## Abstract

Data is information translated into appropriate way, so that it can be transmittable, process able and storable. Handling persistent storage data, SQL is the most popular language for querying. Not Persistent data is also suitable for querying—in case of messages queued in a system. A Message is a piece of data transmitted over a medium. Messages, which waits for service in a Queue provides valuable information related to its route, it might be priorized and be attended before or re-routing to other available services, in order to reduce the queuing time.

Enterprise messages does not often work with only one client, as the time passes the number of clients increases, therefore software technology continues evolving. Messaging is a technique to communicate clients and services in asynchrony manner.

Java Message Services provides a language which can filter messages based on message’s header properties. The language can be used in java as String field for querying messages. Although is compatible to have a selector String, introduces a risk of type safe due to the manual manipulation of the query.

Domain-specific languages let increases expressiveness as they are more suited, to this case can be suited in queries so that it can applied to multiple message-enterprise-broker hubs.

Acknowledgment

Abbreviations and acronyms

Table of contents

# Introduction

## Problem Statement

### Motivation

Enterprise environment usually works around requests and services, request start out from clients and end up in service providers which process them and gives back a reply to client. Part of them are proceeded right after the client submit the request which returns the answer to client immediately, but others are queued until service gets available. In the meantime, request wait for undetermined amount of time.

Getting information of the elements of queues provides the possibility of improving the information service. It allows as to make decisions related to routing or sorting messages based on a business convenient strategy. Besides, queue’s information helps us to decide topics related to infrastructure, whether to set up new server instances or rearrange the actual one.

This study analyses enterprise message structure with multi companies converging to an array of brokers, it also examines the structure of message in order to filter and provides further operation related with their headers. In addition, this study provides a suitable API to handle String selector query in as a java statements.

### List research questions

The more suited a Domain-specific language is suited; the more expressivity is suited for a specific model. Consequently, what is the best way to create a language, suitable for querying non-persisting data?

In spite the fact that, there are several languages in data query domain, why do they are different? Is there any other language more expressive than others?

Expressiveness is defined as the capability of a language imprints a statement maintaining the most of meaning of an idea, Is the expressiveness of a language linked to an easy and effective manner to create queries?

How frequent do the systems use the message filtering -to worth the construction of a tool that facilitate the composition of queries?

# Background

## Domain Specific language

Domain-specific languages (DSLs) are computer programming languages of limited expressiveness focused on a particular domain. They offer concrete expressiveness and ease use in their domain of application, compared with general-purpose programming languages. The level of expressiveness can be determined by Chomsky Hierarchy:

Regular language grammars allow any single non-terminal to be rewritten as a terminal or a terminal followed by a non-terminal. Terminals refer to the words in the language, whereas non-terminals refer to the symbols that represent clusters of terminals and non-terminals. Together, they form the rules of the lexicon. Context-free rules allow any single non-terminal to be rewritten as any string of terminals and non-terminals. Context-sensitive grammars have rewrite rules where non-terminals can be rewritten as non-terminals surrounded by terminals (Jaskolski, 2015)

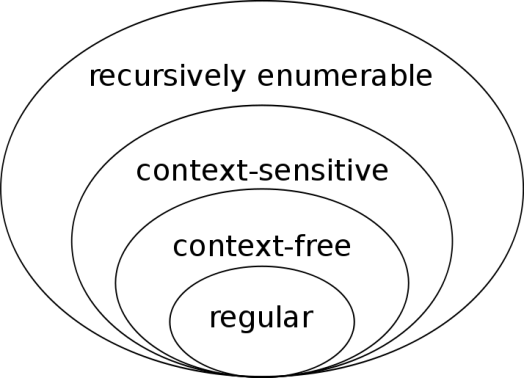


Figure : diagram of the languages on the Chomsky hierarchy (Finkelstein)

### Expressiveness of Language

The expressiveness of a language is determined from the inner-most restricted layer to the outer most expressive layer.

Built a language in order to be as expressive as possible, seems to be a more suitable for many cases, moreover the expressiveness is tied to computation time, the more expressive a language can be , the more time in process a statement will take.

DSLs trade general expressiveness of general purpose languages (GPL) for more specialized expressiveness. Finding a point to restrict the domain brings a scenario of abstraction concretization balancing. Because, the more expressiveness is increased, the more complexity of the language results within a domain

The general approach of GLP provides general solutions which in most of the case can’t be targeted in a specific domain, so simple solutions can be expressed in a complex way. Restricting the domain in smaller sets targeting the domain allows a more specific solution reducing a possible of complexity.

While the general approach of GPLs is able to provide general solutions, they are suboptimal and not targeted. Restricting the domain to a smaller set of problems allows for a more specific solution [4]. While DSLs may fall into any of the levels described in figure 2.1, in many cases they are within the more restricted layers as their  
grammar is more restricted than that of GPLs. GPLs would tend to be type 0.

### Implementation of a language

DSL can be implemented from scratch or extending a given base language. Implementing from scratch can be fairly difficult to design and implement, in addition there is a high chance that it doesn’t meet the expectations at the first time.

Moreover, the other approach, which a GPL host a specific language seams more convenient, given that people are already familiar in a host language. Inheriting the infrastructure of some other GLP and focusing in a specific domain, can result in a rich infrastructure reusing syntax, semantics, and language tools.

It can be called Domain-specific embedded language (DSELs). Either approach can be seen from the cost perspective; the initial cost of building DSLs can be higher at the beginning than using the same language, but at the end can save some cost. The key point in this decision is how well-defined the domain is.

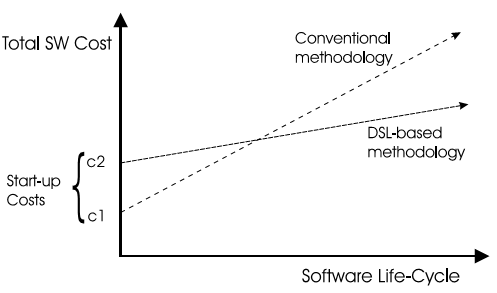
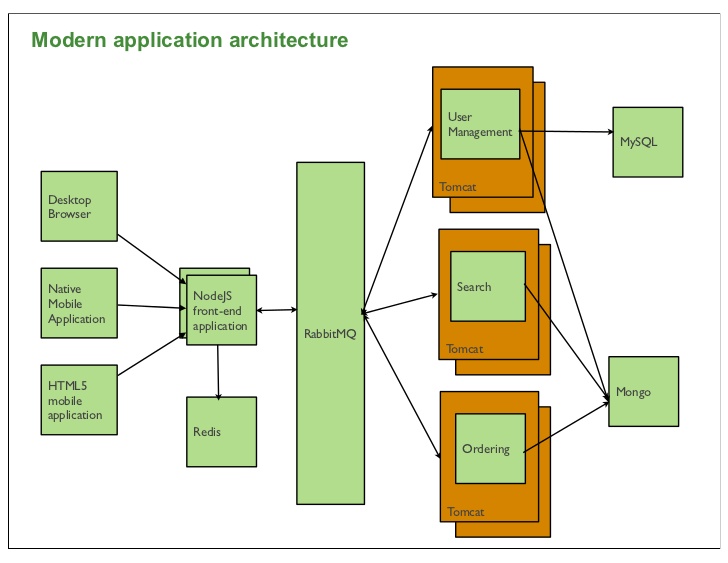


Figure : The payoff of DSL Technology (Hudak)

## Modern Applications

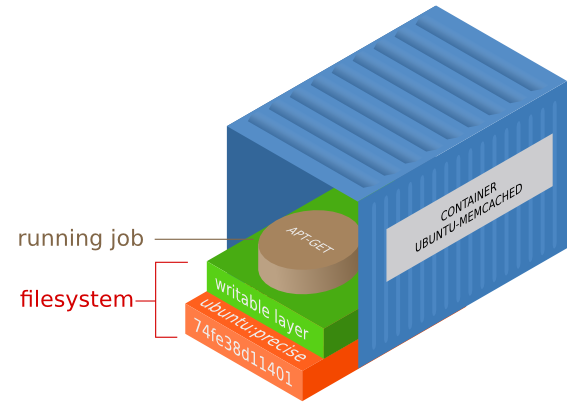


## Logic containers

### Virtualization

Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others (Singh, 2004)

A container is a runtime artifact, which wraps up a collection of software pieces in file system that contains everything it needs to run, so that can be deployed as an independent functional unit. Inside a container all logic and utilities to execute a business are set up, so that components like networking endpoints, operative system, and applications are packaged and deployed. Encapsulating all software components in a container guarantees that the container will always run in the same behavior, regardless of the environment it is running in.



For instance, a container can be composed by Ubuntu which has a writable layer; persistent data can be placed there. On top of the previous components can be installed an application.

The benefit of packaging in containers is the portability of the component, so it makes easy to transport the application across the data center or simple client computers.

Containers usually are instantiated from an image. Images can be seen as a template of the container which has a general version of the component. Container has more concrete configuration related the deployment, administration and network connections

## Enterprise Messaging

### Introduction

Over the years, the number of transactions in systems has grown significantly not only in volume of information but also in complexity. The complexity relies on the fact that business applications needs reliable, faster and sophisticate ways to communicate. Increasing the ways of processing and communicating business process can ease manageable and reduce complexity. It allows Enterprise software designers have more techniques, therefore tools to builds software, so that administrators have less hurdle in their tasks.

It’s worth to mention, that the successfully implementation of a software business process is not only under software team but also it relies in the business party. As software tools and techniques increase, business rules might encourage to make modifications in their flows.

Business flow deals with transactions and request, which is a way to communicate a client requests to services. Besides transactions are intended to be safe, which means that it ensures that a request will be made, knowing that any event that can happen against the normal execution of a request.

Not every kind of transaction are meant to be attended right after clients commits, instead it might take a longer, for instance a transaction goes forward along a business flow process which not all services are available upon the transaction submit. In consequence transactions are forced to be queued in some part of the business flow and wait for service.

One important concept related enterprise messaging is the asynchronous messaging within systems over network. In an asynchrony context, sender intends to deliver a message to a receiver, so it is not required for sender waits for the message to be received by the recipient right away. Sender can send the message and continues processing its next task. The message itself carries all data related with the business logic, which allow to process in the receiver.

In an Enterprise Application, a set of applications are interconnected through hub components, so that client use a API to compose a message and then delivers to a central hub component, which dispatch to receiver.

The central component is represented as Message-Oriented Middleware, which oversees all incoming messages and distribute to receivers according preset business deliver criteria.

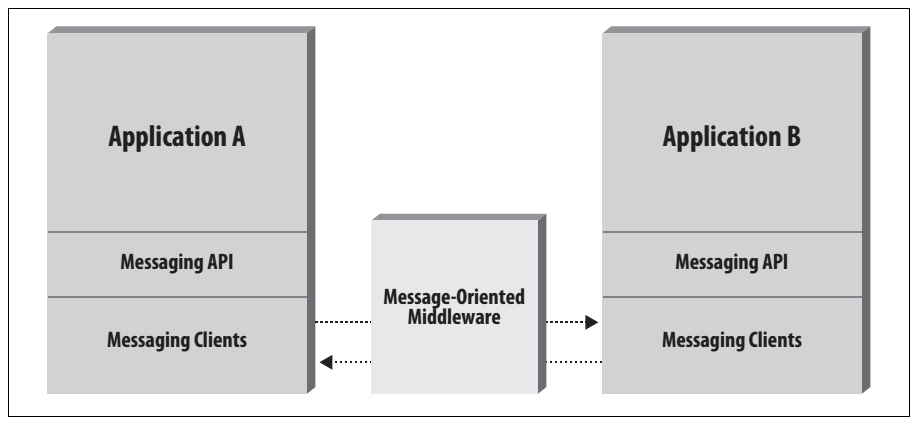


Figure Message-Oriented Middleware (Mark Richards, 2009)

The central component can be implemented using a message router, broker or Enterprise Service Bus (ESB) implementations. Its duty is to deliver incoming message to their corresponding receivers which are determined by the broker criteria, so as it is a decoupled system, the receivers can’t see each other, they only can see the central component, it is important when it works with multiple business parties.

One of the benefit of centralized architecture occurs when client sender doesn’t know who will be the receivers of a certain messages, broker can find matches within message’s properties and criteria stored in the broker, therefore forward the message to anyone who shows interest on the message. It shows its interesting matching parameters between the message properties and the broker criteria.

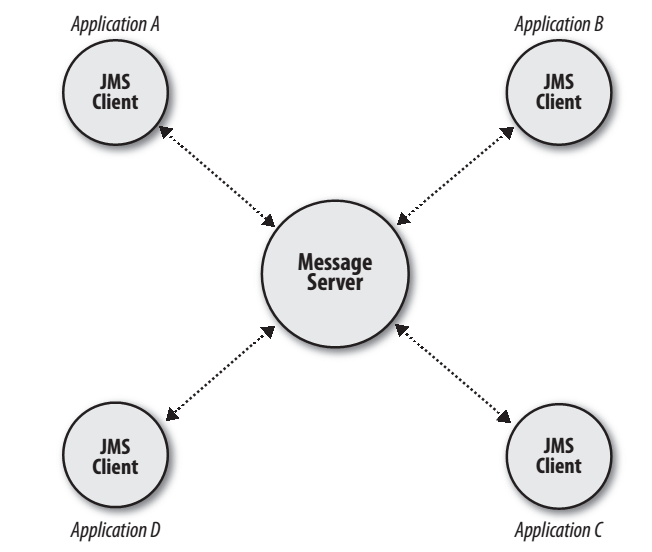


Figure . Centralized Architecture (Mark Richards, 2009)

Moreover, there is a decentralized architecture that does not depend of a central broker, so applications can communicate each other without it. The address to identify each application is the IP network address. This method can be used when the client has the address of the receiver.

a software suite with common business applications, tools for modeling how the entire organization works, and development tools for building applications unique to the organization,

### Message Broker

It is an intermediary program component, which translates a sender message protocol format into a one compatible to the receiver and route the message to receiver. It allows that a broker can receive message from multiple destinations and get them routed into their destinations.

The aim of a broker is take incoming message from client applications and perform some actions such as:

transform the message into an alternative representation without changing the client information attached on it.

Filtering messages according their label fields, so that any destination is filled only with messages on receiver interest. It performed by simple criteria on broker.

routing and transformed

## Service Oriented Architecture(SOA)

*Service-orientation* is a paradigm, which it enforces set up business process in terms of services and service-based development.

It defines services providers that are abstracted from the corresponding services implementations, so that a whole business process might be broken down in smaller process pieces. It allows better maintenance of each piece. However, having smaller process needs more management effort to maintain the business flow. There is a central core to overcome it, which coordinates the executions of process, failover services in case any of them are down, and further managements task the execution of the services. This component is called Enterprise Service Bus, which stands as a host for clients and service providers allocating them into a business process

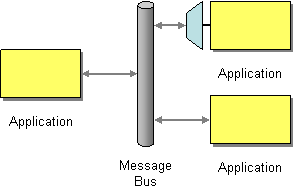
Moreover, in enterprise companies exist many generations of software, some of them are newer than other, connect them into one business process can be challenging, thus it might be written in different languages and programing paradigms. So, SOA implementations provide interfaces for many languages, so that even old but reliable software can be plugged into a new business process. Whether the client request might be synchronic or synchronic, ESB can act as a routing central.

In case of synchronic, a client request is expected to be proceed right after the request is placed into a system, so services are required to be up and running. Eventually some computational services can be unavailable for short periods or considering human work as a service, it is not considered active all the time, anyway it is encouraged not to lose client request.

A-synchronic request, are not meant to be proceed right away, likely the services are busy, but the request can be queued so it can be proceeded later.

SOA provides Messaging method to build an abstract layer within SOA needs to fully abstract business services from its underlying implementations. Through messaging services, the business service doesn’t need to be concern about the language the service is written of, deployed platform. Besides SOA provides a robust level of monitoring and control of request for coming in and out of an ESB. Almost all the commercial and open source ESB products available currently supports JMS messaging as communication protocol

Currently, transactions are considered as a multitier operation, which means that many services from different companies can be involved in the transaction. Either physical servers are in the same location or across the world, Messaging communications should have a standard interface to connect services each other.



## Java Persistent API

Enterprise applications are built based on the necessity to collect, transform and report vast amounts of information. The aim for storage it can be diverse depending of the business area. Either sensitive data or not, it should be kept somewhere. It became an important business nowadays. Although all software technology available for data management, application designers still spend much effort to keep it save and available.

There are many database concepts undergoing with several different applications, but the strongest concept that stays on front is the relational database. It turns out that most of the world corporate data is stored in relational database structure. For most of enterprise application, data structure design is crucial step in the information system lifecycle. Indeed databases often have a lifespan that continues long after the application has faded away.

The design of database or application has been always challenging for software engineers, whether design an application for a certain database or make a database for certain application. Either way the aim is to make them work together with a considerable maintenance.

Developing applications to work well with database systems is a commonly acknowledged hurdle of software development. Java success can be attributed to its widespread specification adoption for building enterprise database systems. Along diverse applications, java applications became very popular in enterprise context. As many designers are using it, Java specifications are reaching the center core of enterprise application development, so it influence in how applications interact each other.

Enterprise applications usually involve persistent data. The data is persistent because data needs to be part of multiple executions along the time, it usually persists for several years, depending regulations and company polices. Also during this time it might emerge changes in the program that uses data. After that time often outlast operating system, compilers and few times hardware.

During that time there will be many changes to the structure of the data in order to store new pieces of information without disturbing old pieces. These operations increase the risk to mishandle the data.

In spite the fact that Java platform has had in working closely with database systems, it is steel being affected by a concept of object oriented Programming. So working across java application with relational databases, moving data back and forth each other is a very frequent operation. Sometimes software designers calculate in terms of resource management, which at first seems to be accurate. Resources such as computation time, processing capacity, memory size, hardware comes often first to improve, relegating the accuracy and legibility of the code.

To make the data available, it is crucial to have a language so that data can be queried. SQL is a declarative language designed for managing data held in a relational database management system. It is widely used for many data base implementations, some frameworks has been built based on it.

Applications can access data through with embedded SQL queries within many applications languages. As it is embedded languages does not validate String queries in compile time, rather it executes string query and validate in runtime. Upon execution of the string query, errors of syntax and semantics are identified.

In addition, companies meet new business rules along the time so often make changes in their databases and applications, so they not used to build a new software, rather they modify their current software. Keep it doing along the time, makes the code of applications complicated or entangled database structure, either way SQL queries often are affected.

java applications the frequency of operations to databases thought SQL is high, is important to take a look how our SQL clauses are written.

A Java program consists of a set of class definitions, optionally grouped into packages. Each class encapsulates state and behavior appropriate to whatever the class models in the real world and may dictate access privileges of its members.

The properties of objects in general in a specific computer programming language, Such object models are usually defined using concepts such as class, generic function, message, inheritance, polymorphism, and encapsulation.

Moving data back and forth between a database system and the object model of a Java application was a lot harder than it needed to be.

. and in some cases very complex, SQL queries embedded in java language turn out to be difficult to manage in terms of legibility.

Java developers either wrote lots of code to convert row and column data into objects, or found themselves tied to proprietary frameworks that tried to hid e the database from them

## Java Message Service

### Introduction

Java Message Service (JMS) is an enterprise messaging system which provides a way to connect business applications each other using asynchrony messages.

Java applications can use JMS API to send and receive messages using a standard and vendor independent way.

Within the ways to communicate applications, each one has more benefits than others using it in a specific architecture. Features like loosely coupled can come up in case of integration of heterogeneous systems. Moreover, setting decoupled applications might lead to difficulties into the transaction integrity, such as handling different protocols and data synchronization.

Communications among applications components are often connected in a synchronic way, which force the components involved be ready at the moment of the transmission. Sometimes receivers or senders are not always available, and a number of requests are not meant to process right away, however requests can’t be misplace.

Communicating through JMS allows a set of functions related message operation, such as create, send, receive and read which allows a loosely coupled application.

Message routing describes how messages are transmitted between sender and receiver. JMS uses a simple message routing scheme, where both producer and consumer exchange messages connected to the same queue or topic.

For instance, a message sender delivers a message to a queue named ‘queue1’, therefore a message consumer create a session to the queue with the same name and therefore it proceed to retrieve the message.

Upon the message is attempt to retrieve, JMS do a message selection through a JMS Selector expression, which search message such that the result of the evaluation of the expression returns true. It acts as a filter expression.

Example of a sender

queue = (Queue) ctx.lookup(“queue1”);

qsender = qsession.createSender(queue);

msg = qsession.createTextMessage(“test1”);

qcon.start();

msg.setText(message);

qsender.send(msg);

Example of receiver using

Queue queue = (Queue)ic.lookup(QUEUE);

QueueReceiver qreceiver =qsession.createReceiver(queue);

qcon.start();

TextMessage msg = (TextMessage)qreceiver.receive(1000,”category=food”);

System.out.println(msg.getText());

### Interface Components

JMS Client: Enterprise applications sends or receives messages

Messages: Business exchangeable information among applications, it is packaged in a JMS format which will be process by a third party.

Managed Objects: are endpoints between server and client, it aims to manage the connections.

### Message Structure

JMS messages are composed by the following parts:

Header: Contains specific parameters used by both clients and providers to identify and route messages, all messages uses the same format.

Properties: contains an incorporate facility for adding optional fields related to the application business logic, which can describe the intention or category of the message. It helps to redirect messages to an appropriate application service according to filtering criteria.

Body: contains all details of the request. It is intended to be retrieved upon message arrive to receiver. It is intended to be read at message process phase. It can have different shape such as Map, Text, Byte,

### JMS Selectors

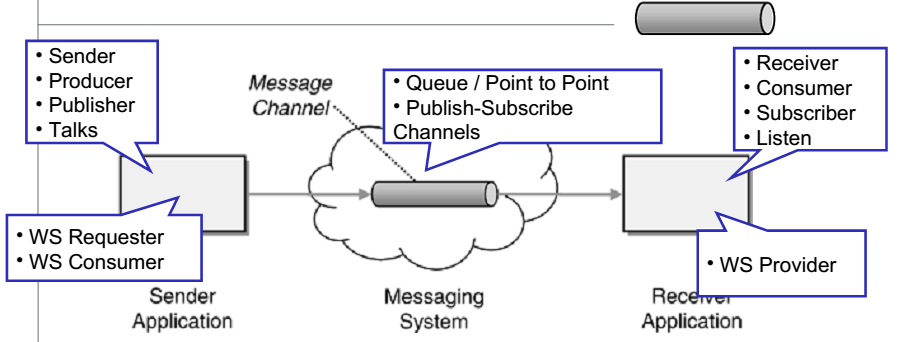
A message selector is a variable-length String, whose syntax is based on a subset of the SQL92 conditional expression syntax. It is used by an application to register its interest in certain property of a message.

It is written in a shape of expression that satisfies Structured Query Language.

The order of evaluation of a message selector is from left to right within precedence level. A selector can contain tokens, operators, and expressions conforming to the rules outlined here under.

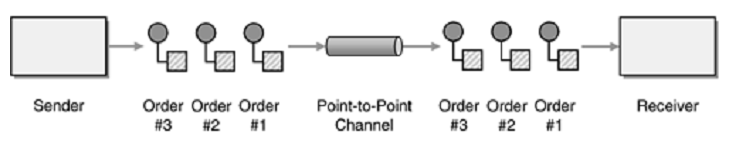
Arithmetic expressions are composed of themselves, arithmetic operations, identifiers (identifier value is treated as a numeric literal), and numeric literals.

### Message Channel



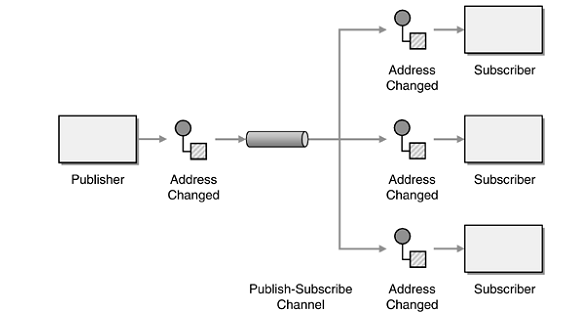
A message channel is a logical channel in a messaging system. It means, that messages sent through different channels provides a basic sorting functions into different message types. Message queues and message topics are examples of message channels. A physical channel is not the same that a logical channel

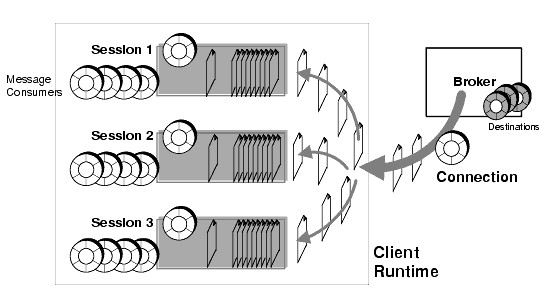
### Point to Point Channel



Point-to-point message allows JMS client to send and receive messages both synchronously and asynchronously based on virtual queues. In this model, there exist message senders and message receivers. The point-to-point messaging model has traditionally been a pull based or polling based model, where messages are requested from the queue instead of being pushed to the client automatically. A feature of using this channel is that messages sent to a queue are received by only one receiver, even though there might be many receivers listening on a queue for the same message.

### Publish-Suscribe Channel





In the publish-and-subscribe there are message producers labeled as publisher and consumers labeled as subscribers. Unlike the point-to-point model, messages published to a topic using publish-subscribe channel can be received by multiple subscribers. A subscriber express its interest in some specific topic registering to message broker by the name. The technique is like broadcasting messages, every subscriber which is registered to a specific topic will receive a copy of the message.

There are different kinds of subscribers, such as Non-Durable and durable. Nondurable subscriptions are temporary subscriptions that receive message only when subscribers are listening on the topic. In contrast, durable subscriptions receive a copy of the published message even if the listeners are offline upon the message is published. Subscribers will get offline messages come they back online again

## Advanced Message Queuing Protocol (AMQP)

AMPQ is an open standard application layer protocol for message-oriented middleware. At the beginning the aim was to solve problems related with finance industry heterogeneous platforms. AMPQ set out a standard to address interoperability issue. It consists in how the messages should be structured and transmitted between platforms to companies can easily exchange messages each other, regardless the broker or middleware vendors they use.

As AMPQ provides interoperability between several languages, JMS in spite the fact that has a very robust and mature specification, it does not provide interoperability outside Java Platform.

Although there are some implementations for connecting Java applications through JMS with non-Java applications such as python or Microsoft .NET, they are either proprietary solutions or challenging design and problematic implementation. It is always a solution, to implement a bridge between non-standard technology, however it increments not only the complexity, but also the cost of maintenance.

Moreover, there are protocols such as: STOMP or MQTT which address the same problem with different approach. MQTT focuses in the low footprint because it is being used in a low resources environment such as microcontrollers and mobile applications.

In the other hand, STOMP provides a text based solution, looking similar to an HTTP structure. A message has a header with properties and frame body. The idea was basically to extend the current interoperability that HTTP has into message oriented environment.

Nevertheless, there are some broker implementations that can handle the mentioned protocols, one of them is rabbitMQ. It allows connections of heterogeneous applications taking advantage of any feature of each protocol, therefore the applications are less forced to set up one specific.

In comparison with JMS, AMPQ handles message routing differently than JMS. Instead sending a message directly to a queue, it sends the message to an exchange along with a routing key. Exchanges are bind to queues in AMQP. The Binding is a directive indicating where the message will be routed. Message consumers attach to a queue and receive a message from the queue that is bound to an exchange.

JMS flow: Producer 🡪 queue 🡪 consumer

AMQP: Producer 🡪 exchange 🡪 binding🡪 queue 🡪 consumer

AMPQ Exchanges are entities where messages are sent. Exchanges take a message and route it into zero or more queues. The routing algorithm used depends on the exchange type and rules called bindings. There are five directives for exchange: Direct, Fanout, Topic Exchange, header exchange and System Exchange.

In comparison. a JMS queue is consumed by no more than one client and a JMS topic may be consumed by multiple clients. In contrary AMQP only has queues that are only consumed by a single receiver. AMQP producers don’t publish directly to queues, instead it is sent into an exchange, which through its bindings may get sent to one queue or multiple queues depending its rules, effectively emulating JMS queues and topics

## Cloud computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models. (NIST, 2011)

According to NIST, cloud computing leverages the continues expansion and availability of the computing resources, pursuing the lowest management effort. Application designers often are encouraged to deliver portable solutions, which are intended to be suitable for deployment as a service, reducing configuration as much as possible. It leads to use higher level of software technologies, such as containers or virtual machines. Thus, any of them use to have all embedded necessary components to run. Therefore, it can be considered as a logic functional unit, which can connect with other units or external resources.

Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations) (NIST, 2011).

Heterogeneous systems can work together or can be integrate when those comply a set of compatible standards. AMQP is a protocol suitable for messaging exchange, whether application’s language is the same or no. Unlike JMS in that case, needs a message broker that can bridge the platforms that are intended to be connected. It Restricts the aim of connecting thin or thick clients due to not all clients are capable to set up middle component.

Resource pooling. The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth

MOM (Message Oriented middleware) provides a central component which handles requests of all clients and send over a broker’s network. MOM model allows the creation of linked brokers according of clients demands. Those brokers can be created anywhere as long as are connected to the main one. It provides a location independence.

## Mishandled high risk operations

## Type safe

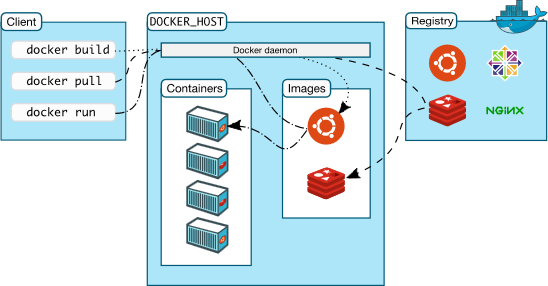
# Proposed Solution

## Container Business logic

### Docker

Docker is a new container software container technology, which allows the management of a small pieces of software packaged into a containers run as a complete functional units.

Docker is intended to benefit developers and system administrators in the way of software distribution along software lifecycle. For developers, it means they can focus on writing their code regardless the specific details of deployment process. For application operations, which care the implementation, load balancing and troubleshooting of everyday issues, docker gives simplicity because of portability. Many new versions can be override only replacing the production container, having in mind that it is not a persistent data. In consequence servers become stateless in sense that there are no leftovers from previous deployments or installations. In other words is easy and safe restart from scratch.



----FIX IT

Docker is a tool that makes it easy to package an application and all of its dependencies into a container. It does this by providing a toolset and an uni ed API for managing kernel-level technologies, such as LXC containers, cgroups and a copy-on-write lesystem.

Docker relies on AuFS (Advanced Multi-Layered Uni cation Filesystem) as a lesystem for containers. AuFS is a layered lesystem that can transparently overlay one or more existing lesystems. AuFS allows Docker to use certain images as the basis for containers. For example, you might have an Ubuntu image that can be used as the basis for many different containers. Thanks to AuFS, only one copy of the Ubuntu image is required, which results in savings in storage space and memory consumption, as well as the faster deployment of containers. Another bene t of using AuFS is the ability to version images. Each new version is simply a di 1 of changes from the previous version, e ectively keeping image les to a minimum. This also means that there always is a complete audit trail of what has changed from one version of a container to another, just like version control systems used in software development[13]

## Query DSL

QueryDSL was created in response of query maintaining, queries were made by String concatenation so when the query got long or complex, it results in an illegible code which hard to read. Sometimes special characters are not acceptable in host languages but are necessary in query, the use of escape characters; aggravate the reading, therefore modifying even a small part of query tends to be a challenge and unsafe.

Autocomplete feature is a very common utility in programming of most of the languages, so QueryDsl takes advantage of it and uses to build queries, so that programmers does not have to type the entire entities names or meet a String specific syntax. It makes the constructions safer and faster.

## Query languages

Java Persistent Query Language (JPQs) is a portable query language designed to combine the syntax and simple query semantics of SQL with the expressiveness of an object-oriented expression language (Schincariol, 2009).

JPQL (Java Persistence Query Language) relies in object oriented entities, so there is a representation of the elements of entity relational data model in the object-oriented programming environment. For instance, tables are represented by classes, table attributes as class fields and special annotations for relations within data-model tables.

The action of create classes or entities from tables is called table-mapping, which create an object representation of each component, so that each entity can be queried.

JPQL Queries can perform operations such as: reading, filtering, modifications and aggregations which are basically operations carried out by SQL query.

One of the most important reasons to endeavor the use of this method is the compilation time verification. Applications which use pure SQL as the query language embedded not only JAVA but also others; query sentences are intended to be checked at execution time, so the applications compiles regardless any SQL verification which are treated as string, therefore any syntactic or semantic error will raise at execution.

Error such as mistyped attribute or table names, case sensitive, embedded parameters, special characters or simple empty spaces can raise an error, which is not detected by the compiler consequently it forces to recompile the whole application.

As SQL queries retrieve data, indeed database connections are vital for most of applications, it becomes crucial to enforce a type safe method which prevents these issues. Using JPQL, queries are validated in compilation time, which saves time and effort. Let’s show the difference by an example:

A simple SQL String

EntityManager em = ...;

String jpql = "select p from Person where p.age > 18";

Query query = em.createQuery(jpql);

List result = query.getResultList();

The String jpql exemplifies an SQL querie, unfortunately at compilation time it does not show any problem, but there is one in the query. So the error will be found at line at execution of the query. Moreover, there exist cases where even if the query doesn’t throw exception but the result of the query return incorrect data.

Using JPQL to overcome this issue is shown below

EntityManager em = ...

CriteriaBuilder qb = em.getCriteriaBuilder();

CriteriaQuery<Person> c = qb.createQuery(Person.class);

Root<Person> p = c.from(Person.class);

Predicate condition = qb.gt(p.get(Person\_.age), 18);

c.where(condition);

TypedQuery<Person> q = em.createQuery(c);

List<Person> result = q.getResultList();

The use of criteria query and predicates fosters “Type Safe” quality in the query. It uses an API which maps the table Person into a java object so, CriteriaQuery knows which fields the element has so that it can perform JPQL queries based in java language. JPQL queries can be checked in compilation time.

Moreover for a programmer, writing SQL queries in a String format cause a multiples human recheck over parameters names in order to have accuracy of the type safe. Using JPQL, name fields are managed by java object syntaxes, so it is only necessary to make a call with dot operator, which means that JPQL criteria API does not allow the construction of queries that are syntactically incorrect.

Criteria query encapsulates the clauses of traditional query

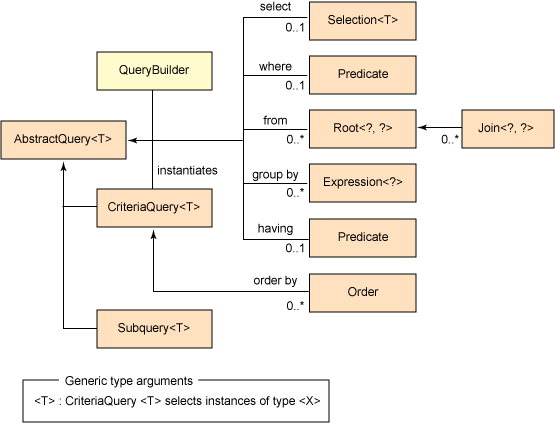


Figure Criteria Query (Poddar, 2009)

## Configuration Management

## Cloud operations

## Application Server

## Related work

### Query dsl

### JPA approach

### AMPQ and JMS

# Worked Example

## Introduction

The project simulates an e-commerce multi-party system. There are 3 main groups of components: Clients, brokers and Business Servers.

In the client component, user creates a request which is going to be packed into a JMS object message, so that the message will be send over the network towards a message broker.

Message Broker undertakes some tasks such as temporal storage of all incoming messages, as well as provide basic sorting and routing functionality over the stored messages. Once a message is received, it classifies into a virtual channel or queue, so that an application subscribed can retrieve and therefore process the message. During the message reside inside the broker, the message does not suffer any modification on the its content, however it is labeled, so that it denotes that the message passed by a specific broker.

Once the message is ready to be read, the subscriber takes over it, so it can be processed. Message broker is no longer on custody of that message, even if there is another subscriber that has a copy of it.

Every application server has its own logic to process client request, either way they use, the response message should be packed into a JMS object Message and send it back to the client.

## Client Component

Client component aims to simulate client requests, based on predefined data. Request are made in the style of a supermarket basket case. Request, as a classic invoice, is composed by a header and a list of items.

Header contains information about the client who made the request, as well as several data fields about the placement of the request such as date, budget, request number among others.

In the worked example header data of every request is chosen randomly, from a predefined set of user data set, so that broker upon message arrive can have more variety data request, consequently expose a higher level of expressivity of querydsl.

The item list is generated random as well. It is composed by an item name, amount and category in every item of the request.

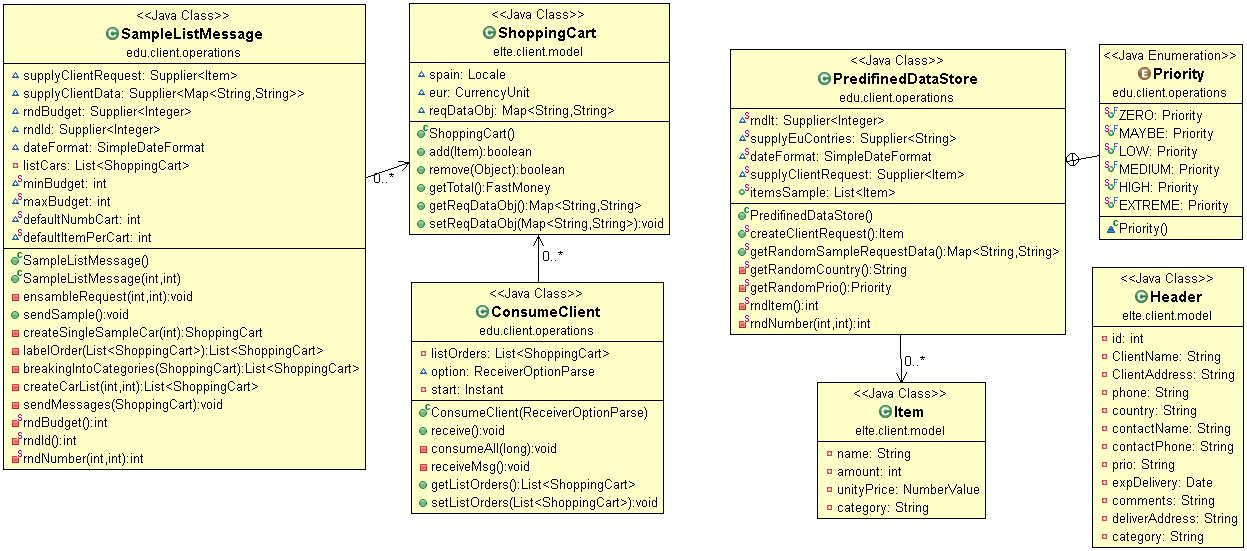
Because of broker classification, messages should be somehow classified during assembling. In a random mode, an item list can have diverse kind of items. Consequently, client component break the list of items into several lists. For instance, a list is break down into lists which have the same category, therefore treat them as individual requests. Nevertheless, every sub request is enumerated and its header information attached, so that in the way back broker or other application component can reassemble into a single response.

Indeed, brokers read only the header part of the message, so that it can apply some criteria of classification. In fact, brokers don’t deal with the body of the message, even if they can, because it is not brokers purpose

In message broker, upon message arrived, messages are classified base on their properties, and distributed into their respective topic.

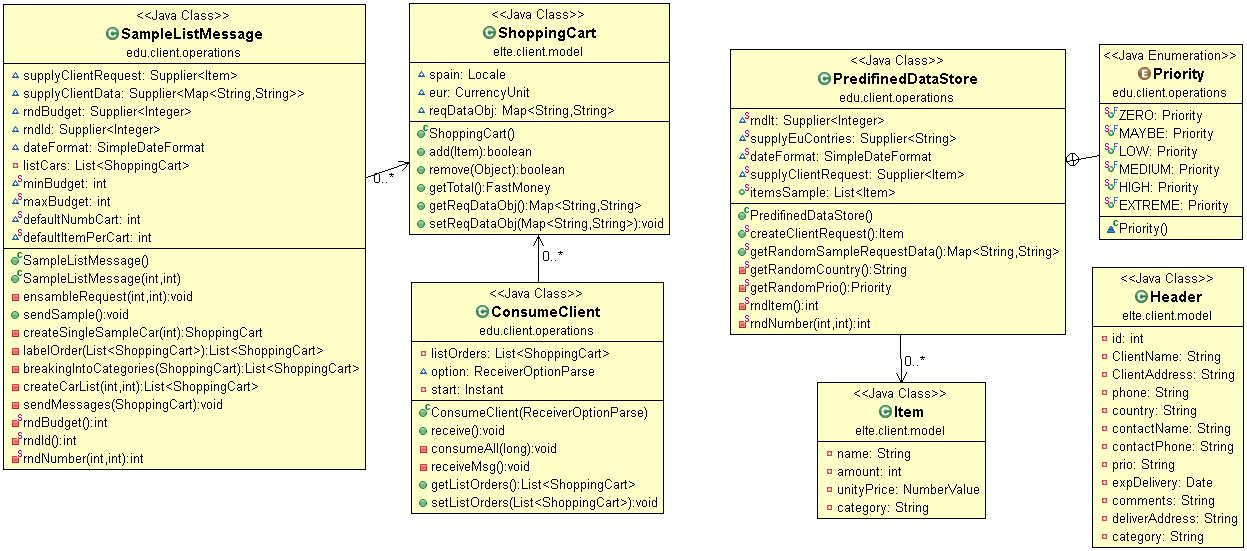
The list of items is full of selections of the client, so as a basic item element, every item is composed by 3 attributes: item name, item amount and category.

* Item name represents a label of a product that a client can request, seldom a client remembers and write the product in exactly as it is registered in the business application, many request might not match properly.
* Amount: It is the number of items required by the client
* Item Category: it is predefined classification of the item, related to the group of the item might belong of. It allows to route the request to the corresponding supplier.



Furthermore, lists of items are grouped into objects which comes from Shopping-Cart class, which has functions related to list management such as “add” and “remove” elements. The mentioned class is also used when the message arrives to the client, to reuse and have less number of classes.

Finally, SampleListMessage assembles the generated item set and header, then it appends some identification information to the message. Next it is packaged into a JMS Object and send it over the network towards the broker



The program can be run as a java jar application. The input interface accepts the number of orders or shopping carts and the number of items per order, it accepts parameters in the format –[option]= number. Let’s see an example

C:\Services\client>java -jar target\client.jar sender -i=1 -c=1

Or can be written into the long format

C:\Services\client>java -jar target\client.jar sender --carts=2 --items=3

The order of the parameters does not affect the command execution, either way the items will be read, in addition there is a command to check a syntax suggestion

C:\Services\client>java -jar target\client.jar sender –help

Output example:

C:\Services\client>java -jar target\client.jar sender --carts=1 --items=2

{carts=1, items=2}

Sent: Body:

[ClientRequest{name=Pineapple, amount=1, category=fruit}]

JMS Correlation ID: null

JMS timestamp: 1478947990793

JMS expiration: 1478950390778

JMS priority: 4

JMS delivery mode: 2

JMS reply to: null

JMS Redelivered: false

JMS Destination: 'amq.topic'/'requestTopic'; None

JMS Type: null

JMS MessageID: ID:662637a7-2d67-3dfd-baf1-692266f7885c

JMS Content-Type: application/java-object-stream

AMQ message number: -1

Properties:

date = 12/11/2016 11:53:09

country = United Kingdom

address = Heroes street

comments = it is for school assigment

qpid.subject = requestTopic

deliverAddress = devoi utca

phone = 22222222222222

itemsNumber = 1

name = Nicolas

subOrd = 1/2

id = 100

category = fruit

prio = HIGH

budget = 475

Sent: Body:

[ClientRequest{name=ridingequipment, amount=8, category=sport}]

JMS Correlation ID: null

JMS timestamp: 1478947990934

JMS expiration: 1478950390934

JMS priority: 4

JMS delivery mode: 2

JMS reply to: null

JMS Redelivered: false

JMS Destination: 'amq.topic'/'requestTopic'; None

JMS Type: null

JMS MessageID: ID:784097a4-004a-3cf4-b4bd-362e0d7f2ce6

JMS Content-Type: application/java-object-stream

AMQ message number: -1

Properties:

date = 12/11/2016 11:53:09

country = Croatia

address = street 4

comments = for birthday party

qpid.subject = requestTopic

deliverAddress = street 1

phone = 999999999999

itemsNumber = 1

name = Laszlo

subOrd = 2/2

id = 100

category = sport

prio = LOW

budget = 493

## Qpid Broker Server

Message Brokers usually is dedicated on management of exchange messages, acts as a mediator component between client and server. It accepts and register subscriptions from application services interested on broker’s incoming messages, as well as conducts message queuing and forward messages to other compatibles brokers.

Qpid is software tool made by Apache Software Foundation, which implements AMQP. It offers a tool set of features such as queuing, security, transaction management among others. Its API supports multiple program languages, there are implementations for Pearl, Python, Ruby, .Net and JAVA JMS API. Alongside RabbitMQ, currently the most popular implementation of brokers, Qpid is the next popular choice.

Qpid has is composed by virtualHost nodes, which has exactly one virtualHost. Each virtualhost is backed by a persistent storage. It acts in case of durable subscriptions, which holds the messages even when subscriber is offline. Upon the subscriber gets back online, subscriber retrieves all offline messages so that can consume the message.

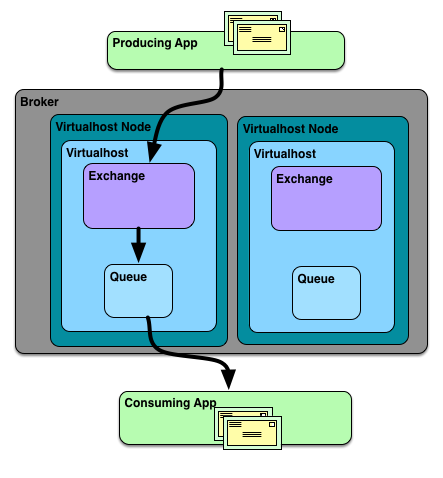


Figure Message Flow through Key Entities (Apache Software Foundation, 2016)

Exchange entity is located inside virtual host, it receives incoming messages from producers then routes them to matching queues within virtual host. Broker provides a set of exchange rules such as direct, topic, fanout, headers. The difference within them is the forward rule propagation. The One of the interest, which was used in the project, is Topic Exchange.

The figure illustrates the exchange topic strategy, whereas amq.topic describes a set of routing keys and queue names, where amq.topic determines the bindings within them. It introduces a keys expression, which are based in wildcards. \* (star) can substitute for exactly one word, and # (hash) can substitute for zero or more words.

The expressions are called routing keys, it must be a list of words, delimited by dots. The words can be anything, but usually they specify some features connected to the message.

SportStore

Supermarket

Type=topic

\*.energy.\*

\*.\*.food

natural.#

In this example, a producer P sends send request messages to an exchange X, which all describe a kind of food. The messages are sent with a routing key that consists of three words separated by a dot. The first word in the routing key will describe a source, the second if is a sport supplement and third if is it food: "<source>.<type>.<consistency>".

We created three bindings: SportStore queue is bound with binding key "\*.energy.\*" and Supermarket with "\*.\*.food" and "natural.#". SportStore describes its interested in all beverages and food which provides energy. Moreover Q2 wants to hear every kind of request related with food, and natural source.

A message with a routing key set to "artificial.energy.food" will be delivered to both queues. Message "natural.energy.juice" also will go to both of them. On the other hand " artificial.energy.beverage " will only go to the first queue, and "natural.fat.beverage" only to the second. "artificial.sweet.food" will be delivered to the second queue only once, even though it matches two bindings. "artificial.sweet.beverages" doesn't match any binding so it will be discarded.

Besides This method, which provides basic filtering based on messages key routing, Qpid support JMS selectors filtering. In spite that concatenating and matching routing keys seems very convenient, often JMS messages are not that easy with their parameters. In the project, client components show many fields in JMS header, so it wouldn’t be neither a clean or elegant solution using plain key routing. Instead, JMS has a selector specification, which handle the complexity of querying the header data of the message. It specifies a language based on SQL. Qpid uses it as filtering option.

Producer

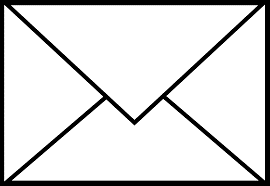
|  |  |  |
| --- | --- | --- |
| Binding topic | Queue | JMS Selector |
| requestTopic | sportStore | (CATEGORY =’sport’ OR CATEGORY =’soccer’ ) AND BUDGET >100 |
| requestTopic | Supermarket | CATEGORY =’food’ OR  BUDGET >100 AND COUNTRY IN (HUNGARY, AUSTRIA) |
| requestTopic | electronicStore | CATEGORY =’software’ OR CATEGORY =’hardware’ OR CATEGORY =periferics’and itemNumber >10 |

SportStore Queue

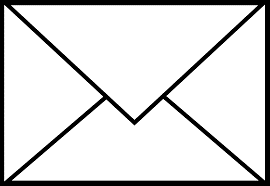
Supermarket Queue

ElectronicStore Queue

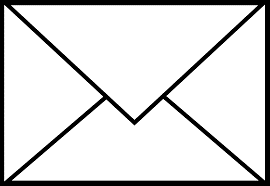
Non Classify Message(discarded)



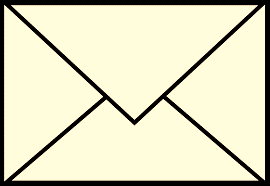
{category = ‘Sport’, budget = 400 , country = France }



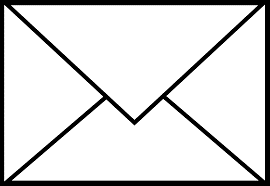
{category = ‘food, category and country = Hungary, item numbers= 19 }



{category = ‘software, and priority =high, and country= Austria}



{category = ‘soccer, budget 700, country= China}



{category = ‘periferics, itemNumber = 8, delivery expected =5 days, country =Italy }

Qpid Broker

Amq.topic

The figure above illustrates outgoing messages sent from producer which contains message properties related to a business request. ‘RequestTopic’ is the name of the entity where all incoming message will take as a hub. Every message might have similar properties, some of those can be null as well. Qpid filters incoming messages according of registered JMS selectors. The filtering process works like the previous example. The only difference is that JMS Selectors are immerse in the exchange process as a substitute of routing key. When a message fulfills a selector, the message will be routed to the selector’s queue. In case of a message fulfil more than one selector, the message will be delivered to every selector’s queue. In the remaining case that none of the selectors are satisfied by a message, the message will be discarded

Qpid has a management interface, which shows the number of message on each topic, as well as the amount of memory allocated used and current connections. In the snapshot bellow, shows that there are three servers subscribed with topic exchange strategy, thus every subscriber receives a copy of the message when its selector asserts true. Next, Queues panel shows all ongoing subscriptions, name of queue, number of messages on each queue. Type means the delivery semantics of the queue, standard means First-in-first-out queue, in addition there are more types like priority, sorted and last-Value-Queue. It shows the number of messages on queue and memory used, maximum boundaries on them are configurable, so that it does not reach memory limits. Messages can be removed from each queue from this point, so it provides a supervision point into the system.

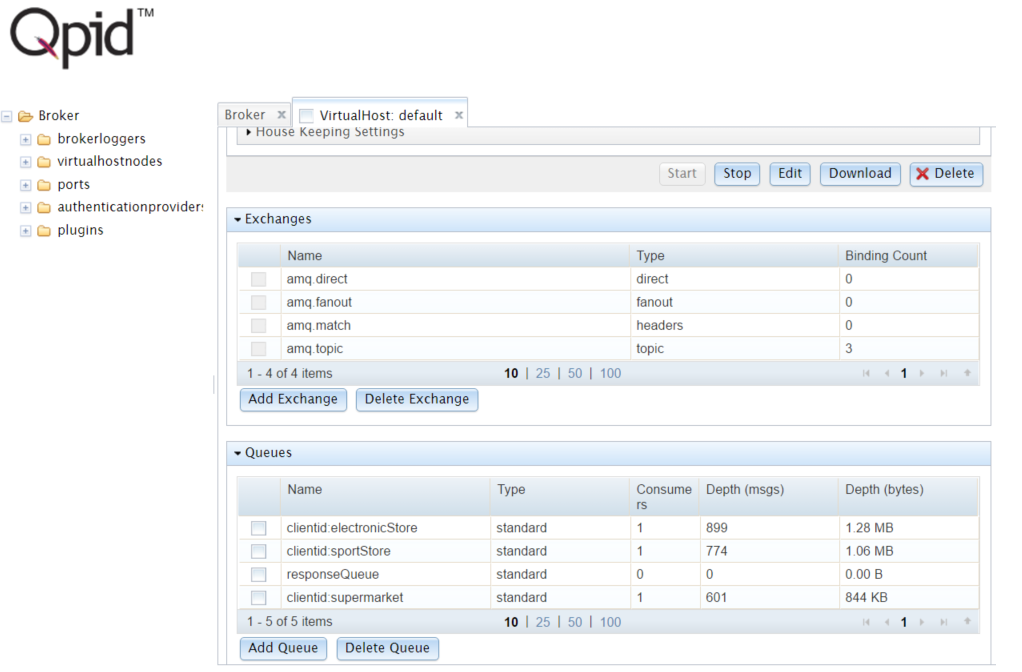


Figure Qpid Management interface

## Mongo Database

Mongo db is a NoSql database, which aims to have better flexibility on data storage structure compared that SQL databases.

For the sake of simplicity, the database chosen was mongo Db then populated from static data. Thereby sample data is stored in JSON format. JSON (JavaScript Object Notation) is an open, human and machine-readable standard that that simplifies data interchange. It is used along with XML format, which is the most common data format, which is used in different kind of applications. JSON standard supports basic data types such as numbers, string, Booleans, arrays, and hashes.

Mongo DB uses document’s Database to store records, just as tables and rows in a relational database. It uses JSON to save data and returns the same format. It can be easily parsed or with little transformation directly to many programming languages or straight to web using JavaScript. It reduces the number of transformations that a DB entry undergoes.

Moreover, as Mongo is NoSQL is less restricted than relational database in terms in constrains and integrity. It rather stores simple entries instead a structured data, thus not all items should have the same fields. Although It implies more flexibility, it reduces the integrity.

For instance, in the console prompt bellow shows two items of the supermarket database. The entries are not only composed by some common fields such as: id, class, name, amount, value; but also with not common like description. It means that one entry does not have values on a field then id doesn’t save it, it allows save space avoiding null values, as might happen in relational database.

The business application searches the requested item into the database, so first check by name and then retrieves the price and other available fields. Hence, the object is parsed from JSON to java Object, then the application proceeds to make the offer and queued into the application before reply to the client.

C:\>mongo

MongoDB shell version: 3.2.6

connecting to: test

> use supermarketdb

switched to db supermarketdb

> db.itemStore.find().pretty()

{

"\_id" : ObjectId("583c27b29c1b3c10781e27d1"),

"\_class" : "itst",

"name" : "backpack",

"amount" : 11,

"money" : {

"number" : "9"

},

"description" : "green",

}

{

"\_id" : ObjectId("583c27b29c1b3c10781e27d2"),

"\_class" : "itst",

"name" : "tent",

"amount" : 2,

"money" : {

"\_class" : "org.javamoney.moneta.spi.DefaultNumberValue",

"number" : "8"

}

}

## Server Component

The server component represents the business logic of the system. As the project was built within several business entities, any of them have a different way to process their logic. Integrating business services is often challenging when connection protocols are no standard or makes a custom connection logic. In real world, business applications services are written in many programming languages which goes beyond protocols and configurations, which make even more difficult to integrate them. However, there is a upcoming trend which make components standard as much as possible. Consequently, it encourages software architects and component developers to make input and output interfaces standard, so that component integrations get simpler and allow to have a better management on the business flow. In the worked example, server components were written in java, and simulates a heterogeneous system.

The server component contains a group of business applications each of them is registered to the broker, so upon the message is delivered from the broker the business application start its process. The registered applications are: Sport Store, Electronic Store, and Supermarket.

Sport Store is a prototype retail sport center which sells sport stuff, so for instance it Sport Store receives request related with categories such as: football, gym, fitness, shoes.

Electronic Store is another business application which sells products related with categories such as: software, hardware, computer.

Supermarket is a business application, which sells any products related with, food, beverages, shoes, fruits within others.

There are products within the stores, which categories can be found in more than one store. Then every store will receive the request, therefore client might have several offers.

Every store in this prototype has their own database with products, containing price and stock so that stores can work independently. Upon a request is delivered from broker, an application servers start looking up in its own database if the request can be satisfied then offers it a price, otherwise it ignores the request. Every application server has a request listener running, so it can notice the presence of a new incoming request.

After the process request, it is stored in a java Queue before replying offers. It is intended to have a stage where a human or another system can verify whether an offer is acceptable or not. Because, to make an offer not only depends on stock, but also on other parameters such as time to deliver, country, etc. which needs further verification. Once the verification is done, offers are sent back to the broker, then to the client.

Coming back to registration, which is made through java code, which is the process where a business server shows interest on specific item attributes of incoming request on the broker.

Before attempting connections, it is necessary to compose a JMS Selector so that, it will be used as a criteria parameter to filter request in the broker. JMS Selector is used in a similar way as exchange rules in AMQP.

Indeed, JMS selector’s syntax is similar with SQL query syntax, it comes from a subset of SQL called SQL92. Comparing to SQL queries and JMS selectors, both are query languages of a different data storage structure. On one side, SQL Databases save persistent data and intend to be robust structure. On the other side, JMS queues keeps temporal data which means that data is kept while a sorting or filtering operation is undertaken. In case of durable subscriptions data is stored when subscriber is offline. Nevertheless, once subscriber comes back online, broker intends to flush all messages stored in queues

JMS selectors are simpler than SQL queries, because SQL queries were designed to query a much larger and complex amount of persistent data. However, JMS selectors can reach a considerable level of complexity when messages come with multiple kinds of parameters on message’s header and multiple brokers linked routing messages based on their selectors assets.

Sometimes SQL queries turn to be complex because of the structure and labeling used. Applications queries those databases through plain String SQL, so it will be validated on runtime. Before runtime is uncertain if the query is valid, so it introduces a type safe issue.

JPA contributes easing the issue, creating object representation from database’s tables which are called Java Entities, it also creates a source of correspondence between data types of a database and java. Hence Java entities, which recognize data types and table name as well as its relations, can be acceded from application then create queries in a convenient mode. Queries are created through JPA criteria query, which is in java domain language mainly familiar with java application developers. Eventually, after JPA criteria query is completed, it will be compiled and result a plain SQL String which will be sent through application database

In the present project JPA approach has been taken in consideration in JMS selectors maters. Compositions of JMS selectors are made through Query Classes, Boolean Builders and Expressions which is a similar approach of JPA with SQL queries. Query classes are a representation of the message as a Java object similar as Java Entities. Boolean Builder takes predicates as parameters and concatenates them with logical connectors in a java language domain, it is similar to Criteria Query in JPA. Expressions is a generic class which is inherited by classes such as Boolean Expression, String Expressions and other similar. A Boolean expression embeds a java Boolean statement, so that it can provide a set of suitable methods to concatenate to another Expression then with another one and so on.

For example, RequestData is a java class with some fields, which has a special annotation denoting that is a query capable class. Upon the compiler start its process, a class generation process starts as well. The process search for the @QueryEntity annotation onto the class name, in order to create a new class where fields are taken into expressions which are capable to be concatenable through Boolean Builder.

@QueryEntity

public class RequestData {

private String id;

private String subOrd;

private String name;

private String address;

private String phone;

private String country;

private short budget;

…. more fields

private Object items;

public RequestData() {}

The new created class will be prefixed with a Q at front of the name so that it denotes that is a generated class and query capable component. RequestData is a Java class where contains the expected fields that incoming messages can have.

QRequestData is a generated class which allows that queries can be built base on concatenation of statements of the original class. For instance, RequestData has string fields category, date, budget, etc. which in QRequestData are wrapped into StringPath, DatePath<java.util.Date>, NumberPath<Short> respectively. All of them inherits from expression so they can be used as a parameter in a Boolean Builder.

import static com.querydsl.core.types.PathMetadataFactory.\*;

import com.querydsl.core.types.dsl.\*;

import com.querydsl.core.types.PathMetadata;

import javax.annotation.Generated;

import com.querydsl.core.types.Path;

/\*\*

\* QRequestData is a Querydsl query type for RequestData

\*/

@Generated("com.querydsl.codegen.EntitySerializer")

public class QRequestData extends EntityPathBase<RequestData> {

public static final QRequestData requestData = new QRequestData("requestData");

public final StringPath category = createString("category");

public final NumberPath<Short> budget = createNumber("budget", Short.class);

public final DatePath<java.util.Date> date = createDate("date", java.util.Date.class);

…more fields and constructors

}

Eventually, let’s look at Boolean Builder in action. It offers several ways to build a query. One approach is inline which is shows in the concatenation of category, it allows to concatenate Boolean statements with Boolean operators in one java statement. It improves legibility in case of a many Boolean condition of the same field are connected. In the example, category can be either software, hardware or computer, so it is easier to write and read them inline.

On the second approach, Boolean builder can compose a selector in a multiline way. It works like String Builder in Java, in this case instead of string it takes Boolean Statements, which are represented as Boolean Expressions. This approach is helpful when building rules of different fields. Although it appears like writing style, it helps in future modifications of the rules to make them more clear.

Moreover, it reduces the fact of type errors due to java query dsl API is checking it on compilation time, so there is no need to set up an execution of the program just to check whether the rules are syntactically and semantically correct or no.

public static String expressionComposer() {

QRequestData rd = QRequestData.requestData;

QueryGeneratorSample qg = new QueryGeneratorSample();

List<String> europeList = Arrays.asList("Belgium", "Bulgaria", … , "United Kingdom");

List formatedCountries= europeList.stream().map(x->"'"+x+"'").collect(Collectors.toList());

BooleanBuilder bb = new BooleanBuilder(rd.country.in(formatedCountries));

bb.and(rd.category.eq("software").or(rd.category.eq("hardware")).or(rd.category.eq("computer")));

bb.and(rd.date.isNotNull());

bb.and(rd.budget.between(100,900));

return qg.genQuery(bb);

}

Indeed, there are more features on the API such as concatenation of Boolean statements through OR, AND, or IN. On the previous example, europeList is a list of String with the name of the countries of Europe. The notation IN in the example assets whether the country of a request is in the list or not. On the background process it generates a JMS selector according with the specification of the syntax of IN.

In Addition, ‘OR’ Boolean operators, can be written with the expression andAnyOf which concatenate all Boolean Expressions parameters, then concatenate them with OR operator. In example Below shows the construction a selector based on three BooleanExpressions sp,sh,ts where the expression assets if the category of a request is equal to sport, shoes, test respectively, then concatenate them with OR. It is a multiline approach of the it also aims the improve the legibility of the rules inside java code.

BooleanExpression sp = rd.category.eq("sport");

BooleanExpression sh = rd.category.eq("shoes");

BooleanExpression ts = rd.category.eq("test");

BooleanBuilder builder = new BooleanBuilder();

builder.andAnyOf(sp, sh, ts);

assertEquals("category = 'sport' OR category = 'shoes' OR category = 'test'",serialize(builder));

Besides there are existent operators of arity different than two, such as between and NOT. In case of “between”, Boolean expression checks if the field is valid to operate with “between”, it means that it validates if is numeric value otherwise it raises a compilation error.

BooleanExpression sp = rd.category.eq("sport");

BooleanExpression sh = rd.category.eq("shoes");

BooleanExpression ts = rd.category.eq("test");

BooleanExpression bdg = rd.budget.between(100,500);

BooleanExpression itn = rd.budget.gt(100);

BooleanBuilder builder = new BooleanBuilder(sp);

builder.and(bdg.or(itn)).andNot(ts);

assertEquals("category = 'sport' AND ((budget BETWEEN 100 AND 500) OR budget > 100) AND NOT (category = 'test')", serialize(builder));

Based on JMS selectors features, selectors can perform basic arithmetical and comparison operation such greater than, lower or equal than a value field, add, subtract. Let’s see in the example below how to apply in the Expressions API.

Expression gt=rd.budget.gt(100);

Assert.assertEquals("budget > 100", serialize(gt));

Expression lt=rd.budget.lt(50);

Assert.assertEquals("budget < 50", serialize(lt));

Expression add=rd.budget.add(5);

Assert.assertEquals("budget + 5", serialize(add));

Expression sus=rd.budget.subtract(8);

Assert.assertEquals("budget - 8", serialize(sus));

Expression mul=rd.budget.multiply(3);

Assert.assertEquals("budget \* 3", serialize(mul));

Expression dv=rd.budget.divide(2);

Assert.assertEquals("budget / 2", serialize(dv));

Finally, some expression functions related with string fields.

Expression nul=rd.comments.isNull();

Assert.assertEquals("comments IS NULL", serialize(nul));

Expression stw=rd.category.startsWith("sp");

Assert.assertEquals("category LIKE 'sp%'", serialize(stw));

Expression end=rd.category.endsWith("rt");

Assert.assertEquals("category LIKE '%rt'", serialize(end));

## Cloud deployment

The project emulates multiple business parties converging to a one single center array of broker components. Business applications can be implemented in many ways, diverse numbers of standards, and a variety of versions on each one. In that sense, becomes a challenge to integrate them.

Implementing business applications, from the perspective of infrastructure, needs 3 main components: networks, processing machines and storage, so that application persistent data is saved in storage and can communicate with other applications through networks.

Cloud computing introduces virtual machines and containers, which are meant to be functional units capable to run programs and communicate with other components across the network.

In this project, as it works within multiple components, it was detached in functional units such as Client, broker, supermarket, electronic Store, sport Store. It is intended to run from different locations on production mode. The strategy was to make as loosely couple as possible so that it can be easily deployed into the cloud without any interdependency of them.

The project describes the necessities of a regular shopper client. First, the client submits a list of items needed which will be formatted as a market shopping list in a natural language. It has the client contact as a header and a java object attached with the list of items, eventually it will be composed as a JMS message and sent it to the broker.

Second, Broker receives the request, then evaluating their JMS selectors registered, then it allocate the message into corresponding topics, it does not send to application yet.

Third, Message is allocated in broker in a place where application can be read the request. Then when the application is working it activates their listeners and start consuming the allocated messages from broker. Upon the message arrive to the business application, starts processing requests one by one. It performs a database search for items and fit the quantities according the logic of the application, Once the application finish it returns a offer, then it compose a response message and queued into the application, ready for the reply instruction from the administrator. Once has the instruction. The offer is send back to the broker and then compiled within other offers from other servers then send it back to the client.

Finally, when clients get the offers can compare and make their purchases.

# Assessment

## Accuracy of technology selected

### Management on cloud

## Safer type query

## Client

## Code maintaining ???

## Comparison with modern applications trend

## Fields of application of the solution

## Similar approaches

## Mislead approaches

# Conclusion

## Results

## Research questions

## Limitations and future work

## Appendices

## Bibliography

Apache Software Foundation. (2016, February). Qpid for Java 6.1.0.

Finkelstein, B. J. (n.d.). Retrieved from commons.wikimedia.org: https://commons.wikimedia.org/w/index.php?curid=9405226

Hudak, P. (n.d.). *Modular Domain Specific Languages and Tools.* (D. o. Science, Ed.) New Haven: Yale Univertity.

Jaskolski, Y. (2015). *Domain-specific query languages.* Sydney, Australia: Macquarie University.

M. Mernik, J. H. (2005). When and how to develop domain-specific.

Mark Richards, R. M. (2009). *Java Message Service* (First ed.). Sebastopol, United States of America: O'Reilly.

NIST. (2011, 11). The NIST Definition of Cloud Computing. (T. G. Peter Mell, Compiler) United States of America.

Parsons, M. F. (2011). *Domain-Specific Languages.* Addison-Wesley.

Poddar, P. (2009, September). Retrieved from IBM developerWorks: https://www.ibm.com/developerworks/library/j-typesafejpa/

Schincariol, M. K. (2009). *JPA 2 Mastering the Java™ Persistence API.* New York, United States of America: Springer-Verlag.

Singh, A. (2004, January). Retrieved from An Introduction to Virtualization: http://www.kernelthread.com/publications/virtualization