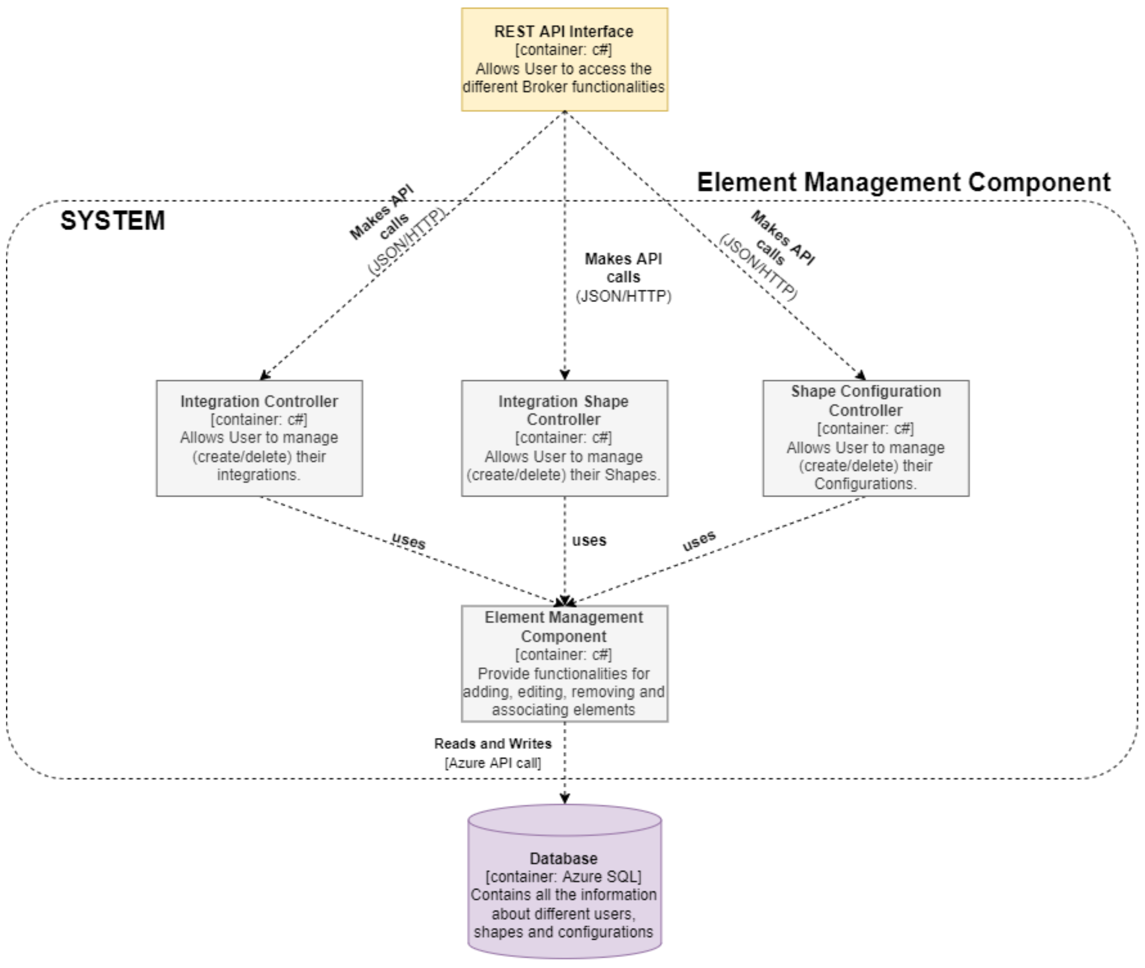

SYSTEM
Create a new Integration
Create a new Integration Shape
Create a new Shape Configuration
Remove an Integration
<<include>>
Remove associated Shapes
Edit/Delete Integration Shape
Edit/Delete Shape Configuration
Element Management Component

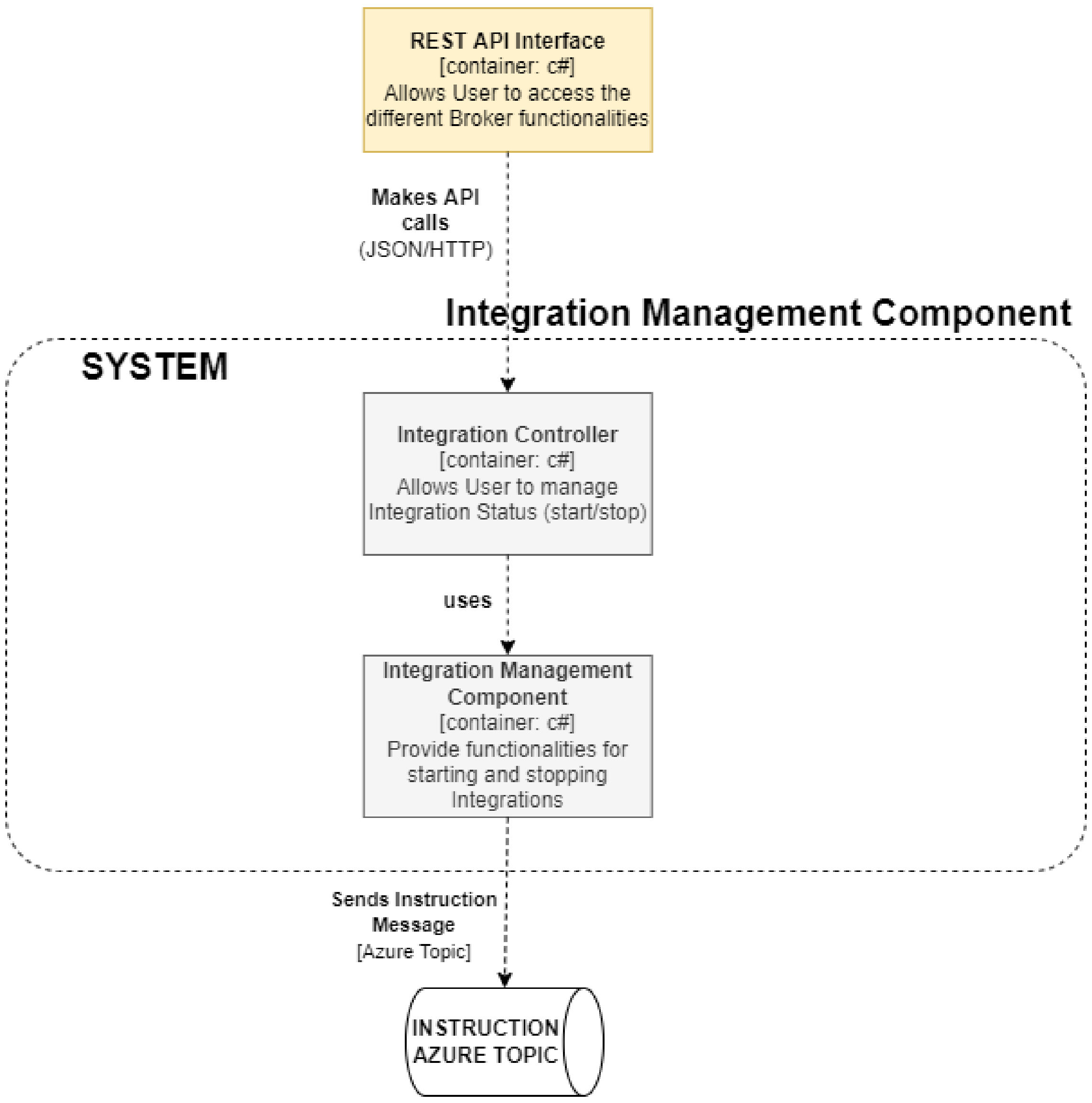

SYSTEM
Start an Integration
Stop an Integration
<<include>>
<<include>>
Publish Intruction
Integration Management Component

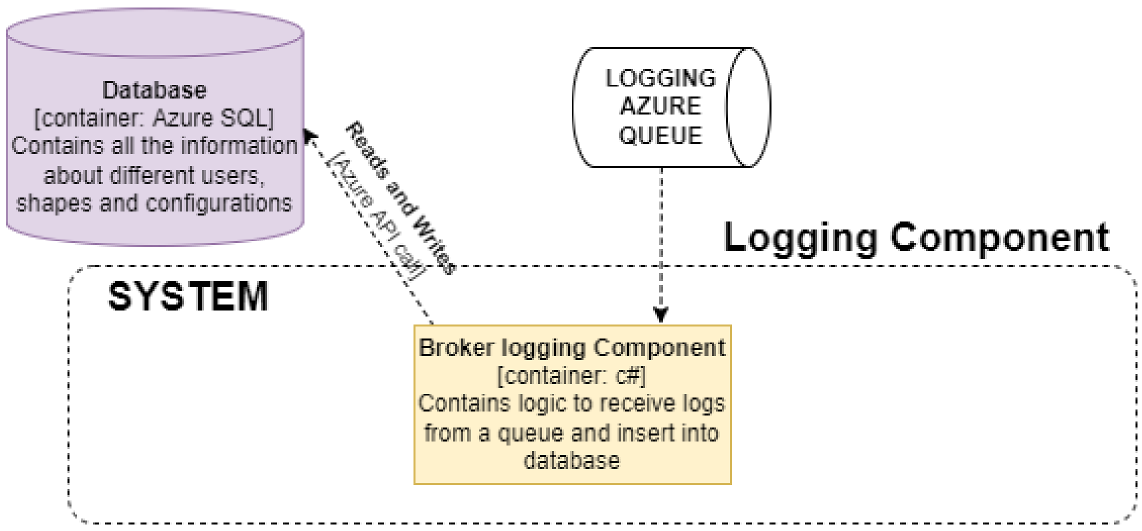

SYSTEM
Retrieve Logging Message
Insert Logging Message
Logging Component

Chapter 5 Implementation




REST API Interface
[container: c#]
Allows User to access the different Broker functionalities
Element Management Component
SYSTEM
Makes API calls (JSON/HTTP)
Makes API calls (JSON/HTTP)
Makes API calls (JSON/HTTP)
Integration Controller
[container: c#]
Allows User to manage (create/delete) their integrations.
Integration Shape Controller
[container: c#]
Allows User to manage (create/delete) their Shapes.
Shape Configuration Controller
[container: c#]
Allows User to manage (create/delete) their Configurations.
uses
uses
uses
Element Management Component
[container: c#]
Provide functionalities for adding, editing, removing and associating elements
Reads and Writes
[Azure API call]
Database
[container: Azure SQL]
Contains all the information about different users, shapes and configurations


REST API Interface
[container: c#]
Allows User to access the different Broker functionalities
Makes API calls
(JSON/HTTP)
Integration Management Component
SYSTEM
Integration Controller
[container: c#]
Allows User to manage Integration Status (start/stop)
uses
Integration Management Component
[container: c#]
Provide functionalities for starting and stopping Integrations
Sends Instruction Message
[Azure Topic]
INSTRUCTION AZURE TOPIC


Database
[container: Azure SQL]
Contains all the information about different users, shapes and configurations
Reads and Writes
[Azure API call]
LOGGING AZURE QUEUE
Logging Component
SYSTEM
Broker logging Component
[container: c#]
Contains logic to receive logs from a queue and insert into database

References

[1] Alfred D. Chandler, Jr. Strategy and Structure: *Chapters in the History of the American Industrial Enterprise*: MIT Press, 1969.

[2] *What is ERP*, Oracle. Accessed on: Dec. 18, 2021. [Online]. Available:

https://www.oracle.com/erp/what-is-erp/#link2

[3] *Message Brokers*, IBM. Accessed on: Dec. 18, 2021. [Online]. Available:

https://www.ibm.com/cloud/learn/message-brokers

[4] Vivek Kale, *Guide to Cloud Computing for Business and Technology Managers*: CRC Press, 2014.

[5] Liam Quin, *Extensive Markup Language (XML)*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.w3.org/XML/#intro

[6] *XML Schema*. Accessed on Dec. 18, 2021. [Online]. Available:

https://en.wikipedia.org/wiki/XML_schema#:~:text=Two%20more%20expressive%20XML%20schema%20languages%20in%20widespread,XML%20document%20itself%2C%20or%20via%20some%20external%20means.

[7] David C. Fallside and Priscilla Walmsley, *XML Schema Part 0: Primer Second Edition*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.w3.org/TR/xmlschema-0/

[8] Michael Kay, *XSL Transformations Version 3.0*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.w3.org/TR/xslt-30/#what-is-xslt

[9] Norman Walsh and Anders Berglund and John Snelson, *XQuery and XPath Data Model*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.w3.org/TR/xpath-datamodel-30/#intro

[10] Mandi Ohlinger and Kent Sharkey and Saisang Cai (02, February 2021). *Biztalk Documentation*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.microsoft.com/en-us/biztalk/core/runtime-architecture

[11] Mandi Ohlinger and Kent Sharkey and Saisang Cai and Venu (02, February 2021). *Biztalk Server Message*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.microsoft.com/en-us/biztalk/core/the-biztalk-server-message

[12] Mandi Ohlinger and Kent Sharkey and Saisang Cai (02, February 2021). *Lifecycle of a Biztalk Message*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.microsoft.com/en-us/biztalk/core/lifecycle-of-a-message

[13] Mandi Ohlinger and Kent Sharkey and Saisang Cai (02, February 2021). *The MessageBox Database*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.microsoft.com/en-us/biztalk/core/the-messagebox-database

[14] Boomi, *Boomi Integration and iPaaS*. Accessed on Dec. 18, 2021. [Online]. Available:

https://help.boomi.com/bundle/integration/page/c-atm-Integration_and_iPaaS.html

[15] Boomi, *Boomi Technology Connectors*. Accessed on Dec. 18, 2021. [Online]. Available:

https://help.boomi.com/bundle/connectors/page/c-atm-Technology_connectors.html

[16] Boomi, *Boomi Getting Started*. Accessed on Dec. 18, 2021. [Online]. Available:

https://help.boomi.com/bundle/integration/page/c-atm-Getting_started.html

[17] Boomi, *Boomi Essentials Training*. Accessed on Dec. 18, 2021. [Online]. Available:

https://train.boomi.com/courses/2304d0ac-b9fa-4c43-af4a-f2851285dfc2#5d04f307-de4f-4153-9281-2d484276bb85

[18] W3Schools, *Stored Procedure*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.w3schools.com/SQL/sql_stored_procedures.asp

[19] Workato, *Workato LinkdIn: About*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.linkedin.com/company/workato

[20] *Workato EverybodyWiki*. Accessed on Dec. 18, 2021. [Online]. Available:

https://en.everybodywiki.com/Workato

[21] Workato. *Getting Started*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.workato.com/getting-started.html#getting-started

[22] Workato. *If Conditions*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.workato.com/features/if-conditions.html#if-conditions

[23] Workato. *Recipe Jobs*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.workato.com/recipes/jobs.html#viewing-details-for-a-specific-job

[24] *Introducing JSON*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.json.org/json-en.html

[25] Martin Drapeau (2018, October 6). *Our Friends CSV and JSON*. Accessed on Dec. 18, 2021. [Online]. Available:

https://medium.com/@martindrapeau/the-state-of-csv-and-json-d97d1486333#:~:text=JSON%20was%20created%20by%20Douglas%20Crockford%20in%2

02001%E2%80%932002,a%20JavaScript%20object.%20It%20looked%20something%20like%20this%3A

[26] Two Bit History (2017, September 21). *The Rise and Rise of JSON*. Accessed on Dec. 18, 2021. [Online]. Available:

https://twobithistory.org/2017/09/21/the-rise-and-rise-of-json.html

[27] Workato. *Workato Connectors*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.workato.com/connectors.html

[28] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns*. Addison-Wesley Professional: 2004.

[29] Matt Milner, *PluralSight Learn BizTalk*. Accessed on Dec. 18, 2021. [Online]. Available:

https://www.pluralsight.com/blog/it-ops/learning-path-biztalk

[30] Bill Wagner and Rodrigo C. M. Santos and Kent Sharkey and David Coulter and Ben Newcomb and Genevieve Warren and David Pine and Nick Schonning and Scott Addie and Petr Kulikov and Maira Wenzel and Youssef Victor and Luke Latham and Petr Onderka. *A tour of the C# language*. Accessed on Dec. 18, 2021. [Online]. Available:

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[31] Eran Stiller. *RabbitMQ vs. Kafka: An architect's dilemma*. Accessed on Dec. 18, 2021. [Online]. Available:

https://betterprogramming.pub/rabbitmq-vs-kafka-1ef22a041793

[32] Philippe Dobbelaere and Kyumars Sheyk Esmaili. *Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper*. Association for Computing Machinery: 2017

[33] Boomi Community. *How long are the process reporting logs kept? And, can the duration be changed?*. Accessed on Dec. 18, 2021. [Online]. Available:

https://community.boomi.com/s/question/0D51W00006As07QSAR/how-long-are-the-process-reporting-logs-kept-and-can-the-duration-be-changed

[34]

Chapter 5

Appendix

Chapter 5

Appendix A

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 5

Appendix B

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 5