**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

**Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores**


# Desenvolvimento de Sistemas de Informação Empresariais com tecnologias open source e java EE


**PEDRO MIGUEL FERREIRA MARQUES ALBERTO**

(Licenciado)


Relatório Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Doutor Walter Jorge Mendes Vieira

Júri:

Presidente: Doutor Nuno Miguel Soares Datia

Vogais: Doutor Vitor Manuel Guerra Vaz Silva
Doutor Walter Jorge Mendes Vieira

**Outubro, 2018**

**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

**Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores**


# Desenvolvimento de Sistemas de Informação Empresariais com tecnologias open source e java EE


**PEDRO MIGUEL FERREIRA MARQUES ALBERTO**

(Licenciado)


Relatório Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Doutor Walter Jorge Mendes Vieira

Júri:

Presidente: Doutor Nuno Miguel Soares Datia

Vogais: Doutor Vitor Manuel Guerra Vaz Silva
Doutor Walter Jorge Mendes Vieira

**Outubro, 2018**

*Aos meus ...*

# Agradecimentos

Para a realialização desta tese de mestrado foi necessário uma grande força de vontade e uma grande disciplina para conseguir conciliar com as restantes tarefas e responsabilidades do dia a dia, mas sem dúvida que sem a ajuda, colaboração e compreensão de algumas pessoas não seria possível concluir este trabalho e é a essas pessoas que agradeço.

Ao Doutor Walter Vieira, pela sua orientação, disponibilidade, pelas criticas, por todas as reuniões e revisões que fez ao relatório.

A todos os professores que durante estes três anos de mestrado me acompanharam, destacando os Professor Doutor Walter Vieira, Professor Doutor Nuno Datia e Professor Doutor Luis Morgado que me transmitiram conhecimentos que me foram úteis e continuaram a ser no futuro.

Ao meu colega de mestrado Mikael Malho que nestes três anos de mestrado sempre me acompanhou ao longo de todo o curso.

À minha entidade patronal Inditex, especialmente aos meus colegas Engenheiro Luis Pereia e Rafael Martins por todo apoio e motivação incondicional que me ajudou e muito durante este periodo.

Aos meus amigos, Francisco Marques, Sergio Guerra, Rodrigo Tomé, Filipe Pires, André Gomes, André Segundo e aos outros que sabem quem são, pela motivação e apoio e pela compreensão da minha falta de disponibilidade para muitos dos convites que me foram feitos.

À minha irmã Sandra Alberto, pelo carinho e amor que sempre demonstrou e também pela compreensão pelo pouco tempo que tive disponível durante estes três anos.

À minha mãe Ana Alberto e ao meu pai Carlos Alberto, pelo amor, carinho e ajuda que durante este periodo de menor disponibilidade da minha parte nunca me faltou.
Um agradecimento especial à minha restante famila por todo o apoio.

À minha namorada Susana Cardoso, pelo amor, pela compreensão da minha falta de tempo e por todo o apoio durante estes anos.

E por fim, um agradecimento ao meu cão Koda que muitas vezes me aqueceu os pés durante a noite enquanto trabalhava para este mestrado.

# Resumo

Cada vez mais os sistemas de informação empresariais (do inglês, enterprise information systems) tendem a ter uma complexidade elevada associada, já que os clientes pretendem sistemas robustos, interoperáveis, íntegros, escaláveis, com alto desempenho e alta disponibilidade. Para conseguir estas metas, os sistemas de informação empresariais utilizam muitas vezes acesso concorrente aos dados, possivelmente, distribuídos, em âmbito transaccional e podendo envolver fontes de dados heterogéneas, como, por exemplo, bases de dados relacionais e filas de mensagens.

Existem várias ferramentas proprietárias, por exemplo da Microsoft, que ajudam a minimizar esta complexidade, no entanto, podem colocar outras dificuldades como é o facto do sistema poder ficar dependente de determinada tecnologia ou o preço do sistema poder ter um custo muito elevado. Por outro lado, existem soluções *open source* que possibilitam o desenvolvimento de sistemas de informação empresariais, no entanto essas soluções, por vezes, são desenvolvidas por diferentes empresas o que obriga a uma complexidade extra (comparando com as soluções proprietárias) para integrar todo o sistema.

Este estudo visa comparar o esforço despendido no desenvolvimento de sistemas de informação empresariais utilizando tecnologias open source, nomeadamente JEE 8 e MYSQL, em relação às tecnologias Microsoft, nomeadamente WCF e SQL Server.

**Palavras-chave:** sistemas de informação empresariais, robustez, interoperabilidade, integridade, escalabilidade, alto desempenho, alta disponibilidade, open source

# Abstract

Increasingly, enterprise information systems tend to be associated with high complexity, since customers want robust, interoperable, scalable, high performance and high availability systems. To achieve these goals, enterprise information systems often use concurrent access to data, possibly distributed, in a transactional context and may involve heterogeneous data, such as relational databases and queues of message.

There are a number of proprietary tools, such as Microsoft, that help to minimize this complexity, however, may pose other difficulties such as the fact that the system can be dependent on a given technology or the price of the system can be very expensive. On the other hand, there are open source solutions that enable the development of information systems but these solutions are sometimes developed by different enterprises, which leads to extra complexity (compared with the proprietary solutions) to integrate the entire system.

This study aims at comparing the effort expended in the development of business information systems using open source technologies, namely JEE 8 and MYSQL, in relation to Microsoft technologies, namely WCF and SQL Server.

**Keywords:** business information systems, robustness, interoperability, integrity, scalability, high performance, high availability, open source

# Índice

**Índice** **xiii**

**Lista de Figuras** **xix**

**Lista de Tabelas** **xxiii**

**Listagens** **xxv**

**1 Introdução e Visão Geral** **1**

1.1 Introdução . . . . . 1

1.2 Objectivos . . . . . 2

1.3 Convenções . . . . . 2

1.4 Organização do documento . . . . . 3

**2 Sistemas de Informação Empresariais (SIE)** **5**

2.1 Medidas de qualidade de um SI . . . . . 5

2.1.1 Características de um SI com qualidade . . . . . 5

2.1.2 Medida de qualidade . . . . . 6

2.2 Principais conceitos presentes em SI Empresariais . . . . . 10

2.2.1 Arquitecturas de SI . . . . . 10

2.2.1.1 Arquitectura Monolítica . . . . . 11

2.2.1.2 Arquitectura de 2 camadas . . . . . 12

2.2.1.3 Arquitectura de 3 camadas . . . . . 13
2.2.1.4 Arquitectura a utilizar . . . . . 14
2.2.2 Bases de dados distribuídas (BDD) . . . . . 15
2.2.2.1 Tipos de BDD . . . . . 15
2.2.2.2 Estratégias para construção de uma BDD . . . . . 16
2.2.2.3 Fragmentação dos dados . . . . . 17
2.2.3 Replicação de dados . . . . . 18
2.2.3.1 Vantagens e desvantagens da replicação . . . . . 19
2.2.4 Topologias de replicação . . . . . 19
2.2.4.1 Master – slave(s) . . . . . 19
2.2.4.2 Master – Master . . . . . 21
2.2.5 Escalabilidade na camada de dados . . . . . 21
2.2.6 Controlo transaccional . . . . . 23
2.2.7 Desacopolamento na comunicação entre sistemas . . . . . 25

**3 Camada de dados (CD) 29**
3.1 Construção da CD . . . . . 29
3.2 Bases de dados distribuídas (BDD) . . . . . 30
3.2.1 Fragmentação em MYSQL vs SQLServer . . . . . 30
3.3 Controlo transaccional . . . . . 34
3.3.1 Implementação SQL Server vs MySQL . . . . . 34
3.4 BDD Replicação . . . . . 36
3.4.1 Implementação em MySQL vs SQL Server . . . . . 36

**4 Camada Aplicacional (CA) 47**
4.1 Organização da Camada Aplicacional . . . . . 47
4.2 Windows Communication Foundation (WCF) . . . . . 48
4.2.1 Organização do WCF . . . . . 48
4.2.2 Tipos de Instâncias . . . . . 51
4.2.2.1 Serviço Per-Call . . . . . 51

4.2.2.2 Serviço Per-Session . . . . . 52
4.2.2.3 Serviço Singleton . . . . . 53
4.2.3 Construção de um Serviço WCF . . . . . 53
4.2.3.1 Service Contracts . . . . . 53
4.2.3.2 Operation Contracts . . . . . 54
4.2.3.3 Data Contracts . . . . . 55
4.2.3.4 Serviço WCF . . . . . 56
4.2.3.5 Cliente de um Serviço WCF . . . . . 58
4.2.4 Transacções . . . . . 58
4.2.4.1 Gestão explícita de transacções . . . . . 59
4.2.4.2 Gestão de transacções distribuídas . . . . . 60
4.2.4.3 Propagação de transacções . . . . . 60
4.2.4.4 Transacções: Completar e Votar . . . . . 62
4.2.5 Gestão de concorrência . . . . . 63
4.2.6 Filas de mensagens - MSMQ . . . . . 65
4.2.6.1 Criar Serviço WCF com MSMQ . . . . . 66
4.2.6.2 Binding . . . . . 66
4.2.6.3 Iniciar serviço WCF com MSMQ . . . . . 68
4.2.6.4 Cliente de serviço MSMQ . . . . . 68
4.2.7 EntityFramework (EF) . . . . . 69
4.2.7.1 Contexto do EF . . . . . 69
4.2.7.2 Exemplo de utilização EF . . . . . 70
4.3 Java Enterprise Edition 8 (JEE) . . . . . 75
4.3.1 Arquitectura do JEE 8 . . . . . 76
4.3.2 Componentes . . . . . 76
4.3.3 Contentores . . . . . 77
4.3.4 Serviços . . . . . 77
4.3.5 Empacotamento . . . . . 78
4.3.6 Enterprise JavaBeans (EJB) . . . . . 80

4.3.6.1 Tipos de Enterprise JavaBeans (EJB) . . . . . 80
4.3.6.2 Estrutura de uma Session Bean . . . . . 81
4.3.6.3 EJB Stateless . . . . . 82
4.3.6.4 EJB Stateful . . . . . 84
4.3.6.5 EJB Singleton . . . . . 87
4.3.6.6 EJB Message-driven beans . . . . . 88
4.3.6.7 Estrutura da escrita de um Message-driven bean . 90
4.3.6.8 Concorrência . . . . . 94
4.3.6.9 EJB síncrono vs assíncronos . . . . . 96
4.3.6.10 Transacções . . . . . 97
4.3.6.11 Como utilizar EJB . . . . . 102
4.3.7 Java Persistence API (JPA) . . . . . 104
4.3.7.1 Anotações JPA . . . . . 104
4.3.7.2 Persistence Unit: . . . . . 106
4.3.7.3 JPA - Entity Manager . . . . . 108
4.3.7.4 Utilização EJB com JPA . . . . . 108

**5 Protótipo** **111**
5.1 Enquadramento . . . . . 111
5.2 Projecto . . . . . 111
5.2.1 Requisitos do sistema . . . . . 114
5.2.2 Camada de Dados (CD) . . . . . 115
5.2.2.1 Esquema Lógico Global . . . . . 115
5.2.2.2 Esquema de Fragmentação . . . . . 116
5.2.2.3 Esquema de distribuição . . . . . 117
5.2.2.4 Replicação de dados . . . . . 118
5.2.2.5 Implementação SQL Server . . . . . 122
5.2.2.6 Implementação MYSQL . . . . . 126
5.2.3 Camada Aplicacional (CA) . . . . . 130
5.2.3.1 Implementação com JEE . . . . . 130
5.2.3.2 Implementação com WCF . . . . . 144

**6 Resultados** **157**

6.1 Camada de Dados . . . . . 157

6.2 Camada Aplicacional . . . . . 158

**7 Conclusões** **161**

**Referências** **163**

**A Organização do CD** **i**

# Lista de Figuras


2.1 Niveis de disponibilidade . . . . . 7
2.2 ASI lógica vs ASI física [1] . . . . . 11
2.3 Arquitectura Monolítico . . . . . 11
2.4 Arquitectura de 2 camadas . . . . . 12
2.5 Arquitectura de 3 camadas . . . . . 13
2.6 BDD homogénea . . . . . 16
2.7 BDD heterogénea . . . . . 16
2.8 Fragmentação Horizontal . . . . . 17
2.9 Fragmentação Vertical . . . . . 18
2.10 Master – slave(s) . . . . . 20
2.11 Multi – Master . . . . . 21
2.12 Read scale-out com replicação [10] . . . . . 22
2.13 Sharding [10] . . . . . 22
2.14 Modelo X/OPEN [1] . . . . . 23
2.15 Protocolo two-phase commit [6] . . . . . 25
2.16 Publish-subscribe [1] . . . . . 26
2.17 Espaço virtual compartilhado . . . . . 27
2.18 Imagem de exemplo de um sistema com filas de mensagens . . . . 28

3.1 Visão global . . . . . 31

3.2 Funcionamento da replicação no MySQL [7] . . . . . 38

4.1 Camada aplicacional . . . . . 47
4.2 Comunicação entre Cliente e Serviço WCF [3] . . . . . 49
4.3 Tabela de comparação de Bindings . . . . . 51
4.4 Instância Per-call . . . . . 52
4.5 Instância Per-session . . . . . 52
4.6 Instância Singleton . . . . . 53
4.7 Item: ADO.NET Entity Data Model . . . . . 70
4.8 Gerar EDMX desde a BD . . . . . 71
4.9 Modelo carregado no EDMX . . . . . 71
4.10 Contentores JEE [2] . . . . . 76
4.11 Serviços oferecidos pelos contentores JEE [2] . . . . . 78
4.12 Arquivos nos contentores JEE [2] . . . . . 79
4.13 POJO vs EJB . . . . . 81
4.14 Interfaces EJB [2] . . . . . 82
4.15 Interação entre clientes e EJB Stateless [2] . . . . . 83
4.16 Interação entre clientes e EJB Stateful [2] . . . . . 85
4.17 Interação entre clientes e EJB Singleton [2] . . . . . 87
4.18 Arquitectura MOM [2] . . . . . 89
4.19 API Classic do JMS [2] . . . . . 90
4.20 JMS no Glassfish . . . . . 90
4.21 Diagrama de sequência de transacção . . . . . 100
4.22 Modelo EA - tabela Cliente . . . . . 104
4.23 Glassfish JDBC . . . . . 107

5.1 Arquitetura física . . . . . 112
5.2 Esquema lógico global . . . . . 115
5.3 Fragmentação produto . . . . . 116
5.4 Esquema lógico SGBD Encomendas . . . . . 117

5.5 Esquema lógico SGBD Gestão . . . . . 118
5.6 Replicação produto . . . . . 119
5.7 Replicação perguntas e respostas . . . . . 120
5.8 Replicação cliente . . . . . 121
5.9 Configuração para acesso remoto ao DTC . . . . . 122
5.10 Configuração para protocolo TCP/IP . . . . . 123
5.11 Replicação Publisher-Subscriber . . . . . 125
5.12 Replicação Peer-to-Peer . . . . . 125
5.13 Bases de dados e tabelas nos nós MYSQL . . . . . 126
5.14 Servidores da solução . . . . . 130
5.15 Separação da Camada Aplicacional . . . . . 130
5.16 Classes que representam entidades . . . . . 131
5.17 Connection Pools Glassfish . . . . . 133
5.18 Configuração Connection Pools Glassfish . . . . . 133
5.19 JDBC Resource Glassfish . . . . . 133
5.20 Enterprise Java Beans . . . . . 134
5.21 Comunicação assíncrona . . . . . 138
5.22 Configuração JMS - Servidor Encomendas . . . . . 139
5.23 Configuração JMS - Servidor Gestão e Notificação . . . . . 139
5.24 Configuração JMS - Destination Resource e Connection Factories . 140
5.25 Menu aplicação Encomendas e Gestão . . . . . 143
5.26 Servidores da solução . . . . . 145
5.27 Interfaces e operações dos Serviços WCF . . . . . 146
5.28 Fila de encomendas . . . . . 148
5.29 Diagrama de sequência operação encomendar . . . . . 149
5.30 Fila de notificação . . . . . 149
5.31 Activar MSMQ . . . . . 152
5.32 Entity Framework vista grafica . . . . . 153
5.33 Entity Framework propriedades . . . . . 153
5.34 Aplicação Gestão e Encomendas . . . . . 155

# Lista de Tabelas

5.1 Requisitos do sistema . . . . . 114

# Listagem


3.1 addlinkedserver para acesso ao Nó2 . . . . . 31
3.2 addlinkedserver para acesso ao Nó1 . . . . . 31
3.3 SYNONYM para acesso ao Nó2 . . . . . 32
3.4 SYNONYM para acesso ao Nó1 . . . . . 32
3.5 Vista NomeLocal . . . . . 32
3.6 Permissões acesso remoto . . . . . 33
3.7 Criar tabelas federadas . . . . . 33
3.8 Procedimento armazenado para inserir Pessoa . . . . . 35
3.9 Criar user para replicação . . . . . 40
3.10 Backup BD Master . . . . . 40
3.11 Restaurar backup na BD Salve . . . . . 41
3.12 Indicar ao slave quem é o master . . . . . 41
3.13 Criar user para replicação . . . . . 42
3.14 Criar user para replicação . . . . . 42
3.15 Info log Master1 . . . . . 42
3.16 Indicar ao slave quem é o master . . . . . 43
3.17 Info log Master2 . . . . . 43
3.18 Indicar ao slave quem é o master . . . . . 43
3.19 Indicar ao slave o Master1 e o seu canal . . . . . 44
3.20 Indicar ao slave o Master2 e o seu canal . . . . . 44

3.21 Iniciar replicação para todos os Masters . . . . . 45
4.1 Definir dois endpoints para um serviço . . . . . 51
4.2 Exemplo de um Service Contract . . . . . 54
4.3 Exemplo de Operation Contract . . . . . 54
4.4 Exemplo de Data Contract . . . . . 55
4.5 Exemplo da implementação de um serviço WCF . . . . . 57
4.6 Cliente de um serviço WCF . . . . . 58
4.7 Gerir transacção de forma explícita . . . . . 59
4.8 Activar propagação de transacções programaticamente . . . . . 60
4.9 Activar propagação de transacções ficheiro de configuração . . . . 61
4.10 Exemplo: Operação permite propagação . . . . . 61
4.11 Exemplo: Operação não permite propagação . . . . . 62
4.12 Exemplo: Operação obriga a ter propagação . . . . . 62
4.13 Exemplo: Votar automaticamente uma transacção . . . . . 63
4.14 Exemplo: Modo de concorrência Single . . . . . 64
4.15 Exemplo: Modo de concorrência Multiple . . . . . 65
4.16 Contrato para serviço WCF com MSMQ . . . . . 66
4.17 Contrato para serviço WCF com MSMQ . . . . . 66
4.18 Definir endpoint para serviço com MSMQ . . . . . 68
4.19 Iniciar serviço MSMQ . . . . . 68
4.20 Exemplo: Utilização de serviço WCF com MSMQ . . . . . 69
4.21 Exemplo adicionar com EF . . . . . 72
4.22 Exemplo actualizar com EF . . . . . 73
4.23 Exemplo obter recurso com EF . . . . . 74
4.24 Exemplo apagar com EF . . . . . 75
4.25 Exemplo persistência de produto . . . . . 82
4.26 Exemplo persistência de produto . . . . . 83
4.27 Implementação de um EJB Stateless . . . . . 84
4.28 Interação de um cliente com um EJB Stateful . . . . . 85

4.29 Implementação de um EJB Stateless . . . . . 86
4.30 Implementação de um EJB Singleton . . . . . 88
4.31 Produtor de mensagens assíncronas . . . . . 91
4.32 Message-driven bean (MBD) . . . . . 93
4.33 Controlo ao nível do contentor EJB Singleton . . . . . 94
4.34 Controlo ao nível do componente EJB Singleton . . . . . 95
4.35 EJB síncrono caption . . . . . 96
4.36 EJB assíncrono . . . . . 97
4.37 EJB assíncrono Future<?> . . . . . 97
4.38 EJB com tipo de transacção REQUIRED . . . . . 99
4.39 EJB transaccional, rollback explicito . . . . . 101
4.40 EJB transaccional, rollback explicito . . . . . 102
4.41 Definição de um simples EJB Stateless . . . . . 103
4.42 Utilização de um EJB com @EJB . . . . . 103
4.43 Utilização de um EJB com @Inject . . . . . 103
4.44 Utilização de um EJB com JINI . . . . . 103
4.45 JPA - Entidade Cliente . . . . . 105
4.46 Exemplo de PersistenceUnit.XML . . . . . 107
4.47 ânciar EntityManager . . . . . 108
4.48 EJB com JPA . . . . . 108
5.1 Criação de Linked Servers e Synonyms no nó Encomendas . . . . . 123
5.2 Criação de Linked Servers e Synonyms no nó Gestão . . . . . 123
5.3 Procedimento armazendo com transacções distribuídas . . . . . 124
5.4 Configuração para acessos remotos . . . . . 127
5.5 Definir identificador do servidor Encomendas . . . . . 127
5.6 Definir identificador do servidor Gestão . . . . . 128
5.7 Criar utilizador com permissão para fazer a replicação . . . . . 128
5.8 Obter nome do log file e posição no Nó Gestão . . . . . 128

5.9 Iniciar replicação no Nó Encomendas . . . . . 128
5.10 Obter nome do log file e posição no Nó Encomendas . . . . . 129
5.11 Iniciar replicação no Nó Gestão . . . . . 129
5.12 Ficheiro persistence.XML . . . . . 132
5.13 EJB ClienteSessionBean . . . . . 135
5.14 Método save de ClienteDAO . . . . . 137
5.15 MDB Envio de mensagem para a fila . . . . . 141
5.16 MDB Recepção de mensagem da fila . . . . . 142
5.17 Obter EJB com JINI . . . . . 144
5.18 Operação encomendar produto . . . . . 147
5.19 Contrato IServiceGestaoMSMQ . . . . . 150
5.20 Implementação do Serviço ServiceGestaoMSMQ . . . . . 151
5.21 Configuração do ABC . . . . . 152
5.22 Exemplo de uso do EF . . . . . 154

# 1

# Introdução e Visão Geral

## 1.1 Introdução

Os sistemas de informação empresariais (do inglês, enterprise information systems) caracterizam-se pela existência de acesso concorrente aos dados, possivelmente, distribuídos, em âmbito transacional e podendo envolver fontes de dados heterogéneas, como, por exemplo, bases de dados relacionais e filas de mensagens.

As organizações reconhecem que os sistemas de informação empresariais são estratégicos e fonte de vantagens competitivas importantes. O desenvolvimento deste tipo de sistemas não é tarefa fácil, dada a complexidade inerente ao seu desenvolvimento, sendo frequente o uso de plataformas computacionais que facilitem esta tarefa, destacando-se nestas a plataforma java EE, o Spring (java) e o Windows Comunication Foundation (WCF).

Por outro lado, nestes sistemas, habitualmente, são exigidos níveis de escalabilidade e disponibilidade elevados, o que obriga à adoção das tecnologias adequadas para os concretizar.

## 1.2 Objectivos

Este trabalho tem como objectivo a comparação do esforço despendido no desenvolvimento de sistemas de informação empresariais utilizando tecnologias Open Source (JEE e MYSQL) e Comerciais (WCF e SQL Server). Para isso são apresentados exemplos que demonstram a forma como é possivel resolver determinados problemas em ambas as tecnologias. Támbem é apresentado um projecto onde se define os requisitos funcionais e operacionais de um sistema que serão usados para o desenvolvimento de protótipos usando cada uma das plataformas. No final, será feita uma análise comparativa das duas plataformas no que respeita ao esforço despendido para a concretização dos vários requisitos.

## 1.3 Convenções

Ao longo deste documento, seguiu-se um conjunto de convenções que facilitam a leitura e entendimento do mesmo por parte do leitor.

### Código fonte:

Troços de código são apresentados numa caixa de fundo branco, onde cada uma das linhas se encontra numerada, como se exemplifica a seguir:

```
1 Codigo fonte
```

### Citações:

As citações são apresentadas dentro de uma caixa com o fundo cinza, como se exemplifica a seguir:

> Citações

### Notas importantes:

Sempre que se pretende destacar algo, é colocado o texto a **negrito** ou sublinhado.

### Palavras em inglês:

Embora neste documento se tenha traduzido a maioria dos termos para português, alguma palavras foram mantidas em inglês por se considerar que não seria

adquado traduzir a mesma. Nessas situações as palavras foram colocadas em itálico. (Exemplo: *Open Source, Stateless, Statefull...*)

## 1.4 Organização do documento

Este documento é composto por sete capítulos e foi escrito por forma a que o leitor o leia de forma sequencial.

### Capítulo 1. Introdução e Visão Geral

No primeiro capítulo, o actual, apresento o desafio que está presente no desenvolvimento de sistemas de informação empresariais, explico os objectivos que se pretendem alcançar com a realização deste projecto e mostro ao leitor a forma como foi organizado o documento.

### Capítulo 2. Sistemas de Informação Empresariais

No segundo capítulo, são apresentados os principais conceitos a ter em conta no desenvolvimento deste tipo de sistemas. Este capítulo é meramente teórico, e pretende elucidar o leitor dos conceitos que serão utilizados nos capítulos seguintes, altura em que os conceitos são abordados numa vertente mais prática.

### Capítulo 3. Camada de dados (CD)

O terceiro capítulo, mostra e faz a comparação entre a implementação de soluções para conceitos a ter em conta no desenvolvimento da camada de dados, nas tecnologias MYSQL e SQL Server.

### Capítulo 4. Camada de aplicacional (CA)

O quarto capítulo, mostra e faz a comparação entre a implementação de soluções para conceitos a ter em conta no desenvolvimento da camada aplicacional, nas tecnologias JEE e WCF.

### Capítulo 5. Protótipo

No quinto capítulo, apresento um desafio com requisitos funcionais e operacionais e demonstro como o resolver com tecnologias Open Source (JEE e MYSQL) e Comercial (WCF e SQL Server).

### Capítulo 6. Resultados

Neste capítulo apresento os resultados das comparações obtidas durante o desenvolvimento do projecto.

## Capítulo 7. Conclusões

No sétimo e último capítulo, apresento as conclusões retiradas após a realização deste projecto.

# 2

# Sistemas de Informação Empresariais (SIE)

## 2.1 Medidas de qualidade de um SI

No desenvolvimento de Sistemas de Informação Empresariais, tal como no desenvolvimento de qualquer outro SI, é necessário garantir que o produto final cumpre um conjunto de requisitos para que seja classificado como um SI com qualidade.

### 2.1.1 Características de um SI com qualidade

Um SI com qualidade é aquele que consegue cumprir os requisitos funcionais e não-funcionais de um projecto, conforme se descreve a seguir:

- **Requisitos funcionais** – descrevem as funcionalidades e o comportamento do sistema consoante determinados estimulos em tempo de execução, ou seja, descreve o que o sistema tem que fazer a cada ação de um utilizador ou outro sistema.

  Os requisitos funcionais são estabelecidos através do conhecimento passado pelos utilizadores em relação ao processo de negócio.

- **Requisitos não-funcionais** – dizem respeito à qualidade que o SI deve oferecer, como por exemplo, desempenho, escalabilidade, segurança, entre outros.

Os requisitos não-funcionais têm elevada importância para a construção de um SI de qualidade, sendo fundamental que o arquitecto do SI conheça os objectivos a serem concretizados bem como as ferramentas disponíveis para os concretizar.

### 2.1.2 Medida de qualidade

A qualidade de um SI pode ser avaliada seguindo o padrão ISO/IEC 9126-1:2001 [5] que é um padrão internacional para avaliar software. Este standard apresenta as características mais relevantes na avaliação de um sistema de qualidade:

- **Funcionalidade** – Mede a capacidade de um software disponibilizar funcionalidades que satisfaçam o utilizador nas suas necessidades declaradas e implícitas, dentro de um determinado contexto de utilização. Para que um SI tenha boa funcionalidade são analisadas outras características como:
  - **Adequação**, pode ser vista como a medida de satisfação ou não dos requisitos funcionais do sistema.
  - **Precisão**, mede a capacidade que o SI tem em fornecer resultados precisos ou com precisão dentro do especificado pelo cliente.
  - **Interoperabilidade**, mede a capacidade do SI interagir com outros SI especificados.
  - **Segurança**, indica qual o grau de protecção do SI contra acessos indevidos.
- **Confiabilidade** - Um SI diz-se confiável quando é capaz de manter o nível de desempenho acordado sob determinadas circunstâncias. Por exemplo, o sistema deve estar disponível 98% do ano para 40 mil utilizadores em simultâneo, sendo os outros 2% reservados para manutenção. Também pode medir o tempo que o sistema demora a responder às funcionalidades do sistema. A confiabilidade pode ainda ser dividida nas seguintes características:
  - **Maturidade**, mede a capacidade que o SI tem em evitar falhas ocorridas por defeitos no software.

- **Tolerância a faltas**, mede a robustez de um SI, ou seja, a capacidade que o SI tem em funcionar, mesmo que de forma restrita, no caso de um servidor parar, haver problemas nos discos rígidos, inserção ou leitura de dados corrompidos, etc. Um dos aspectos a ter em conta para um SI tolerante a falhas é evitar pontos únicos de falha, quer nas comunicações, hardware ou software.
- **Recuperabilidade**, mede a capacidade que o sistema tem em voltar ao nível de desempenho anterior a falhas ou comportamento imprevisto do utilizador, software ou hardware e recuperar os dados afectados, caso existam.
- **Disponibilidade**, mede a capacidade de um sistema estar pronto para executar uma funcionalidade num dado momento, sob condições especificas de uso. Externamente, a disponibilidade pode ser vista como o tempo total durante o qual o produto de software está disponível.
  A disponibilidade é, portanto, a combinação de maturidade (a qual controla a frequência de falhas), pausas planeadas, tolerância a falhas e recuperabilidade (a qual controla o período de tempo inativo após cada falha).
  A Disponibilidade é uma medida de qualidade bastante importante em SIE complexos já que existe uma elevada exigência por parte das grandes organizações no sentido de ter sistemas sem quebras de serviço.

  Na tabela seguinte, são ilustrados diferentes níveis de medida dos sistemas com alta disponibilidade:

| Niveis de disponibilidade | | | |
|---|---|---|---|
| **Nivel de disponibilidade** | **Percentagem de Uptime** | **Downtime por ano** | **Downtime por dia** |
| 1 Nove | 90% | 36.5 dias | 2.4 horas |
| 2 Noves | 99% | 3.65 dias | 14 min |
| 3 Noves | 99.9% | 8.76 horas | 86 seg |
| 4 Noves | 99.99% | 52.6 min | 8.6 seg |
| 5 Noves | 99.999% | 5.25 min | 0.86 seg |
| 6 Noves | 99.9999% | 31.5 seg | 8.6 mseg |

Figura 2.1: Niveis de disponibilidade

Na implementação de sistemas empresariais com elevada disponibilidade devem ser seguidas as seguintes etapas:

    - Definir o tempo máximo de indisponibilidade e perdas aceitáveis;
    - Definir a melhor arquitectura e topologia de disponibilidade para este contexto;
    - Implementar a redundânica dos componentes;
    - Implementar a monitorização;
    - Implementar os procedimentos operacionais de contingências;

- **Usabilidade** – mede a facilidade com que o utilizador executa alguma funcionalidade do sistema. A medida da usabilidade está directamente ligada com as características abaixo indicadas:

  – **Compreensibilidade**, mede a capacidade que o utilizador tem em entender o sistema. A qualidade de conceitos que o utilizador tem de saber ou a qualidade ou quantidade de documentação existente para o sistema são factores a ter em conta para o utilizador decidir se o SI serve para ele ou não.

  – **Facilidade de aprendizagem**, mede a o esforço que o utilizador tem de despender para aprender a utilizar o sistema. Esse esforço pode ser medido pela quantidade de conceitos ou operações que o utilizador precisa aprender para fazer com que o SI funcione.

  – **Operabilidade**, mede a facilidade que o SI oferece ao utilizador para realizar as operações.

- **Eficiência** - A eficiência é uma das medidas de qualidade mais procuradas durante o desenvolvimento de SI, já que é uma das medidas mais entendidas pelos utilizadores. Quando queremos medir eficiência, medimos basicamente duas características:

  – **Desempenho**, pode ser visto como a capacidade que o sistema tem em alcançar a resposta dentro do período de tempo estipulado. Esta medida tem de ter em conta as condições em que o sistema está a funcionar, ou seja, um SI que está a ser utilizado por 20 mil utilizadores, não deve ser comparado com um sistema que está a ser utilizado apenas por mil utilizadores.

  – **Escalabilidade**, mede a facilidade de expandir os recursos de um sistema sempre que se justifique.
  Existem duas formas possíveis de fazer escalar um sistema, escalar de forma vertical ou de forma horizontal.

    * **Vertical** - Um SI é escalado de forma vertical quando se aumenta os recursos de hardware de um servidor enquanto for possível ou se adquire um servidor com maior capacidade.

      Também se deve ter em conta que o disco continua a ser único e, portanto impõe limites em termos do número de operações de I/O por unidade de tempo.
      Outra limitação desta solução é que, pelo facto de termos uma única máquina, temos um único ponto de falha, pelo que a disponibilidade fica comprometida.
    * **Horizontal** - Um SI é escalado de forma horizontal sempre que se acrescenta um novo nó.
      É possível por exemplo configurar centenas de pequenos computadores num cluster para obter poder computacional agregado que geralmente excede ao de computadores baseados num único processador tradicional.

  – **Elasticidade**, é um conceito que está presente em Cloud Computing, onde o cliente paga pelos recursos que usa. Assim sendo, os fornecedores destes serviços oferecem a possibilidade de ajustar os recursos do sistema em tempo real, quer fornecendo mais recursos ou libertando recursos quando estes não são necessários.

- **Manutenibilidade** – esta qualidade é muitas vezes ignorada pelos utilizadores mas é de elevada importância para os desenvolvedores. Algumas características que caracterizam a manutenibilidade:

  – **analisabilidade**, é o grau de esforço que temos de aplicar para procurar por deficiências no SI ou por partes que devem ser alteradas por algum motivo.

  – **modificabilidade**, mede a facilidade de realizar alterações de implementação no sistema. Esta característica está relacionada com a forma como o arquitecto de software desenhou o SI, como são os níveis de coesão e acoplamento e complexidade.

  – **testabilidade**, representa a capacidade de se testar o sistema modificado, tanto quanto a novas funcionalidades quanto a funcionalidades não afectadas directamente pela modificação;

- **Portabilidade** – mede a capacidade de transferir um SI de um ambiente para outro. A medida da portabilidade tem em conta as seguintes características:

  - **adaptabilidade**, mede a capacidade do software ser migrado para outro ambiente sem precisar de alterações além das previstas.
  - **Capacidade para ser Instalado**, mede a facilidade com que se pode instalar o SI num novo ambiente.
  - **Coexistência**, mede a capacidade com que um sistema funciona partilhando recursos num mesmo ambiente com outros sistemas.

## 2.2 Principais conceitos presentes em SI Empresariais

### 2.2.1 Arquitecturas de SI

Neste ponto serão abordadas algumas arquitecturas de sistemas de informação(ASI), bem como os aspectos a ter em conta na escolha da arquitectura a utilizar.
A ASI tem influência directa na qualidade do SI. Idealmente, o Arquitecto procura uma solução que garanta um sistema que cumpra as características presentes no ISO/IEC 9126-1:2001 que foi abordado no ponto 2.1.

**As ASI têm associadas duas componentes:**

- **Arquitectura lógica:** onde estão presentes as funções do SI e onde se encontra a lógica do fluxo de informação bem como a forma como os dados se encontram organizados. Idealmente a arquitectura lógica deverá ser independente das tecnologias e Hardware.

- **Arquitectura física:** representa os componentes infra-estruturais que estão presentes no SI. Será sobre a arquitectura física que serão implementados os componentes da arquitectura lógica.

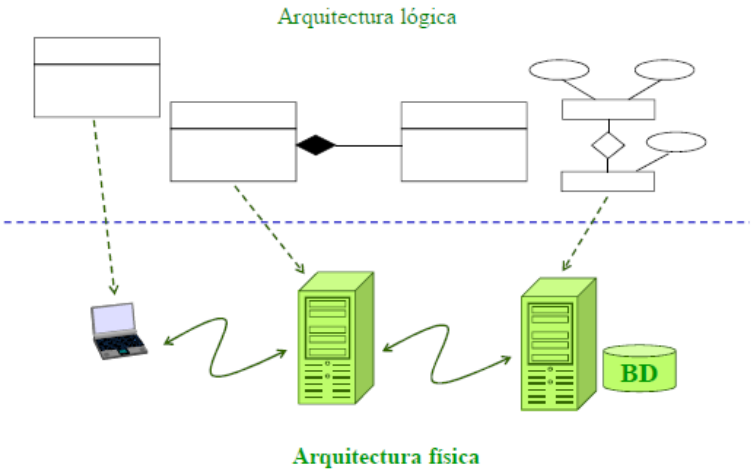


Figura 2.2: ASI lógica vs ASI física [1]

ASI física que representa um nó do SI, poderá conter uma ou mais camadas de ASI lógicas.

#### 2.2.1.1 Arquitectura Monolítica

O sistema de informação monolítico é um sistema que foi concebido para usar uma única máquina, como é possível ver na figura 2.3, tanto a apresentação, como lógica aplicacional e dados estão todos no mesmo computador.

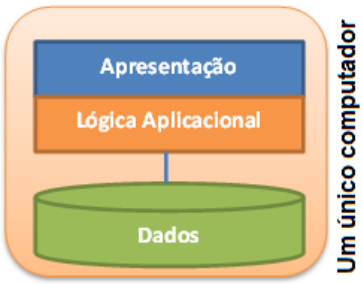


Figura 2.3: Arquitectura Monolítico

Características de um SI monolítico:

- Fraca escalabilidade – utiliza os recursos de uma única máquina;
- Dificuldade de manutenção – sistemas pouco modulares;
- Fraca robustez – utiliza os recursos de uma única máquina;
- Desempenho fraco – utiliza os recursos de uma única máquina;

#### 2.2.1.2 Arquitectura de 2 camadas

Um sistema de 2 camadas é um sistema que foi desenvolvido para ser utilizado em duas máquinas distintas (Cliente e Servidor), sendo que existem duas variantes estruturais deste modelo.

Na figura 2.4 é possível ver as duas variantes estruturais da arquitectura de 2 camadas. Na imagem da esquerda temos a variante "cliente gordo" e na direita a variante "cliente magro".

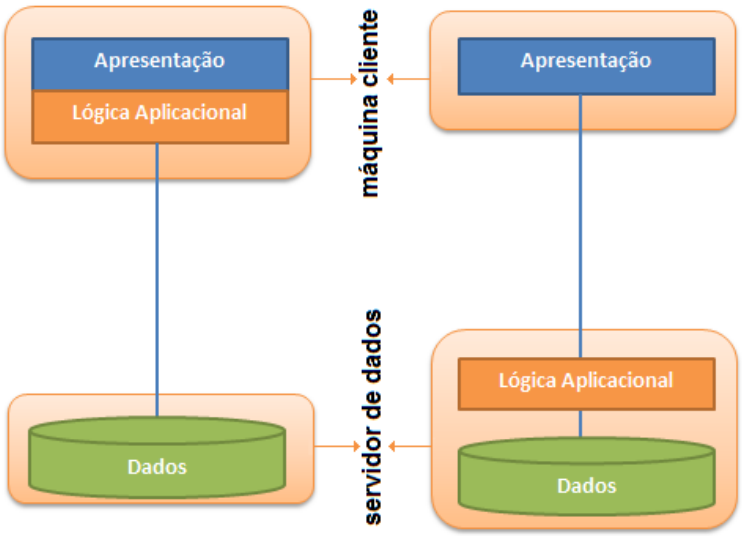


Figura 2.4: Arquitectura de 2 camadas

Características de um SI de 2 camadas com cliente gordo:

- Fraca escalabilidade - número de ligações ao servidor limitado, um único servidor;
- Dificuldade de manutenção - necessidade de alteração da lógica aplicacional em todos os clientes;
- Integridade comprometida - lógica aplicacional no cliente;
- Fraca reutilização - não é modular, tem dependência da organização dos dados;
- Desempenho fraco - grande volume de dados através da rede, nº limitado de ligações ao servidor;

Características de um SI de 2 camadas com cliente magro:

- Escalabilidade fraca - um servidor por aplicação, número de ligações ao servidor limitado;
- Mais fácil manutenção - alterar Lógica Aplicacional implica apenas alterações no servidor;
- Integridade melhorada - lógica aplicacional no servidor;
- Reutilização melhorada - procedimentos partilhados por várias aplicações;
- Desempenho melhorado - menor volume de dados através da rede e processamentos optimizados no servidor mas pode degradar-se o desempenho global;

#### 2.2.1.3 Arquitectura de 3 camadas

A arquitectura de 3 camadas será uma boa opção quando se pretende desenvolver sistemas de informação complexos. Com esta arquitectura temos uma separação bem definida das camadas de apresentação, lógica aplicacional e das bases de dados.

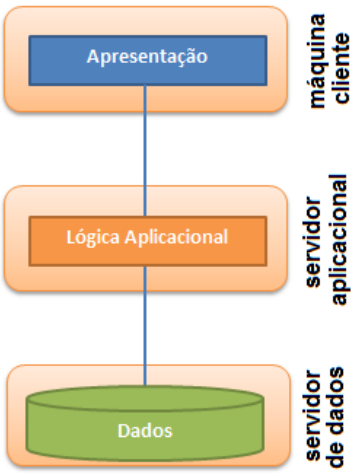


Figura 2.5: Arquitectura de 3 camadas

Características de um SI com arquitectura de 3 camadas:

- Menor dependência entre elementos – Programar para a interface, ou seja, se forem feitas alterações numa das camadas, se a interface se mantiver inalterada, será transparente para as outras camadas;

- Separação de responsabilidades – permite especializações nas equipas de desenvolvimento;
- Boa escalabilidade - é possível acrescentar-se servidores aplicacionais, menor número de ligações aos servidores de dados, multiplexagem de ligações aos servidores de dados, balanceamento de carga;
- Mais fácil manutenção – alterar uma camada sem necessitar de alterar todas as camadas;
- Integridade melhorada - lógica aplicacional nos servidores, logo os clientes não acedem directamente aos dados;
- Reutilização melhorada - encoraja a estruturação modular;
- Desempenho melhorado - menor volume de dados para os clientes, partilha de várias máquinas mais rápidas, balanceamento de carga;

#### 2.2.1.4 Arquitectura a utilizar

Ao serem analisados os prós e contras de cada uma das arquitecturas anteriormente enunciadas não é possível afirmar que existe uma arquitectura ideal para todos os projectos. Será sempre necessário analisar o contexto e a complexidade do projecto para optar por uma arquitectura.
Para projectos demasiado simples, provavelmente a melhor solução será uma arquitectura física monolítica. Quando se está perante um sistema de informação complexo, por exemplo, um sistema de informação empresarial poderá fazer sentido utilizar uma arquitectura de 3 camadas para tirar partido de todas as vantagens anteriormente enunciadas.

Tendo em conta que o alvo deste trabalho são os SI Empresariais Complexos, a arquitectura que será utilizada será a Arquitectura de 3 camadas.

### 2.2.2 Bases de dados distribuídas (BDD)

Bases de dados distribuídas (BDD) são uma colecção de várias bases de dados logicamente inter-relacionadas, distribuídas por uma rede de computadores. Existem dois tipos de bases de dados distribuídas, as homogéneas e as heterogéneas. Homogéneas quando existe software idêntico em todos os nós, e os nós estão conscientes da existência uns dos outros. Heterogéneas quando o software presente nos nós é diferente.
Numa base de dados distribuída os dados estão fragmentados e esses fragmentos podem ou não estar replicados.
Na fragmentação, os dados estão divididos ao longo do sistema, ou seja, em cada nó existe uma base de dados com informação diferente se olharmos de uma forma local, mas se agregarmos a informação que se encontra fraccionada nos diferentes nós iremos obter uma visão global do sistema. Quando um fragmento se encontra replicado, existe uma cópia desse fragmento em vários nós.

**Algumas das motivações que levam os SIE a utilizar BDD são:**

- **Natureza distribuída das aplicações** - É frequente as empresas terem instalações em vários locais.
- **Aumento da robustez e disponibilidade** - Falhas com impacto mais localizado.
  Por exemplo, uma empresa com 100 lojas, se houver um problema na sede, não deverá obrigar a que o negócio pare nas 100 lojas.
- **Aumento do desempenho** - BD menores em cada nó, melhoram o desempenho nos acessos locais.
- **Autonomia de processos locais** - As operações que podem ser realizadas num nó não dependem de outros nós.

#### 2.2.2.1 Tipos de BDD

**BDD Homogéneas**, será classificado como BDD homogénea quando temos em todos os nós do nosso sistema distribuído o mesmo software instalado. Na figura 2.6 é ilustrada uma BDD homogenia com o mesmo software em todos os nós, no caso MYSQL.

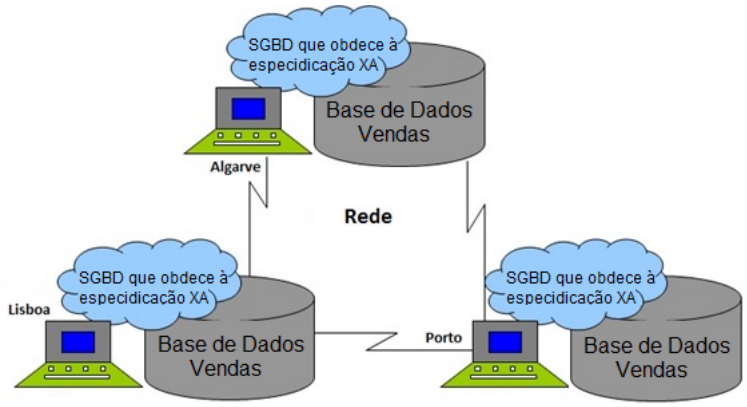


Figura 2.6: BDD homogénea

**BDD Heterogéneas**, será classificado como BDD heterogénea quando estamos perante um sistema distribuído com dois ou mais nós com software diferente, conforme ilustrado na figura 2.7 em baixo, onde têm diferente software e SGBD instalado em cada nó.

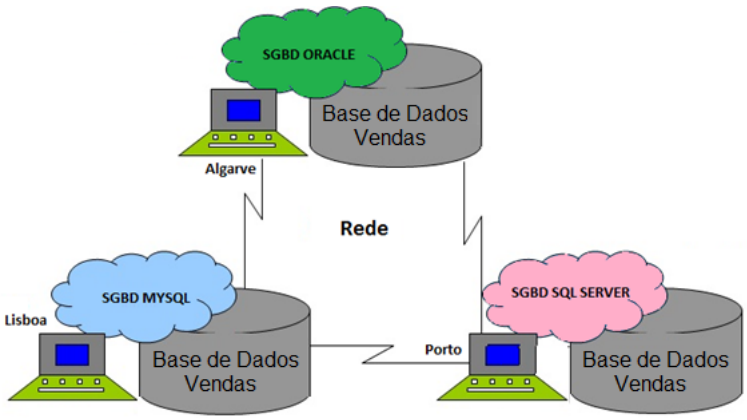


Figura 2.7: BDD heterogénea

#### 2.2.2.2 Estratégias para construção de uma BDD

Para a construção de uma BDD existem duas estratégias a seguir, Top-Down ou Bottom-Up.

- **TOP DOWN** - Esta estratégia é normalmente utilizada em BDD homogéneas. Inicialmente é feito o desenho do esquema lógico global, depois, desenvolvem-se os esquema de fragmentação e distribuição e por fim são construídos os esquemas físicos locais.

- **BOTTOM UP** - Esta estratégia é normalmente utilizada quando se pretende juntar bases de dados já existentes, colocando todas num único esquema global. Nestes casos, é necessário analisar os esquemas lógicos e fisicos de cada uma das bases de dados existentes para construir o esquema lógico global e com isso definir as vistas globais para os utilizadores.

#### 2.2.2.3 Fragmentação dos dados

A fragmentação dos dados pelos diferentes nós de uma BBD pode ser feita através de fragmentação vertical, horizontal ou mista.

**Fragmentação Horizontal**, na fragmentação horizontal, cada tuplo de uma tabela R é atribuído a um único fragmento. Sendo que a união de todos os tuplos de todos os fragmentos tem como resultado a tabela R "centralizada".

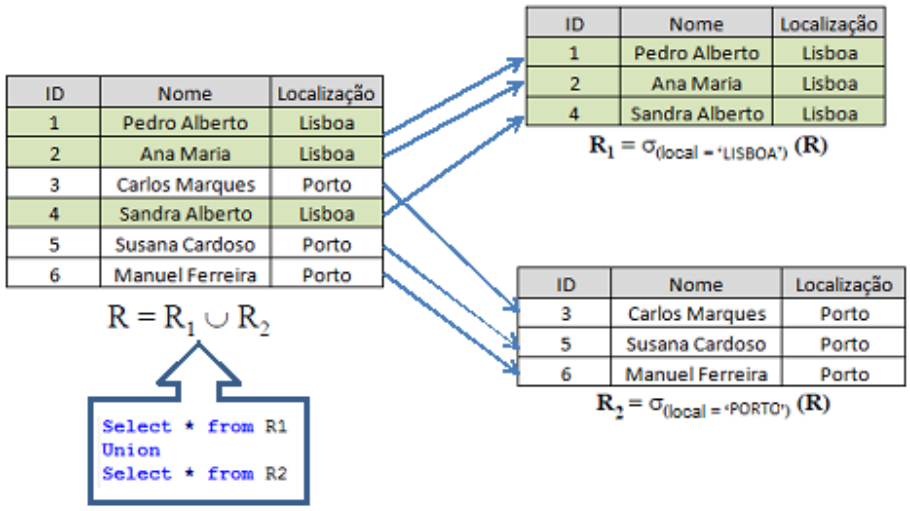


Figura 2.8: Fragmentação Horizontal

**Fragmentação Vertical**, na fragmentação vertical, a tabela R será dividida em dois ou mais esquemas mais pequenos de R, sendo que todos esses esquemas mais pequenos têm que conter obrigatoriamente uma chave comum. A junção de todos os esquemas mais pequenos através da chave, irá origina a tabela R "centralizada".

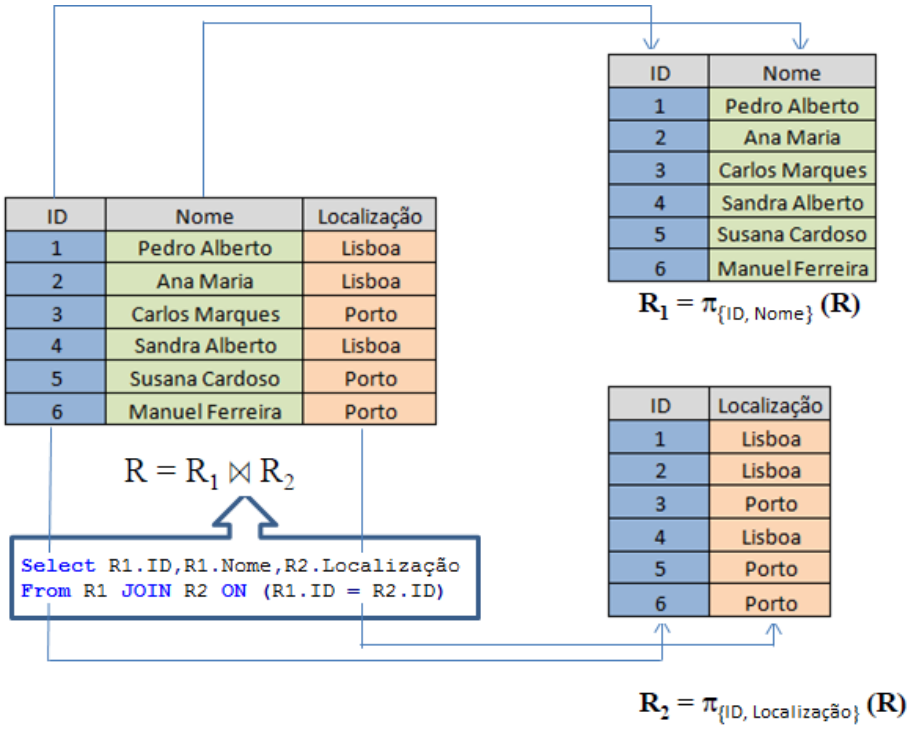


Figura 2.9: Fragmentação Vertical

### 2.2.3 Replicação de dados

A replicação de dados é bastante utilizada em SIE complexos. A replicação pode ajudar a resolver problemas de escalabilidade e disponibilidade bem como evitar a perda de dados em caso de catástrofes.
Existem dois tipos de replicação:

- **Assíncrona (ou "lazy replication")** - As réplicas podem apresentar valores diferentes durante períodos de tempo mais ou menos longos.

- **Síncrona (ou "eager replication")** - Todas as réplicas apresentam os mesmos valores em qualquer momento. (actualizações sincronizadas - transacções)

Também é possível configurar o que será replicado, podendo haver replicações totais onde os recursos são replicados na totalidade para os restantes nós ou poderá existir uma replicação parcial onde apenas parte dos recursos são replicados para alguns nós.

#### 2.2.3.1 Vantagens e desvantagens da replicação

**As principais vantagens da utiliação da replicação são:**

- **Disponibilidade:** Pelo menos para leitura o sistema continua a funcionar desde que pelo menos um dos nós esteja operacional.
- **Desempenho:** É melhorado porque uma interrogação pode ser resolvida localmente sem necessitar de aceder a outro nó.
- **Segurança:** A probabilidade de perda de dados é reduzida em caso de catastrofe, já que os mesmo se encontram em mais nós.
- **Autonomia local:** Alguns dos processamentos são realizados localmente, sem necessitar de acessos remotos.

**As principais desvantagens da utiliação da replicação são:**

- **Complexidade:** Aumenta a complexidade no que diz respeito à garantia de consistência entre as várias replicas sempre que existem actualizações.
- **Implantação mais cara:** Pode ser necessário adquirir hardware para implantar estas soluções.
- **Tráfego na rede:** Aumenta o tráfego de dados na rede.

### 2.2.4 Topologias de replicação

Nesta secção serão apresentadas as principais topologias de replicação bem como as vantagens e desvantagens de cada uma das topologias. Para a escolha da topologia mais indicada para um sistema, é necessário analisar as vantagens e desvantagens de cada topologia para não degradar o sistema.

#### 2.2.4.1 Master – slave(s)

Na topologia Master-Salve(s) as escritas são sempre feitas no servidor Master e a replicação é feita de forma sincrona ou assincrona, dependendo da configuração,

para o(s) servidor(es) Salve(s). É uma topologia muito útil quando temos a maior parte dos acessos ao sistema para leituras, em vez de escritas.

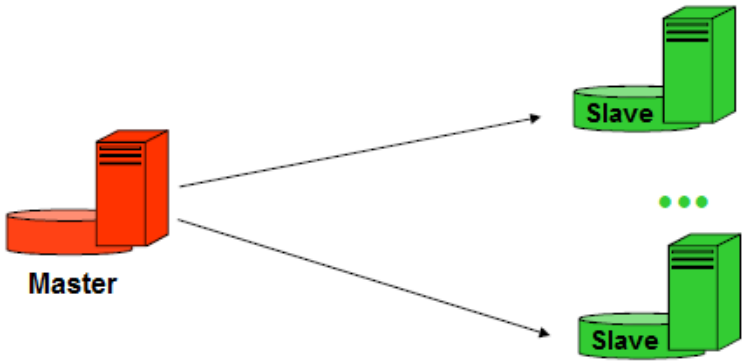


Figura 2.10: Master – slave(s)

**Vantagens**

- As leituras podem ser feitas sempre nos servidores Slaves, sem afectar o servidor Master;
- Os backups de parte ou totalidade do servidor master, podem realizar-se sem impacto no desempenho do servidor Master;
- Os servidores Slave podem ser desconectados e ligados sem quebra do serviço (só no caso assíncrono);

**Desvantagens**

- Ponto único de falhas no servidor Master;
- Em caso de falha do servidor Master é possível existir tempo de inatividade e possivelmente perda de dados;
- Todas as escritas têm de ser feitas no servidor Master, sendo difícil escalar para escritas;

#### 2.2.4.2 Master – Master

A topologia multi-master permite que qualquer servidor aceite instruções de leituras e escritas. É uma topologia muito útil quando existe a necessidade de todos os servidores efetuarem a leitura e escrita dos dados.

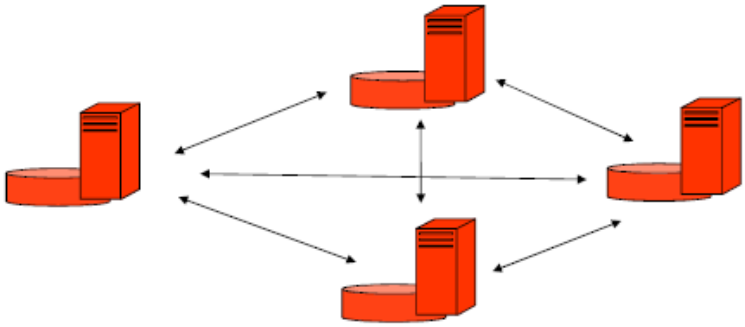

Figura 2.11: Multi – Master

**Vantagens**

- As leituras e escritas podem ser feitas em qualquer servidor;
- Possibilidade de balanciamento de carga nas escritas e leituras.

**Desvantagens**

- Aumento da resolução de conflitos, tendo em conta que podem ser feitas alterações em simultâneo em cada nó;
- Maior latência de comunicação, já que sempre que existem actualizações os nós têm de sincronizar.

### 2.2.5 Escalabilidade na camada de dados

- **Replicação**, se a informação estiver replicada em vários nós, ao utilizar um balanciador de carga os pedidos serão distribuídos pelos diferentes nós, dessa forma não será sobrecarregado o mesmo nó e o sistema será escalado.

Um exemplo de como escalar um sistema de forma horizontal para leituras é o ***Read scale-out* com replicação**[1]. Esta estratégia pode ser adoptada quando temos um sistema que faz muitas leituras e poucas escritas.
Neste caso, poderá ser utilizada uma solução por exemplo de master slave, onde as escritas são feitas no Master e as leituras são feitas nos Slave(s).

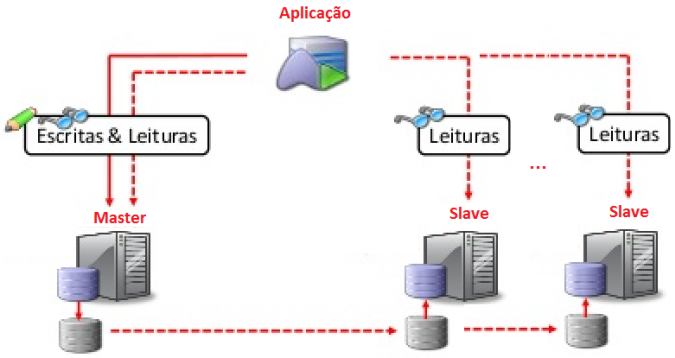


Figura 2.12: Read scale-out com replicação [10]

Mais adiante neste documento será aprofundado o tema da replicação.

- **Sharding**, consiste na partição horizontal das tabelas de uma base de dados, de forma a colocar os fragmentos em diferentes nós.
  Por exemplo, em vez de ter um nó por onde passam todos os acessos aos alunos do ISEL, poderia ser feita uma fragmentação horizontal, onde no nó 1 estariam os alunos com número par e no nó 2, todos os alunos com número ímpar.

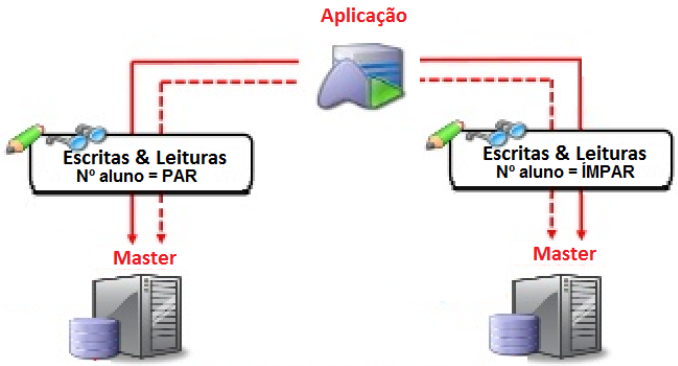


Figura 2.13: Sharding [10]

Desta forma não seria sobrecarregado o mesmo nó, e seria possível distribuir os acessos e desta forma escalar o sistema, também para escrita.

### 2.2.6 Controlo transaccional

O controlo transaccional é um princípio fundamental quando se fala de sistemas de informação empresariais complexos. Isto porque, muitas vezes é necessário fazer várias operações em simultâneo, por vezes em servidores diferentes como se de uma operação atómica se tratasse.
Para garantir essa atomicidade é necessário recorrer a trasacções distribuídas. Através da especificação XA que faz parte do modelo X/OPEN Distributed Transaction Processing (DTP) é possível garantir essa atomicidade.

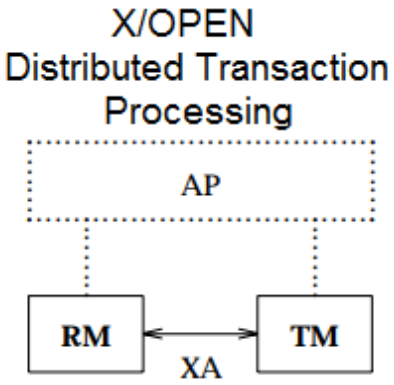


Figura 2.14: Modelo X/OPEN [1]

Tal como é possível observar na figura 2.14, o modelo X/OPEN DTP conta com 3 componentes principais:

- **Application Program (AP):** É responsável por definir os limites das transações e especificar as acções a executar na transacção.
- **Resource Manager (RM):** Providência acesso aos recursos partilhados, como é o caso das bases de dados.
- **Transaction Manager (TM):** Atribui identificadores às transacções, monitoriza o progresso e assume a responsabilidade pela conclusão da transacção bem como a recuperação em caso de falhas.

A especificação XA descreve a interface a utilizar entre o TM e o RM.
O objectivo desta especificação é garantir que múltiplos recursos em diferentes nós sejam acedidos como se estivessemos dentro da mesma transacção, desta forma garantimos as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Para garantir as propriedades ACID a estratéga utilizada pelos

gestores de transacções é o protocolo *two-phase commit* [11].

Para conseguir fazer o controlo transaccional nos SIE são utilizadas várias implementações de gestores de transacções distribuídas, entre elas temos:

- **Distributed Transaction Coordinator (DTC):** É um serviço de sistema da Microsoft que faz a coordenação de transações distribuídas implementando a especificação XA.
  Para transações distribuídas, cada máquina possui um Transaction Coordinator (TC), ou seja, o DTC naquela máquina.
  Quando um nó inicia uma transacção distribuída, o coordenador de transacções desse nó inicia a execução da transacção, faz a divisão da transacção em sub-transaccões, faz a distribuição das sub-transacções pelos gestores de transacções de cada nó e por fim dá ordem de *commit* ou *rollback*.

- **Java Transaction API (JTA):** O JTA, ou Java Transaction API, é uma Java Enterprise API para gerir transações distribuídas. Define uma ligação Java para o standard XA API para transações distribuídas.
  Usando o JTA, é possível escrever um programa que comunica com um serviço de transações distribuídas e usa esse serviço para coordenar a transação distribuída que acede e actualiza dados em dois ou mais recursos (computadores ou bases de dados no caso do JDBC).
  O JTA é uma implementação Java da especificação XA, que permite aos recursos, como um SGBD ou servidor JMS, participarem numa transação distribuída gerida por um TM externo.

O protocolo *two-phase commit* pode ser exemplificado com a figura 2.15 :

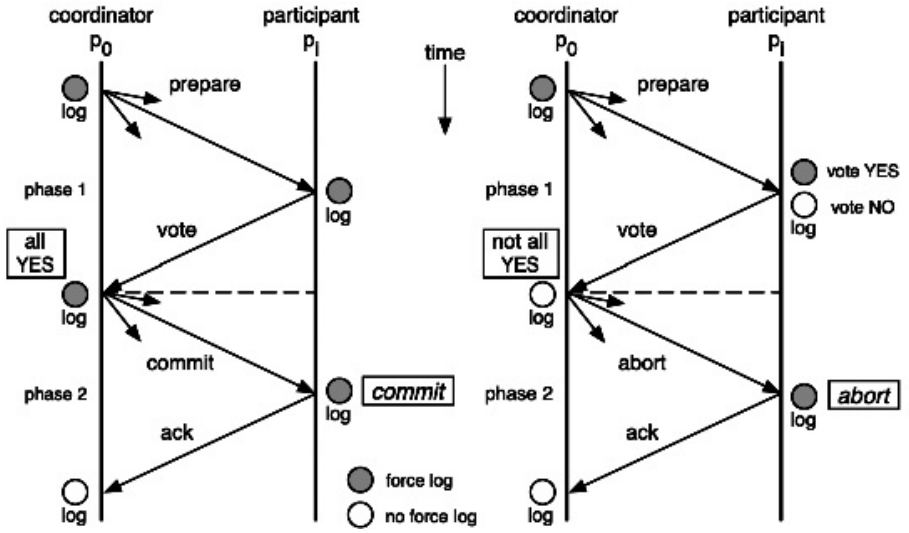


Figura 2.15: Protocolo two-phase commit [6]

Ou seja, existe uma primeira fase onde o coordenador pede a todos os recursos para votarem, caso todos votem positivamente, o coordenador dá ordem para todos fazerem *commit*. Caso exista pelo menos um que vote de forma negativa, o coordenador irá fazer *rollback*.

### 2.2.7 Desacopolamento na comunicação entre sistemas

As técnicas de desacopolamento são fundamentais quando se está perante sistemas de informação distribuídos já que o conceito de indirecção é cada vez mais aplicado a paradigmas de comunicação.

O desacoplamento na comunicação entre sistemas pode ser definido como a comunicação entre entidades de um sistema de informação distribuído sem que estes tenham que estar ligados em simultâneo, por vezes, podem inclusivamente não conhecer a existência um do outro. Os sistemas de informação distribuídos que utilizam esta técnica são menos rígidos no que diz respeito à mudança.

O desacopolamento na comunicação entre sistemas pode ter as seguintes propriedades:

- **Desacoplamento espacial:** O remetente e o receptor não precisam de se conhecer. Este desacoplamento permite um elevado grau de liberdade em lidar com a mudança, já que permite a mudança, alteração, actualização ou migração dos participantes.

- **Desacoplamento temporal:** permite que um sistema inicie comunicação ainda que o sistema destino não esteja iniciado, permitindo que o sistema origem prossiga o seu trabalho sem ficar bloqueado. Ou seja, o remetente e receptor podem ter tempos de vida diferentes, não têm de existir em simultâneo para comunicar.

Existem várias técnicas de comunicação indirecta que são utilizadas, nomeadamente:

- **Publish-subscribe:** É um sistema em que existe o conceito de publicador e subscritor. O publicador dissemina um evento estruturado para um serviço de eventos e os subscritores expressam interesse em eventos específicos. Tendo em conta que podem existir N subscritores para um determinado evento, estamos perante um paradigma de comunicação de um para muitos.

  Modelo do sistema *publish-subscribe*:

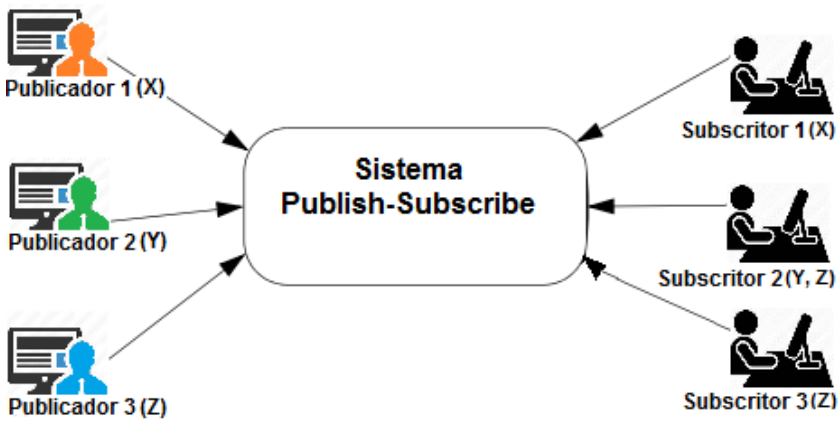


Figura 2.16: Publish-subscribe [1]

- **Espaço virtual compartilhado:** Por exemplo o conceito de tuple space introduzido no Linda [12].
  Na figura 2.17 está representado um espaço compartilhado abstracto, onde os processos compartilham mensagens de leitura e escrita.


Figura 2.17: Espaço virtual compartilhado

O conceito de espaço compartilhado pode ser visto como uma memória partilhada para sistemas distribuídos.
Os clientes que acedem a esta memória partilhada, podem executar as seguintes operações:

- **write**, escrever um tuplo no espaço compartilhado.
- **read**, ler um tuplo no espaço compartilhado.
- **take**, ler um tuplo no espaço compartilhado, removendo esse tuplo do espaço.
- **notify**, ser notificado quando um tuplo que respeite determinado critério seja escrito no espaço compartilhado.

- **Filas de mensagens:** É um sistema muito importante dentro dos sistemas de comunicação indirecta. No caso do publicador-subscritor a comunicação é de um para muitos, ao contrário das filas de mensagens que fornecem um serviço ponto-a-ponto possibilitando a comunicação indirecta e desta forma possibilitando desacoplamento temporal e espacial.
  As filas de mensagens são muito utilizadas na integração entre sistemas dentro de uma empresa, fazendo uso do fraco acopolamento temporal inerente às filas de mensagens.
  Nas filas de mensagens vários processos podem enviar mensagens, bem como vários processos podem remover mensagens de uma fila. Normalmente as filas são FIFO, no entanto, na maioria das implementações existe o conceito de prioridade, podendo ser atendidas mensagens mais prioritárias primeiro que as menos prioritárias.

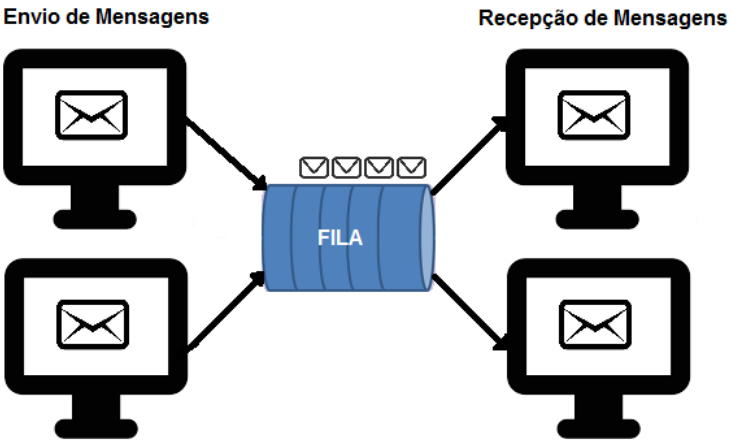


Figura 2.18: Imagem de exemplo de um sistema com filas de mensagens

# 3

# Camada de dados (CD)

## 3.1 Construção da CD

A camada de dados só poderá ser construida depois de se analisar e entender os requisitos impostos pela organização, como por exemplo:

- **Disponibilidade** - Perceber o tempo de quebra de serviço tolerado pela organização;
- **Escalabilidade** - Perceber o hardware que a organização pretende utilizar, e a probabilidade da necessidade de escalar;
- **Número de acessos em simultâneo** - Saber o número de acessos que se pretende em simultâneo para garantir um nível de serviço apropriado;
- **Maioritariamente acessos de escrita ou leitura** - Saber se a aplicação é maioritariamente de escrita ou leitura será de especial importância para tomar decisões na construção da camada de dados, nomeadamente na estratégia de replicação a utilizar;
- **Localização geografica dos servidores** - Replicar a informação em diferentes locais. Desta forma a informação pode ser preservada mesmo que exista uma catastrofe, por exemplo, incêndio, atentado ou sismo. Em caso de zonas sismicas deve haver um estudo prévio para que a informação seja replicada para uma zona que não esteja sobre a mesma placa tectónica.

Após analisar todos os pontos acima mencionados, devem ser adoptadas estratégias para ir de encontro às necessidades da organização, como:

- Hardware necessário;
- Utilização ou não de Bases de dados distribuídas;
- Transacções distribuídas;
- Replicação.

## 3.2 Bases de dados distribuídas (BDD)

Nesta secção será abordada a possibilidade ou não da implementação de BDD em MySQL e SQL Server, bem como apresentada a sua implementação.

### 3.2.1 Fragmentação em MYSQL vs SQLServer

A fragmentação de uma base de dados relacional, consiste na aplicação dos conceitos teóricos das tecnicas de fragmentação vertical e horizontal.
Estes conceitos dão origem a novas bases de dados que devem ser criadas em diferentes nós. Até este ponto, podemos afirmar que qualquer SGBD é capaz de realizar esta tarefa.
Os problemas podem surgir quando existe necessidade de aceder a dados que estão em diferentes nós, como se de uma base de dados centralizada se tratasse.
Tanto em MYSQL como em SQL Server a fragmentação dos dados é possível e o código para criação das tabelas, vistas ou procedimentos armazenados é muito semelhante, não havendo grande diferença do esforço despendido em cada um dos SGBDs.
Já no que diz respeito às configurações necessárias para permitir acessos remotos entre os diferentes nós, existem algumas diferenças.

Na figura 3.1 é ilustrado um exemplo simples da obtenção da visão global a partir de dois fragmentos. Como é possível verificar, existem dois fragmentos R1 e R2 que estão alojados em servidores diferentes. No fragmento R1 está presente um identificador e um nome, no fragmento R2, está presente o identificador e a localização.

A junção destes dois fragmentos dá origem a uma nova tabela R que contém a visão global dos dados.

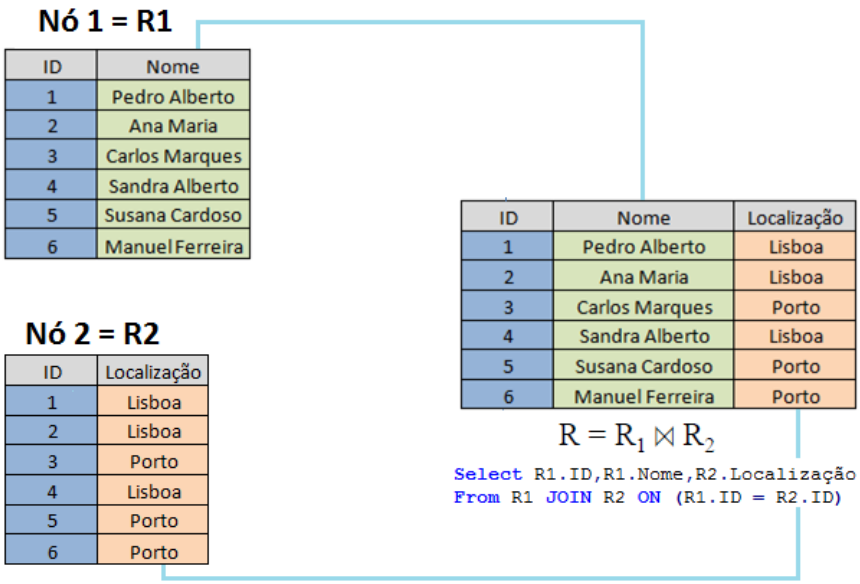


Figura 3.1: Visão global

De seguida são apresentados os passos necessários para criar uma vista que contenha a informação presente nas duas tabelas distribuídas em SQL Server e MYSQL como se de uma tabela centralizada se tratasse.

**Implementação em SQL Server:**

1. Criar uma ligação para cada um dos nós;

   - Nó 1:

```
exec sp_addlinkedserver @server = 'Servidor2\
   MSSQLServer', @srvproduct = 'SQL Server'
```

   Listagem 3.1: addlinkedserver para acesso ao Nó2

   - Nó 2:

```
exec sp_addlinkedserver @server = 'Servidor1\
   MSSQLServer', @srvproduct = 'SQL Server'
```

   Listagem 3.2: addlinkedserver para acesso ao Nó1

2. Criar sinónimos para garantir indepêndencia dos esquemas de fragmentação e lógico global relativamente à distribuição e simplificar a escrita das interrogações e futuras alterações;

- Nó 1:

```
CREATE SYNONYM SERVER2_Localidade for [Servidor2\
   MSSQLServer].BD.dbo.Localidade
```

Listagem 3.3: SYNONYM para acesso ao Nó2

- Nó 2:

```
CREATE SYNONYM SERVER1_Nomes for [Servidor1\
   MSSQLServer].BD.dbo.Nomes
```

Listagem 3.4: SYNONYM para acesso ao Nó1

3. Criar uma vista como se de um SGBD centralizado se tratasse;

- Nó1:

```
create view NomeLocal as
select n.ID, n.Nome, l.Localizacao
  from BD.dbo.Nomes n
  join SERVER2_Localidade l on n.ID=l.ID
```

Listagem 3.5: Vista NomeLocal

**Implementação em MYSQL:**

1. Configurar o Nó 1 e 2 para aceitar pedidos remotos desde o servidor onde se pretende ter a visão global;
   Fazer uma alteração no ficheiro "/etc/mysql/mysql.conf.d/mysqld.cnf". Na linha onde temos o atributo "bind-address = xxx.xxx.xxx.xxx " os "xxx" devem ser substituidos pelo IP da maquina que queremos dar acesso ou colocar "0.0.0.0" se quisermos dar acesso a todas as origens.

2. Dar permissões para o utilizador aceder à base de dados destino;

```
create user 'root'@'192.168.56.102' identified by '
    P4ssw0rd!' ;
create user 'root'@'%' identified by 'P4ssw0rd!' ;
grant all on *.* to 'root'@'192.168.56.102';
grant all on *.* to 'root'@'%';
```

Listagem 3.6: Permissões acesso remoto

3. Activar a opção engine federated;
   Fazer uma alteração no ficheiro "/etc/mysql/mysql.conf.d/mysqld.cnf". É necessário adicionar uma linha no ficheiro a dizer "federated".

4. Criar duas tabelas federadas no nó onde se pretende ter acesso global;

```
create table Nomes(
  ID int,
  Nome varchar(20)

)
ENGINE = FEDERATED
CONNECTION = 'mysql://root:P4ssw0rd!@192.168.56.105/
    BD/Nomes';

create table Localizacao(
  ID int,
  Localizacao varchar(20)

)
ENGINE = FEDERATED
CONNECTION = 'mysql://root:P4ssw0rd!@192.168.56.103/
    BD/Localizacao';
```

Listagem 3.7: Criar tabelas federadas

5. Criar uma vista como se de um SGBD centralizado se tratasse;
   Neste ponto não existe diferenças em relação ao SQL Server.

**Tabelas Federadas:**
O MYSQL disponibiliza vários tipos de tabelas, por omissão o tipo de tabela utilizado pelo MYSQL é InnoDB.
O tipo de tabela FEDERATED permite aceder remotamente a dados de uma base de dados MySQL sem usar replicação ou tecnologia de cluster. Ao fazer consultas sobre uma tabela FEDERATED são carregados os dados das tabelas remotas (federadas) automaticamente. Nenhum dado é armazenado nas tabelas locais.

Comparando as implementações em SQL Server e MYSQL, é possível verificar que:

1. Ambos os SGBD possibilitam a fragmentação dos dados e obter uma visão global após a fragmentação;
2. O MYSQL tem um custo maior no que diz respeito ao tempo e quantiade de configurações a realizar na implementação da solução;
3. Em caso de alteração da localização de um servidor ou dos nomes das bases de dados ou tabelas, o SQL Server simplifica a alteração já que apenas será necessário alterar os synonymos e recriar os linked servers. O MYSQL é mais rígido, sendo necessário alterar todas as tabelas federadas;

## 3.3 Controlo transaccional

O controlo transaccional é um conceito muito importante quando estamos perante bases de dados distribuídas, tendo em conta que estas estão apenas logicamente inter-relacionadas, e é necessário dotar os sistemas de mecanismos que garantam atomicidade e consistência em operações que envolvam vários nós.

### 3.3.1 Implementação SQL Server vs MySQL

**Implementação em SQL Server:**

1. Activar o serviço "Coordenador de Transacções Distribuídas";
2. Utilizar as transacções distribuídas;
   Exemplo de um procedimento armazenado para a inserção de uma Pessoa, na base de dados distribuída apresentada na figura 3.1:

```
1  CREATE PROCEDURE InserirPessoa (@id int, @nome
     varchar(20), @localizacao varchar(20))
2   AS
3   SET XACT_ABORT ON
4    BEGIN DISTRIBUTED TRANSACTION
5       INSERT INTO BD.dbo.Nomes([ID],[Nome])
6        VALUES(@id,@nome)
7
8       INSERT INTO SERVER2_Localidade([ID],[
          Localizacao])
9        VALUES(@id,@localizacao)
10
11   COMMIT TRANSACTION
```

Listagem 3.8: Procedimento armazenado para inserir Pessoa

**Implementação em MYSQL:**

Na documentação do MYSQL é indicado que não suporta por si só o papel de coordenador de transacções, apenas permite participar numa transacção distribuída como gestor recursos.
Assim sendo, será necessário fazer o controlo transaccional na camada aplicacional utilizando um coordenador de transacções externo.
No caso do Java EE corresponde ao Java Transactions Service que é implementado pelos contentores de EJB. Este papel poderá ser desempenhado pelo servidor Glassfish, JBoss ou outro.

**Documentação MYSQL:** [4]
"Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.
- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are "branches" of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

**The MySQL implementation of XA enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.** "

Ao comparar o controlo transaccional em SQL Server e MYSQL, é possível verificar que:

1. Apenas o SQL Server permite fazer o controlo transaccional distribuído na camada de dados;
2. MYSQL não consegue fazer o papel de coordenador de transacções distribuídas, obrigando a que esse papel seja feito na camada aplicacional;

## 3.4 BDD Replicação

### 3.4.1 Implementação em MySQL vs SQL Server

Tanto o SQL Server como o MySQL possibilitam a implementação das topologias de replicação anteriormente apresentadas.
De seguida serão indicados os passos a seguir para a implementação de topologias de replicação em MySQL e SQL Server para posteriormente ser feita uma comparação do esforço despendido.

**Implementação em SQL Server**

De seguida apresento duas opções de como implementar em SQL Server uma replicação do tipo *Master-Slave*:

1. Uma possível solução para a implementação da replicação *Master-Slave* de forma síncrona poderá ser feita através de transacções distribuídas e incluido algumas restrições:
   - Permitir apenas alterações no servidor *master*;
   - Alterações apenas podem ser feitas através de procedimentos armazenados que por sua vez contêm transacções distribuídas;
2. Outra possível implementação seria através do mecanismo de *Publisher-Subscriber* do SQLServer:
   - A configuração deste mecanismo é feito através de um *Wizard* intuitivo do SQL Server;
   - **Publisher:** Representa a instância onde se situa a base de dados que será replicada;
   - **Subscriber:** Representa a instância onde se situa a base de dados que irá receber os dados replicados do *Publisher*;
   - **Tipo de replicação:** É possível configurar o tipo de replicação que será feito: **Snapshot** (cópia completa dos dados de X em X tempo), **Transacional** (para alterações incrementais nos subscritores). O *Wizard* possibilita apenas a escolha de replicação assíncrona.

De seguida apresento duas opções de como implementar em SQL Server uma replicação do tipo *Master-Master*:

1. Uma possível solução para a implementação da replicação *Master-Master* de forma síncrona poderá ser feita através de transacções distribuídas e incluído algumas restrições:
   - As chaves geradas em cada servidor, não podem colidir (por exemplo, utilizar uma chave *identity* com crescimento impar num servidor e par no outro);

- Alterações apenas podem ser feitas através de procedimentos armazenados que por sua vez contêm transacções distribuídas;

2. Outra possível implementação seria através do mecanismo de replicação Peer-to-Peer [1] do SQLServer:

   - A estrutura das tabelas a replicar teria de ser igual em cada servidor;
   - Teria de se garantir que não haverá duplicação de chaves (Por exemplo, utilizar uma chave *identity* com crescimento impar num servidor e par no outro);
   - Despois apenas seria necessário utilizar a configuração *Wizard* deste mecanismo que é disponibilizada pelo SQL Server;
     O *Wizard* possibilita apenas a escolha de replicação assíncrona.

**Implementação em MYSQL**
A implementação da replicação em MYSQL pode ser feita de três formas:

- **Assíncrona**

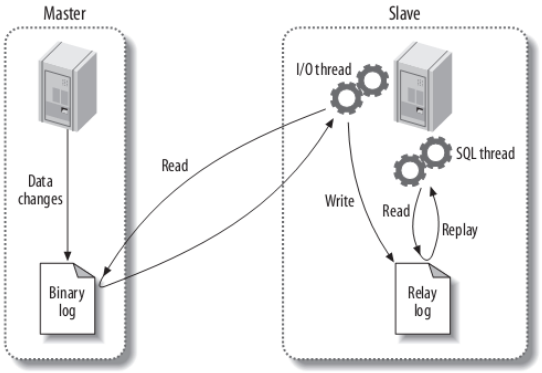


Figura 3.2: Funcionamento da replicação no MySQL [7]

  – O *master* regista todas as alterações no log binário.

  - O *slave*, através do fio de execução de I/O, abre uma conexão com o *master* e inicia o processo de esvaziamento do log binário. O esvaziamento consiste na leitura de cada evento do *master*. O fio de execução de I/O escreve os eventos no *relay* log do *slave*.
  - É possível aplicar filtros para indicar o que deve ser replicado. Sem filtros, todas as bases de dados de origem são replicadas para as bases de dados escravas.

- **Semi-síncrona**

  - O *master* só persiste alterações quando recebe informação por parte de um *slave* a indicar que conseguiu fazer as alterações.
  - Se existirem vários *slaves*, e apenas um conseguir persistir as alterações, os restantes *slaves* vão funcionar de forma assíncrona.

**Documentação MYSQL: [4]**
"Semisynchronous replication can be used as an alternative to asynchronous replication:

  - A slave indicates whether it is semisynchronous-capable when it connects to the master.
  - If semisynchronous replication is enabled on the master side and there is at least one semisynchronous slave, a thread that performs a transaction commit on the master blocks after the commit is done and waits until at least one semisynchronous slave acknowledges that it has received all events for the transaction, or until a timeout occurs.
  - The slave acknowledges receipt of a transaction's events only after the events have been written to its relay log and flushed to disk.
  - If a timeout occurs without any slave having acknowledged the transaction, the master reverts to asynchronous replication. When at least one semisynchronous slave catches up, the master returns to semisynchronous replication.
  - Semisynchronous replication must be enabled on both the master and slave sides. If semisynchronous replication is disabled on the master, or enabled on the master but on no slaves, the master uses asynchronous replication.

"

- **Síncrona**

– É possível através da configuração de clusters como é o caso do MYSQL Galera [13], que garante que os dados só são persistidos depois de todos os nós confirmarem sucesso.

1. **Master-Slave**, Uma possível solução para a implementação da replicação *Master-Slave* de forma assíncrona será feita da seguite forma:

   (a) Configurar o local onde ficará o log binário e definir o ID dos servidores:
   Para isso basta acrescentar duas linhas no ficheiro "/etc/mysql/mysql.conf.d/mysqld.cnf": **Master:**

   - **Server-id = 1**, identificador único do *Master*;
   - **log bin = "/var/log/mysql/maysql-bin.log"**, local onde se gera o log binário;

   **Slave:**

   - **Server-id = 2**, identificador único do *Slave*;
   - **log bin = "/var/log/mysql/maysql-bin.log"**, local onde se gera o log binário;

   (b) Criar um utilizador com permissões para fazer a cópia dos dados do servidor *Master*;

```
GRANT REPLICATION SLAVE ON *.* TO 'user_replica'@'%' IDENTIFIED BY 'P4ssw0rd!';
#user: user_replica
#password: P4ssw0rd!
#Acesso do IP: Qualquer IP '%'
```

Listagem 3.9: Criar user para replicação

   (c) Fazer backup da BD *Master*;

```
SHOW MASTER STATUS
--File: mysql-bin.000006
--Position: 657
mysqldump -u root -p - -opt databaseMaster > databaseMasterBackup.sql
```

Listagem 3.10: Backup BD Master

A primeira instrução irá retornar o nome do ficheiro de log bem como a posição em que se encontra. Esta informação terá de ser guardada para ser utilizada posteriormente no servidor *slave*.
A segunda instrução, fará um *backup* da base de dados actual para ser restaurada no servidor *slave*.

De salientar que embora a replicação seja feita ao nível da base de dados, é possível acrescentar filtros no ficheiro "/etc/mysql/mysql.conf.d /mysqld.cnf"para replicar a base de dados apenas com determinadas tabelas, ou indicar que a base de dados deve ser replicada na totalidade à excepção de determinadas tabelas:

- **–replicate-do-table=name**, filtro que indica a tabela especifica que se pretende replicar. (para várias tabelas, repetir o filtro)
- **–replicate-ignore-table=name**, filtro que indica a tabela especifica que se pretende ignorar. (para várias tabelas, repetir o filtro)

(d) Restaurar na BD *Slave*;

```
1 CREATE DATABASE BDSlave;
2 mysql -u root -p BDSlave < /path/
     databaseMasterBackup.sql
```

Listagem 3.11: Restaurar backup na BD Salve

(e) Configurar o *slave* para apontar para o *master*:

```
1 STOP SLAVE;
2 CHANGE MASTER TO MASTER_HOST='192.168.56.101',
3       MASTER_USER='user_replica',
4       MASTER_PASSWORD='P4ssw0rd!',
5       MASTER_LOG_FILE='mysql-bin.000006',
6       MASTER_LOG_POS=657;
7 START SLAVE;
```

Listagem 3.12: Indicar ao slave quem é o master

A partir deste momento, todas as actualizações no servidor *master* vão ser replicadas no *slave*.

2. **Master-Master**, Uma possível solução para a implementação da replicação *Master-Master* de forma assíncrona será feita da seguite forma:

   (a) Configurar o local onde ficará o log binário e definir ID dos servidores: Deverá ser feito exactamente o mesmo que na configuração *Master-Slave* no ficheiro "/etc/mysql/mysql.conf.d/mysqld.cnf".

   (b) Criar um utilizador com permissões para fazer a copia dos dados do servidor *Master*;

   **Master1:**

```
1  GRANT REPLICATION SLAVE ON *.* TO 'user_replica'
       @'%' IDENTIFIED BY 'P4ssw0rd!';
2  #user: user_replica
3  #password: P4ssw0rd!
4  #Acesso do IP: Qualquer IP '%'
```

Listagem 3.13: Criar user para replicação

   **Master2:**

```
1  GRANT REPLICATION SLAVE ON *.* TO 'user_replica'
       @'%' IDENTIFIED BY 'P4ssw0rd!';
2  #user: user_replica
3  #password: P4ssw0rd!
4  #Acesso do IP: Qualquer IP '%'
```

Listagem 3.14: Criar user para replicação

   (c) No nó Master1, obter informação do log para configurar a Master2 como *Slave*:

```
1  show master status;
2  --File: mysql-bin.000001
3  --POSITION: 607
```

Listagem 3.15: Info log Master1

   (d) Configurar Master2 como *Slave* de Master1:

```
STOP SLAVE;
CHANGE MASTER TO MASTER_HOST='192.168.56.101',
MASTER_USER='user_replica',
MASTER_PASSWORD='P4ssw0rd!',
MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS=607;
START SLAVE;
```

Listagem 3.16: Indicar ao slave quem é o master

(e) No nó Master2, obter informação do log para configurar a Master1 como *Slave*:

```
show master status;
--File: mysql-bin.000004
--POSITION: 147
```

Listagem 3.17: Info log Master2

(f) Configurar Master1 como *Slave* de Master2:

```
STOP SLAVE;
CHANGE MASTER TO MASTER_HOST='192.168.56.103',
MASTER_USER='user_replica',
MASTER_PASSWORD='P4ssw0rd!',
MASTER_LOG_FILE='mysql-bin.000004',
MASTER_LOG_POS=147;
START SLAVE;
```

Listagem 3.18: Indicar ao slave quem é o master

3. **Replicações derivadas**, em MYSQL quando se implementa uma topologia de replicação derivada de *Master-Master* ou *Master-Slave*, é provável que exista um nó que seja *Slave* de duas ou mais *Masters*.
   Por omissão o MYSQL só permite que cada *Slave* tenha um único *master*, ou seja, por omissão não é possível ter um *Slave* com replicas de 2 *Masters* diferentes.
   No entanto o MYSQL permite a criação de canais para que os *Slaves* possam ter diferentes *Masters* como é apresentado de seguida:

   (a) Nas topologias de replicação *multi-source* é requisito que o tipo de repositório seja tabela, assim sendo é necessário acrescentar no ficheiro de configuração "/etc/mysql/mysql.conf.d/mysqld.cnf":

   - **master_info_repository**="TABLE"
   - **relay_log_info_repository** = "TABLE"

   (b) Para indicar ao *Slave* que deve replicar com o Master1 que está na porta 3451 e com o Master2 que está na porta 3452, é necessário o seguinte:

```
STOP SLAVE;
CHANGE MASTER TO MASTER_HOST='192.168.56.103',
MASTER_USER='user_replica',
MASTER_PORT=3451,
MASTER_PASSWORD='P4ssw0rd!',
MASTER_LOG_FILE='mysql-bin.000004',
MASTER_LOG_POS=147 FOR CHANNEL 'master-1';
```

Listagem 3.19: Indicar ao slave o Master1 e o seu canal

```
CHANGE MASTER TO MASTER_HOST='192.168.56.105',
MASTER_USER='user_replica',
MASTER_PORT=3452,
MASTER_PASSWORD='P4ssw0rd!',
MASTER_LOG_FILE='mysql-bin.000021',
MASTER_LOG_POS=742 FOR CHANNEL 'master-2';
```

Listagem 3.20: Indicar ao slave o Master2 e o seu canal

```
1 START SLAVE FOR ALL CHANNELS;
```

Listagem 3.21: Iniciar replicação para todos os Masters

Ao comparar a implementação das topologias de replicação *Master-Master* e *Master-Slave* em SQL Server e MYSQL, é possível verificar que:

1. Ambos os SGBDs possibilitam a implementação destas topologias de forma assíncrona;
2. Apenas é possível implementar estas topologias de forma síncrona em SQL Server, recorrendo a transacções distribuídas;
3. O SQL Server facilita a configuração destas topologias através do *Wizard* intuitivo, ao contrário do MYSQL que é mais moroso e complexo.

# 4

# Camada Aplicacional (CA)

## 4.1 Organização da Camada Aplicacional

Para a organização da camada aplicacional decidi fazer uma divisão em outras duas subcamadas:

- **Lógica de acesso a dados** - É a camada responsável por comunicar com a camada de dados;
- **Lógica de negócio** - É a camada onde está a lógica e regras da aplicação. Comunica diretamente com a camada de apresentação e a camada de acesso a dados;

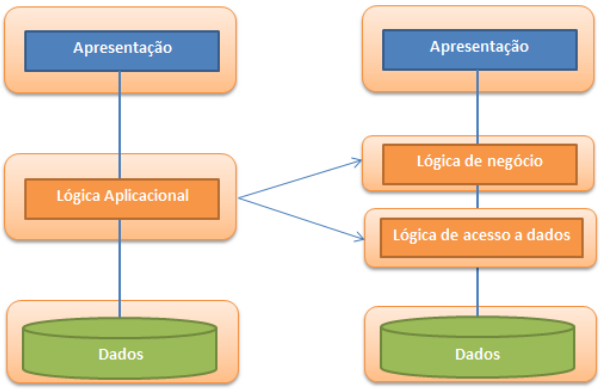


Figura 4.1: Camada aplicacional

Desta forma, se houver alterações à organização dos dados, ao modelo de dados, ao sistema de gestão de base de dados utilizado, ou à tecnologia de acesso a dados utilizada, apenas será necessário alterar a subcamada de lógica de acesso a dados, não sendo necessário alterar a subcamada lógica de negócio.

## 4.2 Windows Communication Foundation (WCF)

O WCF é uma ferramenta de desenvolvimento de *software* para desenvolvimento de serviços no Windows. Embora, seja possível criar serviços sem o WCF, a verdade é que a construção de serviços é significativamente mais fácil com o WCF. O WCF define um conjunto de padrões bem definidos para a interação de serviços, conversões de tipo, empacotamento e gestão de vários protocolos. O WCF fornece interoperabilidade entre serviços.

### 4.2.1 Organização do WCF

O WCF adopta uma estratégia de desenvolvimento orientado aos serviços. Para isso o WCF tem na sua organização conceitos chave que são necessários ter presente para o desenvolvimento de Serviços WCF, tais como:

- **Serviços** - Um serviço pode ser visto como uma funcionalidade que está exposta para ser consumida por clientes. Os clientes interagem com os serviços através de mensagens. Com o WCF, são utilizadas mensagens SOAP na interação entre cliente e serviço, ou seja, mensagens independentes dos protocolos de transporte.

  O serviço WCF expõe apenas as suas funcionalidades para o exterior, por esse motivo um serviço WCF expõe metadados que descrevem a funcionalidade disponível e possíveis formas de comunicação com o serviço. Os metadados são partilhados através de um ficheiro WSDL (*Web Services Description Language*) que contém um formato bem definido para este fim.

  No WCF, o cliente utiliza sempre um *proxy* para fazer as chamadas ao serviço, mesmo que o serviço seja local, tal como é ilustrado na figura 4.2. O *proxy* expõe todas as funcionalidades do serviços WCF para que os clientes possam interagir com o *proxy* como se estivessem a interagir directamente com o serviço WCF.

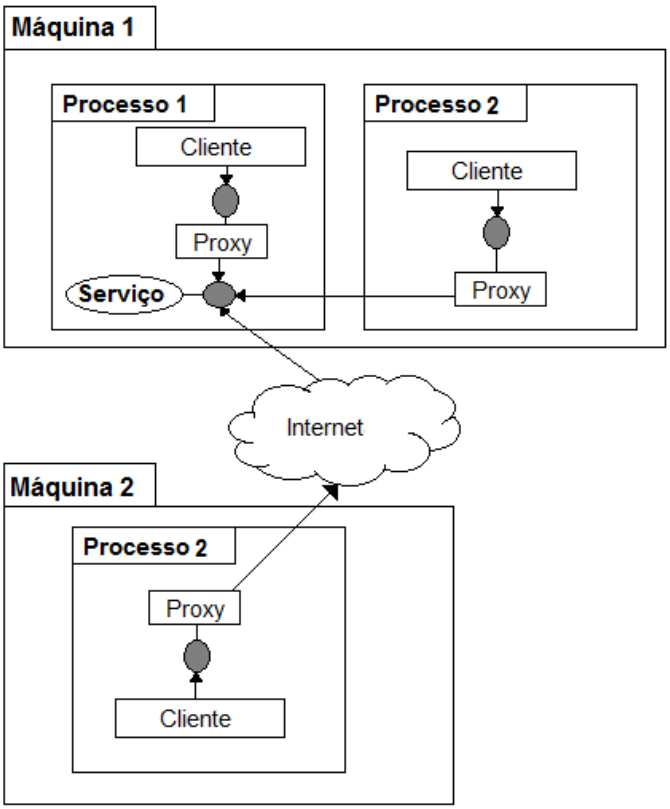


Figura 4.2: Comunicação entre Cliente e Serviço WCF [3]

- **Endereços** - Um serviço WCF está associado a um endereço único. O endereço é composto pela localização do serviço, protocolo de transporte que será utilizado para comunicar com o serviço WCF e opcionalmente uma *string* única como por exemplo o nome do serviço. A localização poderá ser o nome do servidor, IP, site entre outros. O protocolo de transporte poderá ser HTTP/HTTPS,TCP, IPC, MSMQ ou *Service bus*.

  **Alguns exemplos de endereços são:**

  - http://localhost:7575
  - http://localhost:7575/TesteServico
  - net.tcp://localhost:7576/TesteServico
  - net.msmq://localhost/private/TesteServico
  - net.msmq://localhost/TesteServico

  **Exemplo de endereços:**

"http://localhost:7575- Usando HTTP, vai para a máquina chamada localhost, onde estará alguém à espera da chamada no porto 7575.

"http://localhost:7575/MyService- Usando HTTP, vai para a máquina chamada localhost, onde estará alguém no porto 7575 chamado TesteServico à espera da chamada.

- **Contratos** - Todos os serviços WCF expõem contratos. Os contratos descrevem o que o serviço faz utilizando um padrão bem especificado.
  O WCF contém vários contratos, donde se destacam:

  - **Service contracts** - Descreve quais são as operações que o cliente pode executar no serviço. Os *Service contracts* são aprofundados no ponto 4.2.3.1.

  - **Operation contracts** - Este contrato é utilizado em conjunto com o ServiceContract. O OperationContract indica quais as operações do serviço estão visiveis para o cliente interagir, bem como caracteristicas particulares dessas operações.

  - **Data contracts** - Define o tipo de dados a ser trocado entre cliente e serviço WCF. O WCF permite a utilização de tipos como *int* e *string*, mas é possível criar novos tipos com os DataContracts.

  - **Fault contracts** - Define quais são os erros que são gerados pelo serviço e como o serviço manipula e propaga esses erros para os clientes.

- **Bindings** - O WCF definiu alguns *bindings* que contêm um conjunto de caracteristicas em relação ao protocolo de transporte, ao tipo de codificação das mensagens, padrão de comunicação, segurança, propagação de transacções e interoperabilidade.
  Embora o WCF tenha, por omissão, um conjunto definido de *bindings*, é possível definir novos *bindings*. O serviço publica nos seus metadados o *binding* necessário para que o cliente posso utilizar um *binding* igual.
  Na figura 4.3 é apresentada uma tabela que compara alguns dos *bindings* que estão definidos no WCF:

| Nome | Protocolo de Transporte | Codificação | Interoperável | Transacções | Segurança |
|---|---|---|---|---|---|
| BasicHttpBinding | HTTP/HTTPS | Text, MTOM | Sim | Não | Transport, Message, Mixed |
| NetTcpBinding | TCP | Binary | Não | Sim | Transport, Message, Mixed |
| NetNamedPipeBinding | IPC | Binary | Não | Sim | Transport |
| WSHttpBinding | HTTP/HTTPS | Text, MTOM | Sim | Sim | Transport, (Message), Mixed |
| NetMsmqBinding | MSMQ | Binary | Não | Sim | Message, (Transport), Both |

Figura 4.3: Tabela de comparação de Bindings

- ***Endpoint*** - Cada serviço está associado a um ou mais *endpoints* que definem onde o serviço está, como se pode comunicar com o serviço e um contrato a indicar o que o serviço faz. Geralmente os três elementos que compõem o *Endpoint* são lembrados através da sigla "ABC"(*Address*, *Binding* e *Contract*).

  Na listagem 4.1 é possível observar como se pode definir dois *Endpoints* para um mesmo serviço.

```
1  <service name = "TesteServico">
2    <endpoint
3      address = "http://localhost:7575/TesteServico"
4      binding = "wsHttpBinding"
5      contract = "IContratoTesteServico"
6    />
7    <endpoint
8      address = "net.tcp://localhost:7576/TesteServico"
9      binding = "netTcpBinding"
10     contract = "IContratoTesteServico"
11   />
12 </service>
```

Listagem 4.1: Definir dois endpoints para um serviço

### 4.2.2 Tipos de Instâncias

O WCF suporta três tipos de instâncias de serviços: Serviços *per-call*, *sessionful* e *singleton*.

#### 4.2.2.1 Serviço Per-Call

Nos serviços *per-call* é criada uma instância por cada chamada de cada cliente, ou seja, se um cliente fizer cinco chamadas para um serviço *per-call* o WCF irá criar

cinco instâncias, uma por cada chamada, depois do serviço retornar a instância fica pronta para ser destruida.
Serviços WCF *per-call* podem ser melhores para escalar o sistema de forma horizontal, tendo em conta que não partilham estado entre chamadas. Assim sendo, se o sistema necessitar de mais recursos bastará acrescentar mais servidores para atender os pedidos.

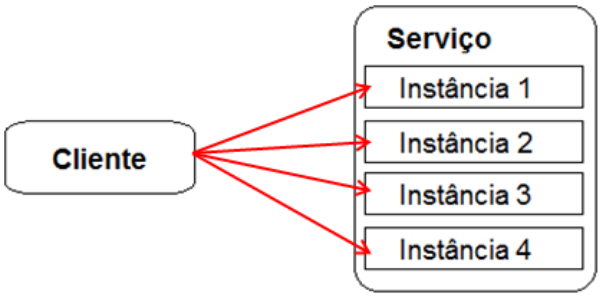


Figura 4.4: Instância Per-call

#### 4.2.2.2 Serviço Per-Session

Nos serviços *per-session* é criada uma instância por cada sessão de cliente, independentemente do número de chamadas que os clientes façam. Ou seja, se um serviço WCF *per-session* tem 100 clientes, mesmo que cada cliente faça N chamadas, irão existir sempre 100 instâncias.
As instâncias são apagadas quando os clientes fecham o *proxy* ou quando existe uma falha, por exemplo, um problema na ligação e a instância termina por tempo expirado.

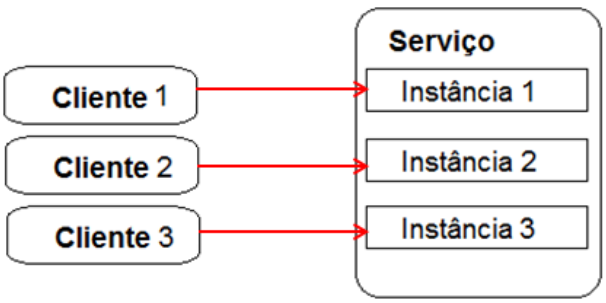


Figura 4.5: Instância Per-session

#### 4.2.2.3 Serviço Singleton

Nos serviços *singleton* existe apenas uma instância para todos os clientes. A instância do serviço é criada quando o serviço WCF é iniciado, e a instância só é terminada quando o serviço WCF é desligado.
Os serviços WCF *singleton* podem não ser aquados para SIE com muitos clientes, visto que são serviços mais dificeis de escalar. Tendo em conta que a instância é partilhada por vários utilizadores, é necessário que os clientes acedam ao estado da instância de forma ordeira, ou seja, um de cada vez o que pode degradar o serviço. Em alternativa o serviço pode ser configurado para que permita multiplos acesso em simultâneo, nesse caso, o programador terá de ter um esforço extra para garantir que todo o código é *thread-safe*.

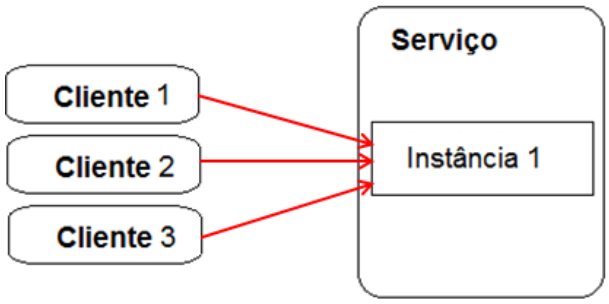


Figura 4.6: Instância Singleton

### 4.2.3 Construção de um Serviço WCF

Neste ponto irei explicar a forma de construir um serviço WCF, para isso, irei primeiro mostrar como definir os contratos dos serviços, das operações e da informação trocada entre cliente e serviço WCF.

#### 4.2.3.1 Service Contracts

O atributo "[ServiceContract]" é utilizado para marcar uma interface como sendo o contrato a seguir para acesso a um serviço WCF. Neste contrato são descritas as operações que estão disponíveis no serviço WCF para o exterior, como será feita a troca de mensagens (se será só num sentido ou em ambos os sentidos).

**Exemplo de um Service Contract:**

```
1 [ServiceContract]
2 interface IExemploSerciceContract
3 {
4   [OperationContract]
5   int contarLetras(string frase);
6 }
```

Listagem 4.2: Exemplo de um Service Contract

Tal como é possível observar na listagem 4.2, um *Service Contract* não é mais que uma interface marcada com o atributo "[ServiceContract]". Neste exemplo apresento também um método, marcado com o atributo "[OperationContract]".

#### 4.2.3.2 Operation Contracts

O atributo "[OperationContract]" apenas pode ser aplicado em métodos, não pode ser usado em propriedades, eventos ou outros. O WCF só percebe operações, e o atributo "[OperationContract]" expõe um método como uma operação lógica para executar como parte do contrato do serviço. Se forem definidos outros métodos na interface que não tem o atributo "[OperationContract]" estes não vão fazer parte do contrato.
De referir que as operações apenas podem retornar ou receber como parâmetro tipos primitivos ou *Data Contracts* que abordarei no ponto 4.2.3.3.

**Exemplo de Operation Contracts:**

```
1 [ServiceContract]
2 interface IExemploSerciceContract
3 {
4   [OperationContract]
5   int contarLetras(string frase);
6
7   [OperationContract(IsOneWay = true)]
8   void adicionaFrase(string frase);
9 }
```

Listagem 4.3: Exemplo de Operation Contract

Tal como é possível observar na listagem 4.3 estão definidas duas operações, já que temos dois métodos marcados com o atributo "[OperationContract]". O primeiro método "contarLetras" tem um tipo de comunicação do tipo *request-reply* que é o tipo de comunicação por omissão. Desta forma, quando um cliente invocar a operação "contarLetras" se a mesma demorar, o cliente ficará bloqueado até que termine de executar a sua tarefa. Já na segunda operação "adicionaFrase" o tipo de comunicação é *one-way* já que temos a operação marcada com "(IsOneWay = true)", significa que não retorna qualquer valor, por esse motivo, o cliente ao invocar a operação não irá ficar bloqueado à espera de retorno, seguindo de imediato o seu trabalho. As operações marcadas com "(IsOneWay = true)" retornam sempre *void* e em caso de excepção na operação o cliente não terá conhecimento.

#### 4.2.3.3 Data Contracts

O atributo "[DataContract]" é um acordo formal entre o cliente e serviço WCF e descreve os dados a serem trocados entre ambos. No contrato estão definidos quais os parâmetros que devem ser serializados ou desserializados de binario para XML entre as duas partes.

**Exemplo de um Data Contract:**

```
1 [DataContract]
2 public class Pessoa
3 {
4   [DataMember(Order = 1, IsRequired =true)]
5   public string PrimeiroNome { get; set; }
6   [DataMember(Order = 2)]
7   public string Ultimoome { get; set; }
8 }
```

Listagem 4.4: Exemplo de Data Contract

Na listagem 4.4 é possível observar que a propriedade "[DataMember]" utiliza os atributos "Order"e "IsRequired".

- **Order**: Utilizado para definir a ordem com que as propriedades do "[DataContract]" são serializados do lado do cliente. Sem este atributo, as propriedades do "[DataContract]" são serializados do lado do cliente por ordem alfabética.

- **IsRequired**: Indica se é de preenchimento obrigatório ou não. Por omissão ou se *false* é de preenchimento facultativo, e *true* é de preenchimento obrigatório.

#### 4.2.3.4 Serviço WCF

Agora que já falei sobre as principais peças necessárias para a construção de um simples serviço WCF, irei apresentar um exemplo completo da implementação de um serviço WCF.

**Exemplo de um Serviço WCF:**

```
1  [ServiceContract]
2  interface IExemploSerciceContract
3  {
4    [OperationContract]
5    int contarLetras(string frase);
6
7    [OperationContract(IsOneWay = true)]
8    void adicionaFrase(string frase, Pessoa remetente);
9  }
10
11 [DataContract]
12 public class Pessoa
13 {
14   [DataMember(Order = 1, IsRequired =true)]
15   public string PrimeiroNome { get; set; }
16   [DataMember(Order = 2)]
17   public string Ultimoome { get; set; }
18 }
19
20 public class ExemploServico : IExemploSerciceContract
21 {
22   public int contarLetras(string frase)
23   {
24     (...)
25     ret total;
26   }
27   public void adicionaFrase(string frase, Pessoa remetente)
28   { (...) }
29 }
```

Listagem 4.5: Exemplo da implementação de um serviço WCF

Na listagem 4.5 é possível observar a estrutura de um serviço WCF com todas as partes necessárias para a construção do mesmo.

#### 4.2.3.5 Cliente de um Serviço WCF

No ponto 4.2.3.4 foi apresentado o exemplo de um serviço WCF simples, neste ponto irei mostrar como é que um cliente de um serviço WCF pode aceder ao mesmo.

O cliente terá que adicionar uma referência para o serviço e atribuir um nome a essa referância. Esta referência poderá ser adicionada através da ferramenta Visual Studio. Neste caso o nome da referência é "ExemploServico".

```
1 ExemploServico proxy = new ExemploServico();
2 int result1 = proxy.contarLetras("Quantas letras?");
3 Pessoa p = new Pessoa();
4 p.PrimeiroNome = "Pedro";
5 p.UltimoNome = "Alberto";
6 proxy.adicionaFrase("Nova frase", p);
7 proxy.Close();
```

Listagem 4.6: Cliente de um serviço WCF

Tal como é possível observar na listagem 4.6 após o cliente adicionar uma referência para o serviço WCF, utiliza o serviço como se de um objecto normal se tratasse.

### 4.2.4 Transacções

Os serviços WCF permitem que se trabalhe directamente sobre um recurso transaccional fazendo a gestão das transacções de forma explícita, ou seja, programáticamente. No caso das transacções distribuídas o WCF permite que essa gestão seja feita com recurso a anotações.

#### 4.2.4.1 Gestão explícita de transacções

Como mostra a listagem **??**, se optarmos por este modelo, somos responsáveis por iniciar e terminar a gestão das transacções de forma explícita.

```
class MeuServico : IServicoTesteContrato
{
  public void MeuServico()
  {
    using (SqlConnection conexao = new SqlConnection("Dados
       de conexao"))
    {
      connection.Open();
      using(SqlCommand comando = new SqlCommand(conexao))
      {
        using(SqlTransaction transaccao = con.
           BeginTransaction())
        {
          comando.Transaction = transaccao;
          try
          {
            //Alteraes na base de dados
            transaccao.Commit();
          }
          catch
          {
            transaccao.Rollback();
            throw;
          }
        }
      }
    }
  }
}
```

Listagem 4.7: Gerir transacção de forma explícita

Como é possível observar na listagem 4.7, depois de criar uma ligação para a BD, criar um comando e obter a transacção e associá-la ao comando, são feitas

alterações na BD. No final das alterações é feito *commit*, se a BD continuar consistente e não houver qualquer excepção, os dados são persistidos, caso contrário, irá entrar no bloco "catch"e irá fazer um *rollback* anulanto todas as alterações que tinham sido feitas dentro da transacção.

#### 4.2.4.2 Gestão de transacções distribuídas

Uma transacção distribuída contém dois ou mais serviços independentes (geralmente em execução em diferentes contextos), ou mesmo apenas um único serviço com dois ou mais recursos transacionais.

No WCF o componente que é responsável por coordenar as transações distribuídas, é o "Microsoft Distributed Transaction Coordinator"(MDTC). Se este componente não estiver configurado corretamente, as transações distribuídas vão falhar.

#### 4.2.4.3 Propagação de transacções

O WCF tem o conceito de propagação de transacções para permitir que um serviço participe na tansacção de um cliente, por sua vez o cliente pode incluir operações em vários serviços WCF dentro da mesma transacção. (O cliente poderá ser um serciço WCF ou não).

A activação da possibilidade de propagar transacções pode ser feita programaticamente ou através do ficheiro de configuração. Para activar programaticamente, quando o *host* está a inciar o serviço WCF tem de colocar o parâmetro "TransactionFlow"do *Binding* que está a usar com o valor verdadeiro, conforme mostra a figura 4.8.

```
1 NetTcpBinding tcpBinding = new NetTcpBinding();
2 tcpBinding.TransactionFlow = true;
```

Listagem 4.8: Activar propagação de transacções programaticamente

Para activar no ficheiro de configuração é necessário acrescentar uma propriedade "TransactionFlow" com o valor verdadeiro, conforme a listagem 4.9.

```
<bindings>
  <netTcpBinding>
    <binding name = "TransactionalTCP"
       transactionFlow = "true"/>
  </netTcpBinding>
</bindings>
```

Listagem 4.9: Activar propagação de transacções ficheiro de configuração

Ao indicar que um serviço WCF permite propagação de transacções não significa que todas as operações do serviço suportam a propagação de transacções. Dentro de um serviço WCF, as operações podem permitir (*Allowed)*) ou não (*NotAllowed)*) o uso de propagação de transacções, mas também podem obrigar (*Mandatory)*) à propagação da transacção, através do atributo "TransactionFlowOption".

As operações podem estar marcadas com os seguintes atributos:

- **TransactionFlowOption.Allowed**
- **TransactionFlowOption.NotAllowed**
- **TransactionFlowOption.Mandatory**

Quando uma operação está configurada com o atributo "TransactionFlowOption.Allowed", se o cliente tiver uma transação, o serviço permitirá que a transação do cliente flua para o serviço.

Na listagem 4.10 mostro um exemplo de como configurar uma operação para permitir a propagação de transacções.

```
[ServiceContract]
interface IMeucontrato
{
  [OperationContract]
  [TransactionFlow(TransactionFlowOption.Allowed)]
  void MeuMetodo();
}
```

Listagem 4.10: Exemplo: Operação permite propagação

Quando uma operação está configurada com o atributo "TransactionFlowOption.NotAllowed", que é o valor por omissão, as transacções dos clientes não são propagadas para o serviço.

Na listagem 4.11 mostro um exemplo de como configurar uma operação para não permitir a propagação de transacções.

```
1 [ServiceContract]
2 interface IMeucontrato
3 {
4   [OperationContract]
5   [TransactionFlow(TransactionFlowOption.NotAllowed)]
6   void MeuMetodo();
7 }
```

Listagem 4.11: Exemplo: Operação não permite propagação

Quando uma operação está configurada com o atributo "TransactionFlowOption.Mandatory", obriga a que exista uma ligação entre cliente e serviço que permita transacções e também obriga a que o cliente chame a operação do serviço no contexto duma transacção. Caso estes prosupostos não se verifiquem será lançada uma excepção.

Na listagem 4.12 mostro um exemplo de como configurar uma operação para obrigar a propagação de transacções.

```
1 [ServiceContract]
2 interface IMeucontrato
3 {
4   [OperationContract]
5   [TransactionFlow(TransactionFlowOption.Mandatory)]
6   void MeuMetodo();
7 }
```

Listagem 4.12: Exemplo: Operação obriga a ter propagação

#### 4.2.4.4 Transacções: Completar e Votar

O WCF é responsável por todos os aspectos da propagação de transacções e da gestão global do protocolo *two-phase commit* e da gestão de recursos facilitando

e muito o trabalho do programador. No entanto, o WCF não sabe se uma transacção deve confirmar ou abortar porque não sabe se as alterações feitas sobre o estado do sistema são consistentes. Por esse motivo todos os serviços têm de votar, indicando se a transacção deve ser confirmada ou abortada. Além disso, apenas o serviço raiz sabe quando todos os serviços terminaram o seu trabalho, e só ai é que é iniciado o protocolo *two-phase commit*. A votação e indicação que a transacção foi completada podem ser feitas de forma declarativa ou explícita. Irei apenas mostrar a forma declarativa.

O WCF pode votar de forma automática para indicar se a transacção deve completar ou abortar. A votação de forma automática é feita através do atributo [TransactionAutoComplete=true].

Na listagem 4.13 é possível observar um serviço com a operação "MeuMetodo" a votar de forma automática, porque é utilizada um atributo para indicar que é precisa uma transacção e que a votação é automática.

```
1 class IMeuServico : IMeucontrato
2 {
3   [OperationBehavior(TransactionScopeRequired = true,
4   TransactionAutoComplete = true)]
5   void MeuMetodo(){
6   }
7 }
```

Listagem 4.13: Exemplo: Votar automaticamente uma transacção

### 4.2.5 Gestão de concorrência

O *thread-safe* num serviço WCF está directamente relacionado com o seu tipo de instância. Por exemplo, um serviço WCF *per-call* é necessáriamente *thread-safe* já que cria uma instância por chamada. Nessa instância só existirá uma única *thread*, no entanto se essa instância aceder a um recurso partilhado será necessário fazer uma gestão da concorrência, já que podem existir multipas *thread* a aceder ao recurso partilhado.
Um serviço WCF *per-session* necessita sempre de gerir concorrência já que um mesmo cliente pode utilizar o mesmo *proxy* e fazer múltiplas chamadas.

Um serviço WCF *singleton* é ainda mais suscetível a haver acessos concorrêntes, já que uma instância serve todas as chamadas de todos os clientes. Por esse motivo tem de se garantir acesso sincronizado.

O acesso concorrente à instância de um serviço é gerido pela propriedade "ConcurrencyMode" do atributo "[ServiceBehavior]". Os valores que a propriedade "ConcurrencyMode" admite são:

- **ConcurrencyMode.Single**
- **ConcurrencyMode.Multiple**
- **ConcurrencyMode.Reentrant**

Quando um serviço WCF é configurado com "[ConcurrencyMode.Single]", o WCF irá fornecer sincronização para o contexto do serviço e irá impedir acessos concorrêntes, associando o contexto que contém a instância do serviço a um *lock* da sincronização.
Sempre que existe uma chamada para um serviço "[ConcurrencyMode.Single]", primeiro é verificado se o *lock* de sincronização está disponível. Em caso afirmativo, o cliente poderá entrar no serviço e irá ficar com o *lock* da sincronização, libertando o *lock* da sincronização quando sair do serviço. Todos os clientes que tentem aceder ao serviço quando este está sem o *lock* da sincronização disponível, ficaram à espera numa fila ordenada.

Na listagem 4.14 apresento um exemplo da configuração de um serviço WCF com "[ConcurrencyMode.Single]".

```
1 [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Single)]
2 class MeuServico: IMeuContrato
3 {
4   void MeuMetodo();
5 }
```

Listagem 4.14: Exemplo: Modo de concorrência Single

Quando um serviço WCF é configurado com "[ConcurrencyMode.Multiple]", não existe qualquer tipo de *lock* de sincronização, todos os clientes que pretendam

aceder ao serviço poderam fazê-lo sem ter que ficar à espera de acesso a um *lock*. Nas instâncias *singleton* e *per-session* é necessário garantir o acesso sincronizado ao estado do serviço. Para garantir esse acesso sincronizado, podem ser utilizados *locks* do .NET como por exemplo os Monitores. No entanto, este tipo de controlo é mais complexo, dando por vezes origem a *deadlock*.
Para facilitar este controlo o WCF tem um atributo "[MethodImpl]" com a propriedade "MethodImplOptions.Synchronized"que pode ser colocada em cada uma das operações do serviço. Desta forma o código ficará *thread-safe*.

Na listagem 4.18 apresento um exemplo da configuração de um serviço WCF com "[ConcurrencyMode.Multiple]".

```
1 [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple
    )]
2 class MeuServico: IMeuContrato {
3   string[] recurso_partilhado;
4
5   [MethodImpl(MethodImplOptions.Synchronized)]
6   void MeuMetodo1(){ ... }
7   [MethodImpl(MethodImplOptions.Synchronized)]
8   void MeuMetodo1(){ ... }
9 }
```

Listagem 4.15: Exemplo: Modo de concorrência Multiple

Este modo de concorrência é idêntico ao "ConcurrencyMode.Single" já que atribui um *lock* de sincronização ao cliente que está a aceder ao serviço e nenhum outro cliente poderá aceder até este libertar o *lock* de sincronização.

### 4.2.6 Filas de mensagens - MSMQ

O MSMQ (Microsoft Message Queuing) possibilita a comunicação assíncrona entre sistemas. A comunicação é feita através de filas de mensagens que está disponível por omissão nos sistemas operativos Windows. O MSMQ possibilita o envio e receção de mensagens entre sistemas de uma forma simples.

Quando se trabalha com o MSMQ existe pelo menos um remetente e um destinatário. Quando o remetente faz o envio de uma mensagem para o destinatário, o destinatário não tem de estar disponível, poderá estar desligado, no entanto a

mensagem não será perdida, ficará armazenada numa fila de mensagens que está alojada no sistema operativo do *host* do remetende. Sairá da fila de mensagens do *host* do remetende quando o destinatário estiver disponível.

Para a utilização de MSMQ com WCF é necessário ter em conta o *Binding*, a configuração cliente e serviço, criação da fila no serviço indicando se é transaccional ou não e o envio de mensagens pelo cliente.

#### 4.2.6.1 Criar Serviço WCF com MSMQ

Para criar um serviço WCF utilizando MSMQ é necessário começar por definir o contrato. Conforme é possível observar na listagem 4.16, foi definido um contrato de um serviço que tem uma operação que não retorna qualquer informação para o cliente. Isto faz sentido, já que estamos perante uma comunicação assíncrona, não se espera uma resposta por parte do serviço.

```
1 public interface IServiceMSMQ
2 {
3   [OperationContract(IsOneWay =true)]
4   void operacaoMSMQ(int i);
5 }
```

Listagem 4.16: Contrato para serviço WCF com MSMQ

Na listagem 4.17 consta a implementação do serviço, que irá consumir as mensagens presentes na fila.

```
1 public class ServiceMSMQ : IServiceMSMQ
2 {
3   public void operacaoMSMQ(int i){
4     ( ... )
5   }
6 }
```

Listagem 4.17: Contrato para serviço WCF com MSMQ

#### 4.2.6.2 Binding

No ficheiro de configuração do serviço, é necessário indicar qual o *endpoint* do serviço. Quando se pretende utilizar filas de mensagens com MSMQ é necessário

escolher o *binding* adequado, no caso é o "netMsmqBinding". Também é necessário indicar o contrato e onde estará alojada a fila de mensagem.

```
<endpoint address="net.msmq://192.168.56.123/private/
   filaWCF"
     binding="netMsmqBinding"
       contract="IServiceMSMQ">
</endpoint>
```

Listagem 4.18: Definir endpoint para serviço com MSMQ

#### 4.2.6.3 Iniciar serviço WCF com MSMQ

No terminal onde está o serviço WCF e a fila de mensagens, é necessário executar as instruções da listagem 4.20, para criar a fila de mensagens com o nome definido no ficheiro de configuração ("filaWCF"). Neste exemplo a fila criada será transaccional. Depois basta iniciar o serviçoWCF.

```
if (!MessageQueue.Exists(@".\private$\filaWCF"))
  MessageQueue.Create(@".\private$\filaWCF", true);

ServiceHost host_msmq = new ServiceHost(typeof(ServiceoMSMQ
   ));

host_msmq.Open();
Console.WriteLine("Servio MSMQ iniciado. Enter para
   terminar");
```

Listagem 4.19: Iniciar serviço MSMQ

#### 4.2.6.4 Cliente de serviço MSMQ

Para que o cliente possa utilizar o serviço, terá que adicionar uma referência para o serviço WCF, para isso terá que ter acesso ao WSDL do mesmo.
Neste caso o cliente terá que adicionar uma referência para o serviço "net.msmq://192.168.56.123/private/filaWCF/?wsdl" e dar um nome ao *proxy*, neste caso foi dado o nome "PRX".

Como é possível observar na listagem 4.20, após o cliente obter a referência para o serviço WCF que utiliza MSMQ o envio das mensagens é transparente, o cliente chama as operações do serviço como se de um serviço síncrono se tratasse.

```
1 private PRX.ServiceMSMQClient prx = new PRX.
     ServiceMSMQClient();
2 prx.operacaoMSMQ(123);
```

Listagem 4.20: Exemplo: Utilização de serviço WCF com MSMQ

### 4.2.7 EntityFramework (EF)

O EntityFramework é uma ferramenta ORM que permite que se faça o mapeamento dos elementos de uma BD relacional numa aplicação orientada a objetos e desta forma obtendo o máximo de produtividade na persistência e recuperação dos dados.

Para se utilizar o EF, é necessário escolher qual a metodologia a utilizar:

- **Database First:** O modelo (EDMX) é gerado a partir do "*reverse engineering*" da base de dados e as classes são (auto) geradas a partir do modelo.
- **Model First:** Cria-se o modelo primeiro. A BD é gerada a partir do modelo e as classes são (auto)geradas a partir do modelo.
- **Code First (nova BD):** Criam-se as classe e mapeamentos manualmente. A base de dados é criada a partir do código. Usam-se "*migrations*" para fazer evoluir a BD.
- **Code First (BD existente):** Criam-se as classes e mapeamentos manualmente. Existem ferramentas de "*reverse engineering*"que facilitam este mapeamento.

Os exemplos apresentados são com a metodologia *Database First*.

#### 4.2.7.1 Contexto do EF

O contexto do EF pode ser visto como uma caixa onde estão as entidades. O contexto controla o ciclo de vida das alterações feitas. As alterações não são persistidas na BD até que seja invocado o método SaveChanges() do contexto.

O EF utiliza dois padrões que descrevo abaixo:

- **Unit of Work:** Pode ser visto como guardar uma cópia do estado do objeto e verificar as alterações em memória. Por esse motivo é que o método SaveChanges existe, porque o estado das entidades fica guardado em memória, até que esse método seja invocado e as mudanças sejam persistidas na BD.
- **Repository:** Faz com que se olhe para as entidades como se de uma coleção de objectos se tratasse. Disponibilizando métodos como "Add" para adicionar, "Remove" para remover ou "Find" para procurar.

As entidades que estão dentro do contexto do EF contêm um estado:

- **Added:** a entidade existe no contexto (DBContext) mas não na BD (implica gerar *INSERT*)
- **Unchanged:** a entidade existe no contexto e na BD e nã foi alterada (não propagar nada para a BD)
- **Modified:** a entidade existe na BD e foi modificada (gerar *UPDATE*)
- **Deleted:** a entidade existe na BD e no contexto, mas foi marcada para remoção (gerar *DELETE*)
- **Detached:** a entidade não está associada ao contexto (não existe "*change tracking*")

Os estados são importantes, já que o método "SaveChanges()" irá agir com o estado definido.

#### 4.2.7.2 Exemplo de utilização EF

Neste ponto irei apresentar um exemplo da utilização do EF com a metodologia *Database First*.

1. **Adicionar o EDMX ao projecto:**
   Dentro do projecto fazer adicionar item e escolher "ADO.NET Entity Data Model" tal como mostra a figura 4.7.

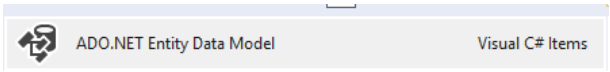


Figura 4.7: Item: ADO.NET Entity Data Model

Na janela seguinte escolher a opção "EF Designer from database" conforme apresentado na figura 4.8.

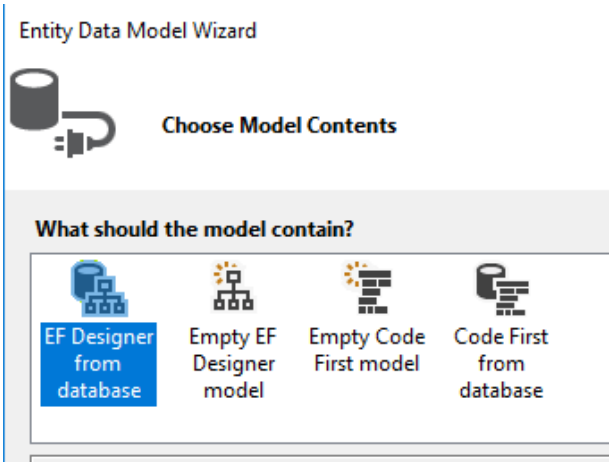


Figura 4.8: Gerar EDMX desde a BD

Na janela seguinte deve ser escolhida uma BD existente e os objectos da BD que se pretendem mapear para a aplicação orientada a objectos.

Estes passos dão origem ao ficheiro EDMX que contém o mapeamento dos objectos da BD escolhidos para a aplicação orientada a objectos. No Visual Studio irá ser possível ter acesso ao modelo da BD da qual foi feito o mapeamento conforme ilusta a figura 4.9.

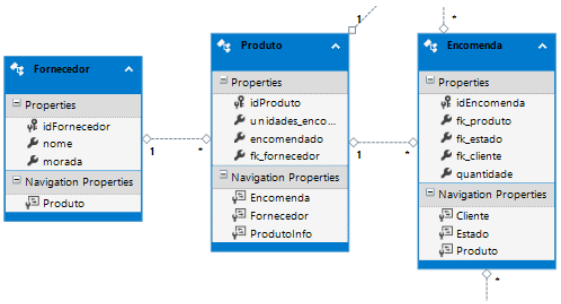


Figura 4.9: Modelo carregado no EDMX

Se for necessário adicionar ou remover algum dos objects da BD, basta actualizar o EDMX que irá surgir novamente uma janela a perguntar quais os objectos da BD devem ser carregados.

2. **Operações CRUD:**
   Na listagem 4.21 apresento o código de exemplo para a inserção de um registo na BD através do EF.

```
1  public bool InserirFornecedor(string nome, string
       morada)
2  {
3    bool ret = false;
4    var forn = new Fornecedor { nome = nome, morada =
       morada };
5    using(var ctx = new ExemploEntities())
6    {
7      ctx.Fornecedor.Add(forn);
8      ctx.SaveChanges();
9      ret = true;
10   }
11   return ret;
12 }
```

Listagem 4.21: Exemplo adicionar com EF

Na listagem 4.22 apresento o código de exemplo para a actualização de um registo na BD através do EF. De realçar, que antes de persistir as alterações através do método "SaveChanges()" foi alterado o estado para "Modified".

```
1  public bool AlterarFornecedor(int idFornecedor)
2  {
3    bool ret = true;
4    using (var ctx = new GestaoEntities())
5    {
6      var fornecedor = (from f in ctx.Fornecedor
7           where f.idFornecedor == idFornecedor
8           select f).FirstOrDefault();
9
10     fornecedor.morada = "Avenida Liberdade";
11     fornecedor.email = "32071@alunos.isel.pt";
12     ctx.Entry(fornecedor).State = System.Data.Entity.
          EntityState.Modified;
13     try
14     {
15       ctx.SaveChanges();
16     }
17     catch (Exception)
18     {
19       ret = false;
20     }
21   }
22   return ret;
23 }
```

Listagem 4.22: Exemplo actualizar com EF

Na listagem 4.23 apresento o código de exemplo para a obter um registo da BD através do EF. De realçar que como não são feitas alterações à BD não foi necessário persistir qualquer informação na BD e por isso não se invocou o método "SaveChanges()".

```
1  public FornecedorDTO GetFornecedor(int idFornecedor)
2  {
3    FornecedorDTO fornecedorDTO = new FornecedorDTO();
4    using (var ctx = new ExemploEntities())
5    {
6      var forn = (from f in ctx.Fornecedor
7        where f.idFornecedor == idFornecedor
8        select f).FirstOrDefault();
9
10     fornecedorDTO.id = forn.idCliente;
11     fornecedorDTO.morada = forn.morada;
12     fornecedorDTO.email = forn.email;
13   }
14   return fornecedorDTO;
15 }
```

Listagem 4.23: Exemplo obter recurso com EF

Na listagem 4.24 apresento o código de exemplo para a eliminar um registo na BD através do EF. De realçar, que antes de persistir as alterações através do método "SaveChanges()" foi alterado o estado para "Deleted".

```
1  public bool ApagarFornecedor(int idFornecedor)
2  {
3    bool ret = true;
4    using (var ctx = new GestaoEntities())
5    {
6      var fornecedor = (from f in ctx.Fornecedor
7      where f.idFornecedor == idFornecedor
8      select f).FirstOrDefault();
9
10     ctx.Entry(fornecedor).State = System.Data.Entity.
         EntityState.Deleted;
11     try
12     {
13       ctx.SaveChanges();
14     }
15     catch (Exception)
16     {
17       ret = false;
18     }
19   }
20   return ret;
21 }
```

Listagem 4.24: Exemplo apagar com EF

## 4.3 Java Enterprise Edition 8 (JEE)

Java EE oferece padrões para manipular transações com JTA (Java Transaction API), mensagens com o Java Message Serviço (JMS) ou persistência com Java Persistence API (JPA). Java EE contém um conjunto de especificações para o desenvolvimento de aplicações empresariais facilitando o desenvolvimento de recursos distribuídos, robustos e altamente disponíveis.

### 4.3.1 Arquitectura do JEE 8

Java EE tem um conjunto de especificações implementadas em diferentes Contentores. Os contentores são ambientes de execução do JEE que fornecem serviços para os componentes que os contentores hospedam. Os componentes utilizam contratos bem definios para permitir a comunicação com a infra-estrutura do JEE bem como com outros componentes.
A figura 4.10 mostra a relação lógica entre componentes bem como os protocolos usados para a comunicação entre componentes.

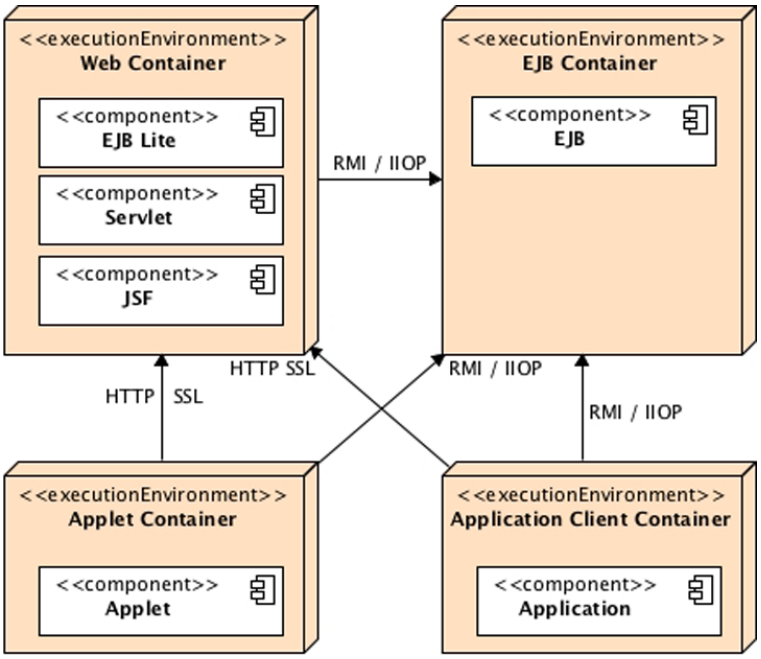


Figura 4.10: Contentores JEE [2]

### 4.3.2 Componentes

Os ambientes de execução do JEE definem quatro tipos de componentes que uma implementação tem de suportar:

- **Applets** - são aplicações de interface gráfica para o utilizador, que são executados num navegador web.
- **Aplicações** - são programas executados pelo cliente.
- **Aplicações Web** - são executado num contentor web e respondem a solicitações HTTP de clientes da web. Os Servlets suportam SOAP e RESTful web service endpoints. Estas aplicações podem conter EJBs.

- **Aplicações empresariais** - construidas com base em Enterprise Java Beans, serviços de mensagem Java, transações Java API, chamadas assíncronas, entre outros. Estas aplicações são executados num contentor EJB.

### 4.3.3 Contentores

Tal como foi ilustrado na figura 4.10, a infra-estrutura do Java EE é dividida em domínios lógicos chamados de contentores. Cada contentor tem uma função específica, suporta um conjunto de APIs e oferece serviços aos componentes (segurança, acesso às bases de dados, gestão de transações, diretório de nomes, injeção de recursos). Os contentores escondem a complexidade técnica e aumentam a portabilidade.
O Java EE tem quatro contentores diferentes:

- **Contentor de Applets** - são fornecidos pela maioria do navegadores web para executar componentes applet. Quando se desenvolve applets, o desenvolvedor pode focar-se no aspecto visual tendo em conta que o contentor oferece um ambiente seguro.
- **Contentor de aplicações cliente (ACC)** - inclui um conjunto de classes, bibliotecas e outros arquivos Java necessário para ter injeção de dependências, gestão de segurança para aplicações Java SE. O ACC comunica com o contentor de EJB utilizando RMI-IIOP e o protocolo HTTP para o contentor Web.
- **Aplicações Web** - são executadas num contentor web e respondem a solicitações HTTP de clientes da web. Os Servlets suportam SOAP e RESTful web service endpoints. Estas aplicações podem conter EJBs.
- **Contentor EJB** - responsável por gerir a execução dos enterprise beans (beans de sessão e beans por mensagens) contendo a lógica de negócios da aplicação JEE. Cria novas instâncias de EJBs, gere o seu ciclo de vida e fornece serviços como transação, segurança, concorrência, distribuição, serviço de nomes ou a possibilidade de ser invocado de forma assíncrona.

### 4.3.4 Serviços

Os contentores fornecem serviços subjacentes para os seus componentes. Os contentores permitem que o desenvolvedor, se concentre na implementação da lógica

de negócios em vez de resolver problemas técnicos enfrentados em aplicações empresariais.
A figura 4.11 mostra os serviços fornecidos por cada contentor.

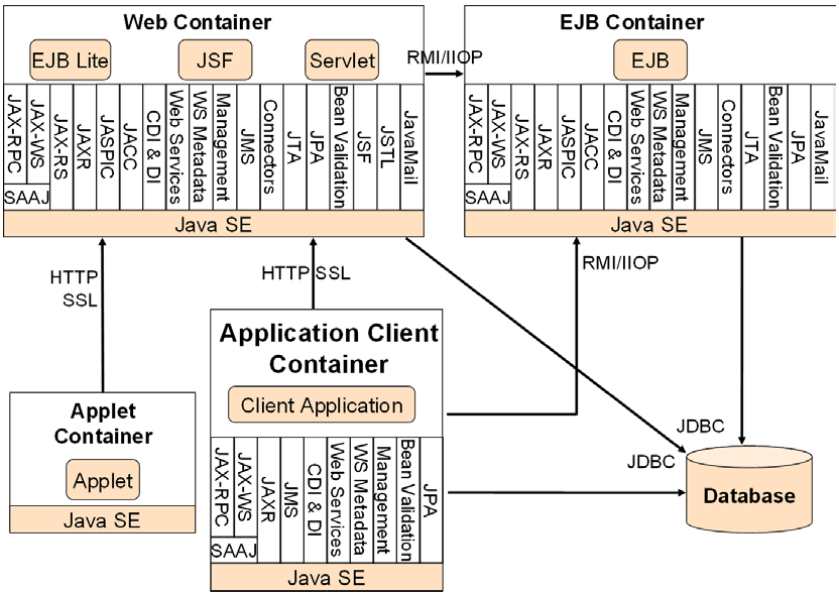


Figura 4.11: Serviços oferecidos pelos contentores JEE [2]

Como pode ser visto na A figura 4.11 o contentor de EJB disponibiliza vários serviços de grande utilidade para o desenvolvedor como por exemplo:

- **Java Message Service** - permite comunicação assíncrona entre componentes através de mensagens.
- **Java Naming and Directory Interface:** - usado para aceder a nomes e ao diretório do sistema. A aplicação associa nomes a objetos, e para encontrar esses objectos faz a procura num diretório.
- **Java Transaction API** - API para demarcação transaccional usada pelo contentor e a aplicação.
- **Java Persistence API** - API para mapeamento relacional de objetos (ORM).
- **Security services** - permite a utilização dos serviços autenticação para impor controlos de acesso aos utilizadores.

### 4.3.5 Empacotamento

Para que seja possível publicar componentes nos contentores Java EE, é necessário que em primeiro lugar o componente seja empacotado num arquivo com

formato standard. O Java SE define Arquivos Java (JAR), os quais são utilizados para juntar muitos arquivos (classes Java, descritores de implementação, recursos ou bibliotecas externas) num arquivo compacto (baseado no formato ZIP).
Na figura 4.12, é possível ver que o Java EE define diferentes tipos de módulos que têm o seu próprio formato de pacote com base neste formato jar comum.

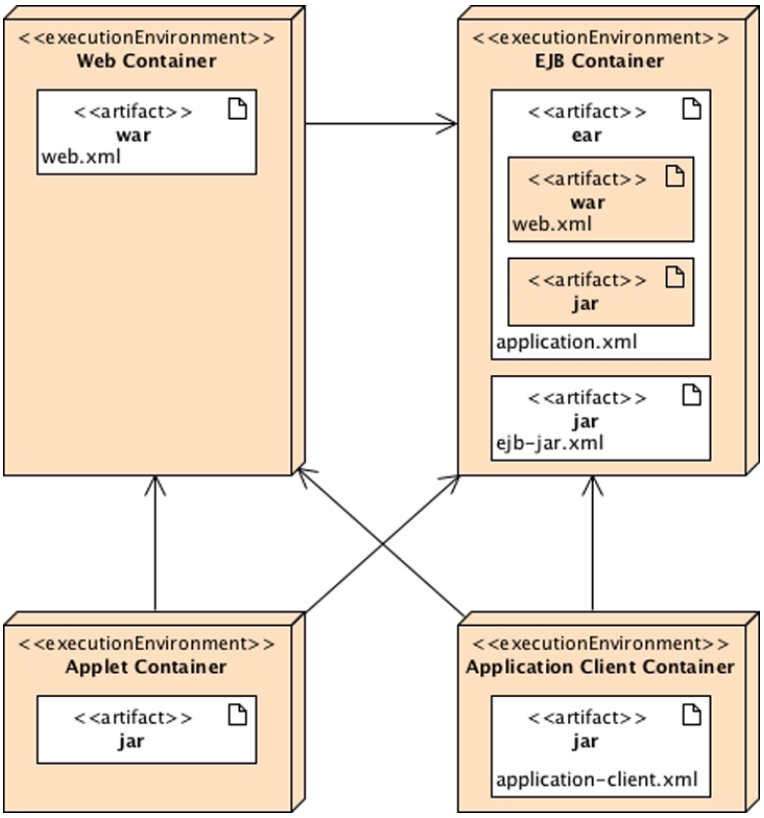


Figura 4.12: Arquivos nos contentores JEE [2]

De seguida são apresentados os formatos utilizados nos contentores do Java EE:

- **JAR (Java ARchive)** - é o arquivo mais simples, normalmente utilizado para juntar todas as classes de uma biblioteca.

- **WAR (Web ARchive)** - é o arquivo que possui todos os arquivos de uma aplicação web Java. Dentro do WAR estão as classes compiladas, outros JARs, arquivos HTML, JavaScript e imagens.

- **EAR (Enterprise ARchive)** - é um arquivo que junta várias aplicações. Um arquivo EAR, pode ter vários WAR, mas também JARs bem como configurações do servidor de aplicação.

### 4.3.6 Enterprise JavaBeans (EJB)

EJBs são componentes que funcionam do lado do servidor e estão num contentor que é executado numa JVM, por exemplo no GlassFish, JBoss ou Weblogic. Estes componentes contêm a lógica de negócios e tratam das transações e segurança. Os EJBs integram-se perfeitamente com outras tecnologias Java SE e Java EE, como JDBC, JavaMail, JPA, Java Transaction API (JTA), Java Messaging Service (JMS), Serviço de Autenticação e Autenticação Java (JAAS), Java Naming e Directory Interface (JNDI) e Remote Method Invocation (RMI). Os EJBs orquestram toda a camada de negocio.

Os EJBs usam um modelo de programação muito poderoso que combina facilidade de uso e robustez. Os EJBs são um modelo simples de desenvolvimento do lado do servidor Java, reduzindo a complexidade e, ao mesmo tempo, trazendo reutilização e escalabilidade para aplicações empresariais de missão crítica.

#### 4.3.6.1 Tipos de Enterprise JavaBeans (EJB)

Os EJB podem ser dos serguintes tipos:

- **Session beans:** Os Session beans são indicados para implementar a lógica de negócio, processos e fluxo do trabalho. Os Session beans são componentes geridos por um contentor, portanto, precisam ser empacotado num arquivo (arquivo jar, war ou ear) e feita a instalação num contentor.
  - **Stateless:** Não contêm qualquer estado associado entre chamadas. Uma instância pode ser usada por qualquer cliente, não existe estado.
  - **Stateful:** Existe estado entre chamadas de um cliente. O estado deve ser preservado entre chamadas de um cliente. Útil quando se está perante um cenário onde é necessário executar tarefas em váras etapas.
  - **Singleton:** Existe apenas um bean que é partilhado por todos os clientes e suporta acesso concorrente.
- **Message-driven beans (MDBs):** Os Message-driven beans podem ser usados para integrar sistemas externos recorrendo a mensagens assíncronas usando o JMS.

#### 4.3.6.2 Estrutura de uma Session Bean

A criação de um EJB de sessão consiste na criação de uma classe POJO com anotações. Conforme é possível ver na imagem em baixo, do lado esquerdo está uma classe POJO e do lado direito um EJB.

POJO - Plain Old Java Object - é simplesmente uma denominação que se dá a um objeto "normal", sem nada de especial. Um POJO tem campos, métodos, construtores, etc, mas não segue nenhum padrão pré-estabelecido.

A única diferença é a anotação @Stateless, que transforma a classe POJO num componente transaccional e seguro.
Para transformar uma classe POJO num EJB basta anotar a classe com @Stateless, @Stateful ou @Singleton, dependendo do tipo pretendido.

```
public class ClienteSessionBean {

    @PersistenceContext(unitName = "GestaoPU")
    private EntityManager emGestao;


    public ClienteDTO findById(int id) {
        ClienteDAO daoCliente = new ClienteDAO(emGestao);
        cli = daoCliente.findById(id);
        return clienteMapper.ClienteToClienteDTO(cli);
    }
}
```

```
@Stateless
public class ClienteSessionBean {

    @PersistenceContext(unitName = "GestaoPU")
    private EntityManager emGestao;


    public ClienteDTO findById(int id) {
        ClienteDAO daoCliente = new ClienteDAO(emGestao);
        cli = daoCliente.findById(id);
        return clienteMapper.ClienteToClienteDTO(cli);
    }
}
```

Figura 4.13: POJO vs EJB

Os Beans de sessão podem implementar interfaces locais, remotas ou não implementar qualquer interface. As interfaces locais e remotas definem um conjunto de métodos que o EJB irá ter disponíveis para o cliente utilizar.

- **Local:** Uma interface marcada com a anotação @Local, indica que os parâmetros dos métodos vão ser passados como referência.
- **Remota:** Uma interface marcada com a anotação @Remote, indica que os parâmetros dos métodos são passados como valor e é necessário garantir que os mesmos são serializados como indica o protocolo RMI.
- **Sem interface:** Se um EJB não implementa qualquer interface @Local ou @Remote, irá disponibilizar todos os métodos publicos do EJB e o acesso será apenas local.

Na figura 4.14 é possível ver os tipos de interface que devem ser utilizadas consoante as necessidades de acesso ao EJB.

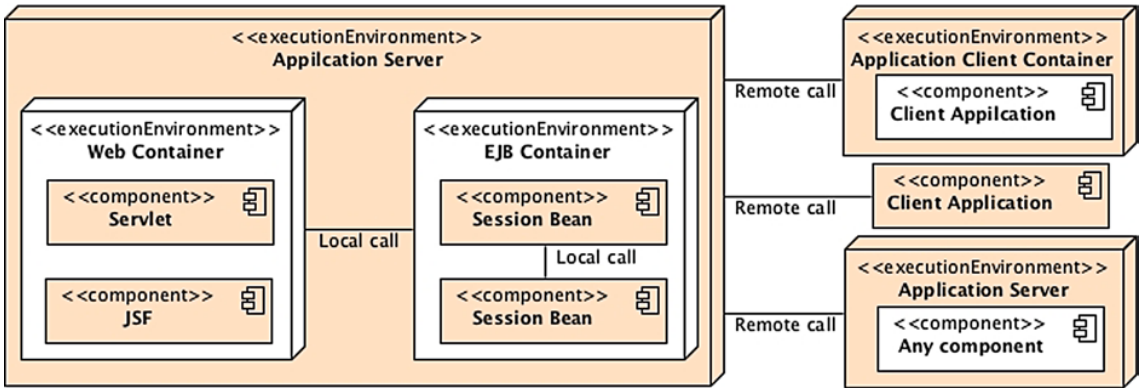


Figura 4.14: Interfaces EJB [2]

Como é possível ver na figura 4.14, quando se pretende aceder a um EJB dentro de um mesmo contentor ou Servidor Aplicacional, o EJB deve implementar uma interface local. Quando se está perante um caso em que é necessário aceder ao EJB desde um cliente que está fora do Servidor Aplicacional, o EJB deve implementar uma interface remota.

#### 4.3.6.3 EJB Stateless

No JEE os EJB Stateless são os tipos de SessionBean mais utilizados. Estes EJB são simples, poderosos e eficientes no processamento de tarefas de negócio sem estado.
Os EJB Stateless têm de garantir que uma tarefa é concluida no final de uma chamada a um método, não podendo preservar estado para futuras chamadas.

Na listagem 4.25 é apresentado um exemplo da criação de um novo produto bem como a sua persistência na base de dados num POJO. Como é possível ver, são feitas várias chamadas ao objecto produto antes da sua persistência na base de dados.

```
1 Produto prod = new Produto();
2 prod.setDescricao("Teclado");
3 prod.setPreco(29.9);
4 prod.setStock(10);
5 prod.setCodigo("tec-123b6");
6 prod.saveToDatabase();
```

Listagem 4.25: Exemplo persistência de produto

Na listagem 4.25, temos a lógica de negócio dentro do objecto Produto. A lógica de negócio deve ser implementada num serviço externo como um EJB, desta forma o objecto Produto poderá ter estado associado no entanto quando for necessário persistir os dados na base de dados, o cliente deverá utilizar uma instância do serviço externo que não tem estado mas que recebe como parâmetro o produto e persiste a informação na base de dados.

Na listagem 4.26 é apresentado um exemplo que utiliza um EJB Stateless:

```
1 Produto prod = new Produto();
2 prod.setDescricao("Teclado");
3 prod.setPreco(29.9);
4 prod.setStock(10);
5 prod.setCodigo("tec-123b6");
6 ejbStateless.saveToDatabase(prod);
```

Listagem 4.26: Exemplo persistência de produto

Na listagem 4.26, é possível ver que a instância Stateless é chamada para persistir um produto. Após a persistência do produto, a instância do EJB Stateless ficará disponível para atender pedidos de outros clientes.

As instâncias de EJB Stateless são geridas numa pool do contentor de EJB como ilustra a figura 4.15.

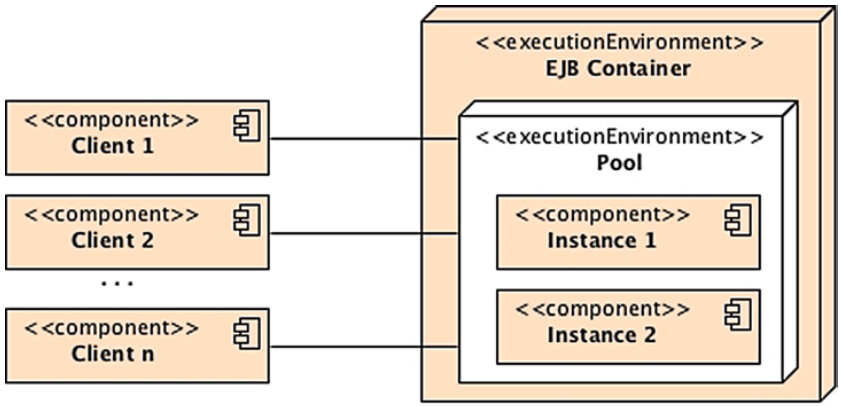


Figura 4.15: Interação entre clientes e EJB Stateless [2]

O contentor de EJBs cria um conjunto de instâncias de EJB Stateless para atender os pedidos dos clientes. Como não têm estado associado, estão em condições de atender qualquer chamada de qualquer cliente, assim sendo as instâncias de EJB

Stateless após concluirem um pedido feito por um cliente voltam a ficar disponíveis na pool do contentor de EJBs para atender outro cliente.

**Implementação de um EJB Stateless:**

```
@Stateless
public class EjbStateless {
  @PersistenceContext(unitName = "produtoPU")
  private EntityManager em;

  public Produto saveToDatabase(Produto prod) {
    em.persist(prod);
    return prod;
  }
}
```

Listagem 4.27: Implementação de um EJB Stateless

Para a construção deste EJB foi apenas necessário marcar a classe com a anotação @Stateless, foi utilizado o serviço de injecção de dependências para iniciar uma referência para o EntityManager.
Nos *Sessionbean Stateless*, o contexto de persistência é transacional, o que significa que qualquer método chamado neste EJB é transacional.

*Sessionbean Stateless* podem suportar um grande número de clientes, minimizando a necessidade de recursos, já que como foi referido anteriormente após concluirem um pedido feito por um cliente voltam a ficar disponíveis na pool do contentor de EJBs para atender outro cliente, desta forma não existe a necessidade de destruir o EJB e criar um novo.
Ao utilizar esta abordagem é possível melhorar a escalabilidade do sistema, já que o contentor não tem de guardar e gerir estados, assim sendo o contentor pode gerar vários *Sessionbean Stateless* para atender vários pedidos em simultâneo.

#### 4.3.6.4 EJB Stateful

EJB *Stateful* tal como os EJB Stateless contêm métodos com a lógica de negócio, no entanto os EJB *Stateful* preservam estado entre chamadas dos cliente ao contrário dos EJB *Stateless*.

Os EJB *Stateful* são bastante úteis quando se está perante tarefas que têm de ser executadas em vários passos necessitando de informação dos passos anteriores. Um exemplo de um caso onde poderá fazer sentido utilizar um EJB *Stateful* é numa loja online onde existe um carrinho de compras para cada utilizador.
A lista de produtos mantém os produtos que o cliente escolheu durante toda a interação. A interação entre o cliente e o EJB *Stateful* poderia ser feita da seguinte forma:

```
1  Produto prod = new Produto();
2  prod.setDescricao("Teclado");
3  prod.setPreco(29.9);
4  prod.setCodigo("tec-123b6");
5  ejbStateful.addToList(prod);
6  prod.setDescricao("Rato");
7  prod.setPreco(15.9);
8  prod.setCodigo("rat-111b9");
9  ejbStateful.addToList(prod);
10 ejbStateful.finishList();
```

Listagem 4.28: Interação de um cliente com um EJB Stateful

Ao ver a listagem 4.28 é possível perceber que quando um cliente chama um EJB *Stateful* o contentor EJB tem de fornecer a mesma instância do EJB para todos os pedidos do cliente, porque nessa instância está o estado do cliente. Por este motivo as instâncias de EJB *Stateful* não podem ser reutilizadas ao contrário do que acontece com os EJB *Stateless*.

As instâncias EJB *Stateful* são geridas pelo contentor de EJB criando uma instância por cada cliente como ilustra a figura 4.16.

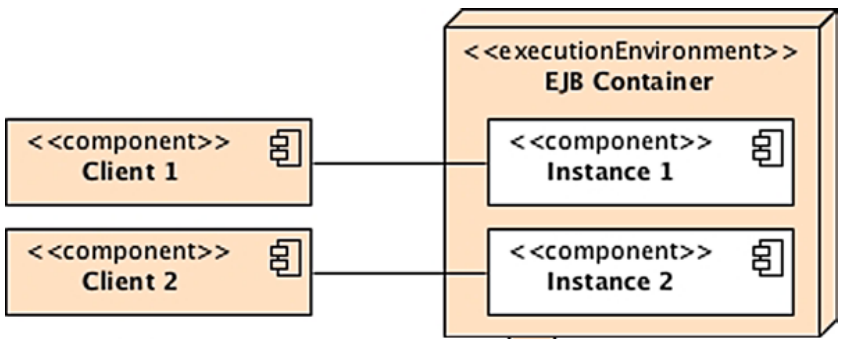


Figura 4.16: Interação entre clientes e EJB Stateful [2]

O contentor EJB cria uma instância por cada cliente, o que poderá ser um problema em termos de memória para sistemas que tenham muitos clientes. Os contentores EJB utilizam uma estratégia para contornar este problema de excesso de recursos em memória designado de Passivação e Activação.

- Passivação: é o processo de remover uma instância da memória e guaradar num local persistente.
- Activação: é o processo inverso, retira o estado do local persistente e restaura numa instância em memória.

Este processo é feito de forma automática pelo contentor EJB.

**Implementação do EJB Stateful demonstrado na listagem 4.29.**

```
1  @Stateful
2  public class EjbStateful {
3    private List<Produto> list = new ArrayList<>();
4
5    public void addToList(Produto p) {
6      if (!list.contains(p))
7      list.add(p);
8    }
9
10   @Remove
11   public void finishList() {
12     (...)
13     list.clear();
14   }
15 }
```

Listagem 4.29: Implementação de um EJB Stateless

Para construir este EJB foi apenas necessário marcar a classe com a anotação @Stateful. A anotação @Remove faz com que após a chamada do método finishList(), a instância seja removida permanentemente do contentor EJB.

#### 4.3.6.5 EJB Singleton

O EJB Singleton garante que cada classe tem uma única instância para toda a aplicação e fornece um acesso global para essa instância. Um EJB *Singleton* é útil quando é necessário partilhar estado com toda a aplicação.

As instâncias EJB *Stateful* são geridas pelo contentor de EJB criando uma instância por cada cliente como ilustra a figura 4.16.

A instância de um EJB *Singleton* é gerida pelo contentor de EJB que cria uma instância para servir todos os clientes como ilustra a figura 4.17.

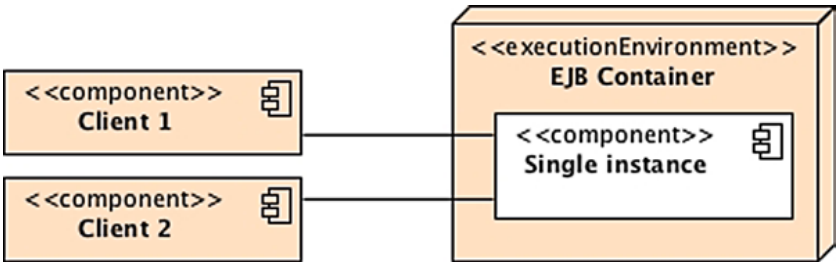


Figura 4.17: Interação entre clientes e EJB Singleton [2]

Como é possível ver na imagem anterior, podem existir N clientes, que todos eles vão partilhar a mesma instância e partilhar estado.

Um exemplo de um EJB Singleton poderá ser um caso onde seja necessário partilhar em toda a aplicação cada um dos produtos escolhidos por cada cliente. Para isso poderia ser criado uma cache recorrendo a um Map<Long, Produto>, onde estaria armazenado o número do cliente e o produto escolhido.

```
@Singleton
public class CacheProdutosEJB {
  private Map<Long, Produto> cacheProdutos = new HashMap
     <>();

  public void addProduto(Long idCliente, Produto prod) {
    if (!cacheProdutos.containsKey(idCliente))
      cacheProdutos.put(idCliente, prod);
  }

  public void removeProduto(Long idCliente) {
    if (cacheProdutos.containsKey(idCliente))
      cacheProdutos.remove(idCliente);
  }

  public Produto getProduto(Long idCliente) {
    if (cacheProdutos.containsKey(idCliente))
      return cacheProdutos.get(idCliente);
    else
      return null;
  }
}
```

Listagem 4.30: Implementação de um EJB Singleton

Para a implementação do EJB Singleton, apenas é necessário anotar a classe com @Singleton. Com isto, todos os clientes vão patilhar a mesma instância e neste caso, partilhar o estado cacheProdutos.

#### 4.3.6.6 EJB Message-driven beans

Os MDBs utilizam a estratégia da arquitectura MOM (Message-oriented middleware), fazendo com que a comunicação seja assíncona entre os componentes (portanto, que exista um fraco acopolamento temporal).
Na figura 4.18 é possível entender o funcionamento da arquitectura MOM.

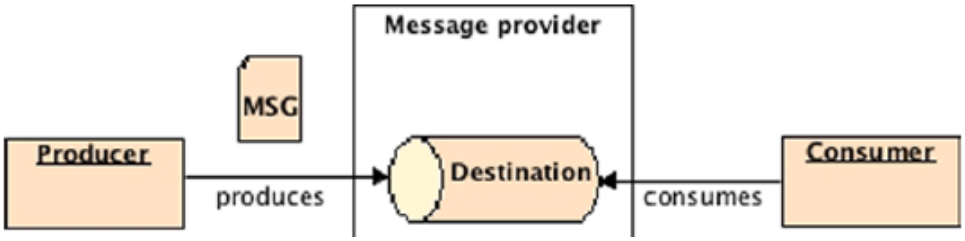


Figura 4.18: Arquitectura MOM [2]

- **Producer:** É o remetente da mensagem.
- **Destination:** A fila para onde a mensagem é enviada, ficando a aguardar ser consumida.
- **Consumer:** É o componente que está interessado em consumir a mensagem.
- **Message provider:** *Software* responsável por armazenar e enviar as mensagens.

Os MDBs (Message-driven beans), são consumidores de mensagens que são executados dentro do contentor de EJB. Tendo em conta que o contentor de EJB trata de tudo o que está relacionado com transações, segurança, concorrência entre outros, os MDBs focam-se apenas no consumo das mensagens.
Os MDBs não têm estado, e por esse motivo os contentores EJB podem ter várias instâncias de MDBs a consumir mensagens. Para um componente comunicar com um MDB tem que enviar uma mensagem para o destino de forma assíncrona, não podendo comunicar directamente.

O JMS é a API padrão do Java para a criação, envio, leitura ou consumo de mensagens assíncronas. Existem várias versões desta API (Classic API, Simplified API, Legacy API (P2P) ou Legacy API (Pub-Sub)). Irei estudar apenas a versão Classic API do JMS.

A versão Classic API do JMS pode ser entendida com recurso à figura 4.19.

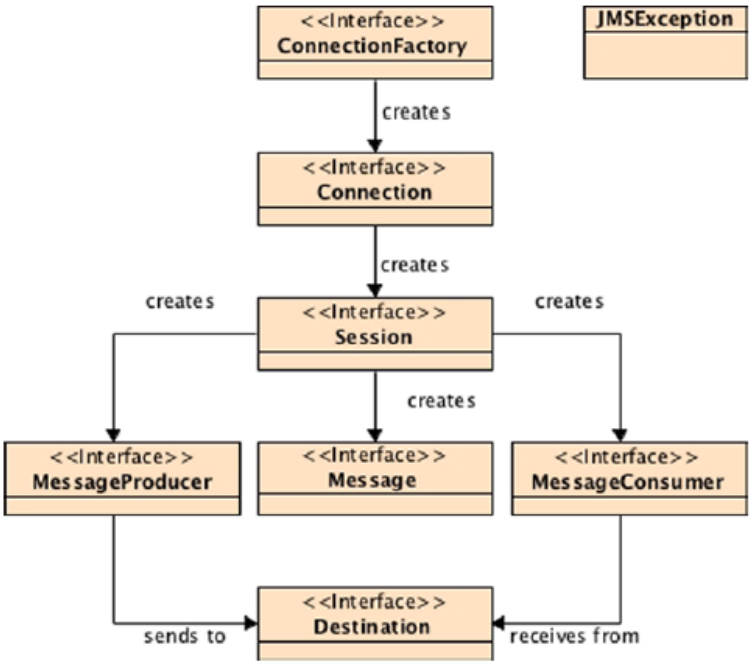


Figura 4.19: API Classic do JMS [2]

O ConnectionFactory é utilizado para criar uma conexão com o proveedor que armazena as mensagens. Após obter a Connection, é criada uma sessão e dessa forma existirá uma única thread para a troca de mensagens. Posteriormente é criado o produtor de mensagens da sessão indicando o destino.

#### 4.3.6.7 Estrutura da escrita de um Message-driven bean

Para exemplificar a forma como se constrói um MDB foi utilizado o Glassfish como proveedor de mensagens. Para o exemplo foi criado um Connection Factory denominado de "jms/exemploConnectionFactory"e foi criado como destino uma Queue denominada de "exemploQueue".

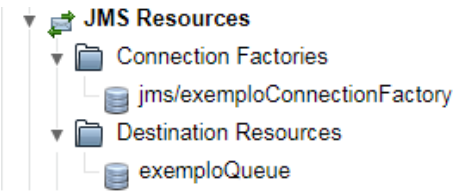


Figura 4.20: JMS no Glassfish

**Produtor da mensagens:** Para testar o MDB foi utilizado um simples Session Bean que apresento na listagem 4.31

```
@Stateless
public class TesteProdutorSessionBeans {

  @Resource(mappedName="exemploQueue")
  private Queue dest;

  @Resource(mappedName="jms/exemploConnectionFactory")
  private ConnectionFactory queue;

  @Override
  public void metodo1() {
    Connection connect = null;
    try {
      connect = queue.createConnection();
      Session session = connect.createSession(true, 0);
      MessageProducer producer = session.createProducer(dest
          );
      MapMessage message = session.createMapMessage();
      message.setInt("idCliente", 123);
      message.setInt("idEncomenda", 2222);
      producer.send(message);
    } catch (Exception je) {
      if (connect != null) {
        try {
          connect.close();
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    }
  }
}
```

Listagem 4.31: Produtor de mensagens assíncronas

O produtor de mensagens utiliza a API Classic do JMS referida anteriormente. Foi utilizado injecção de depêndencias através da anotação @Resource para instânciar o ConnectionFactory e a Queue.

Para construir um MDB basta criar uma simples classe POJO, extender a interface MessageListener fazendo a implementação do método onMessage(Message message) e depois fazer uso das anotações do JEE.

Vejamos o exemplo da listagem 4.32.

```
@MessageDriven(
  activationConfig = { @ActivationConfigProperty(
  propertyName = "destination", propertyValue = "
     exemploQueue"), @ActivationConfigProperty(
  propertyName = "destinationType", propertyValue = "javax.
     jms.Queue")
  },
  mappedName = "exemploQueue")
public class mdbExemplo implements MessageListener {

  public mdbExemplo() {
  }

  public void onMessage(Message message) {
    try {
      MapMessage msg = (MapMessage) message;
      int idEncomenda =msg.getInt("idEncomenda");
      int idCliente = msg.getInt("idCliente");
      System.out.println("Teste de: " + idCliente + " para
         encomendar " + idEncomenda);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Listagem 4.32: Message-driven bean (MBD)

Como é possível ver no exemplo da listagem 4.32, o MDB obtém informação de qual a fila que irá estar à escuta através da informação passada na anotação @MessageDriven que idêntifica o tipo da fila, no caso Queue e o nome da mesma. Desta forma, sempre que essa fila receba uma mensagem o método onMessage irá ser chamado.

#### 4.3.6.8 Concorrência

A concorrência apenas terá de ser tida em conta nos EJBs do tipo *Singleton* já que são os únicos que permitem acesso em simultâneo de vários clientes a uma mesma instância.

O controlo da concorrência pode ser feita ao nível do contentor EJB ou do próprio componente EJB.

- **Controlo ao nível do contentor**: Quando o controlo é feito do lado do contentor, é possível utilizar a anotação @Lock com os valores READ para partilhado ou WRITE para exclusivo.

```
1  @Singleton
2  @Lock(LockType.WRITE)
3  public class CacheProdutosEJB {
4    private Map<Long, Produto> cacheProdutos = new
       HashMap<>();
5
6    (...)
7
8    @Lock(LockType.READ)
9    public Produto getProduto(Long idCliente) {
10     if (cacheProdutos.containsKey(idCliente))
11       return cacheProdutos.get(idCliente);
12     else
13       return null;
14   }
15 }
```

Listagem 4.33: Controlo ao nível do contentor EJB Singleton

No exemplo apresentado na listagem 4.33, é possível ver que a anotação @LOCK pode ser colocada tanto ao nível da classe como ao nível dos métodos ou em ambos.
Se colocado apenas ao nível da classe, todos os métodos vão seguir o mesmo nível de controlo. Se colocado ao nível do método, apenas esse método ficará com esse nível de controlo.

Ou seja, neste exemplo tendo em conta que a classe está marcada com @Lock exclusivo, todos os métodos à excepção do "getProduto"são exclusivos. Significa que um cliente ao chamar um desses métodos só irá ganhar acesso se mais nenhum estiver a aceder ao método. No caso do método "getProduto"que está marcado com @Lock partilhado, todos os clientes podem acedem em simultâneo.

- **Controlo ao nível do componente**: Neste caso o contentor permite acesso total à instância, ficando a responsabilidade da sincronização do lado do componente EJB.
  Para este tipo de controlo, a classe deve ser marcada com a anotação @ConcurrencyManagement e com o valor BEAN. Os métodos onde existe necessidade de controlo de concorrência deve ser garantida a sua sincronização utilizando primitivas como synchronized ou volatile.

```
1  @Singleton
2  @ConcurrencyManagement(ConcurrencyManagementType.BEAN
       )
3  public class CacheProdutosEJB {
4    private Map<Long, Produto> cacheProdutos = new
       HashMap<>();
5
6    public synchronized void addProduto(Long idCliente,
       Produto prod) {
7      if (!cacheProdutos.containsKey(idCliente))
8        cacheProdutos.put(idCliente, prod);
9    }
10
11   public Produto getProduto(Long idCliente) {
12     if (cacheProdutos.containsKey(idCliente))
13       return cacheProdutos.get(idCliente);
14     else
15       return null;
16   }
17 }
```

Listagem 4.34: Controlo ao nível do componente EJB Singleton

No exemplo da listagem 4.34, é possível ver que o método addProduto que necessita de controlo de concorrência utiliza a primitiva synchronized para

garantir que os dados são sincronizados.

#### 4.3.6.9 EJB síncrono vs assíncronos

**EJB síncrono:** Por omissão os EJBs são síncronos, isto significa que se um cliente executar um método que demora muito tempo a terminar a sua tafefa, o cliente irá ficar bloquado até que o método retorne.

```
1  @Stateless
2  public class EjbStateless {
3
4    public void metodo1() {
5      Thread.sleep(10000);
6    }
7
8    public int metodo2() {
9      Thread.sleep(10000);
10     return 1;
11   }
12 }
```

Listagem 4.35: EJB síncrono caption

No exemplo ilustrado pela listagem 4.35 temos um EJB síncrono, o cliente ao invocar o "metodo1"ou "metodo2"irá ficar bloqueado durante 10 segundos.

**EJB assíncrono:** Embora por omissão os EJBs sejam síncronos é possível criar EJBs assíncronos. Para isso deve ser utilizado a anotação @Asynchronous. A anotação @Asynchronous pode ser colocada ao nível da classe, indicado que todos os métodos são assíncronos ou ao nível dos métodos.
Desta forma, mesmo que o cliente chame um método que tenha um tempo de execução elevado o cliente não ficará bloqueado, já que se esse método for assíncrono o contentor irá retornar o controlo de imediato ao cliente e irá executar o trabalho demorado numa outra thread.

De referir que as chamadas assíncronas não têm de retornar obrigatóriamente void, podem retornar também java.util.concurrent.Future<V>, onde o V é o valor e Future permite obter o valor que está a ser executado noutra *thread*.

```
@Stateless
public class EjbStateless {

  @Asynchronous
  public void metodo1() {
    Thread.sleep(5000);
  }

  @Asynchronous
  public Future<Integer> metodo2() {
    Thread.sleep(5000);
    return AsyncResult<>(1);
  }
}
```

Listagem 4.36: EJB assíncrono

No exemplo da listagem 4.36 temos dois métodos assíncronos, assim sendo o cliente ao chamar qualquer um dos métodos não irá ficar bloqueado e poderá seguir o seu trabalho.
Em relação ao metodo2, o cliente poderá guardar o valor retornado a quando da chamada do metodo2, e consultar posteriormente. Desta forma o método2 poderá já ter retornado e desta forma o cliente não terá de ficar 5 segundos à espera que o método2 termine.

```
final Future<Integer> teste = ejbStateless.metodo2();
//trabalho que demora 4 segundos
assertEquals(1, teste.get());
```

Listagem 4.37: EJB assíncrono Future<?>

Como é possível ver na listagem 4.37, quando o cliente obtém o retorno do método2, só terá de aguadar 1 segundo.

#### 4.3.6.10 Transacções

Os contentores de transacções contêm um gestor de transacções que usa JTA e JTS. As transacções são naturais para os EJBs, por omissão cada método é encapsolado numa transacção e gerido pelo contentor EJB. Desta forma o desenvolvedor não tem de se preocupar com transacções, esse trabalho será feito pelo

contentor de EJB. No entanto, se o desenvolvedor pretender pode gerir as transacções programaticamente, ficando com a responsabilidade de iniciar e terminar transacções.

**Transacções geridas pelo contentor EJB:**
Nesta abordagem o contentor de EJB é responsável pela demarcação transaccional. O contentor de EJB permite as seguintes formas de demarcação transaccional para os *Session beans*:

- **REQUIRED** - Esta opção faz com que o método seja sempre executado dentro de uma transacção. Se o método tiver sido chamado por um cliente que tem um contexto transaccional, o médodo será invocado dentro da transacção do cliente. Se for chamado fora do contexto transaccional, será criada uma nova trasancção para o efeito. (Este é o comportamento por omissão dos beans).
- **REQUIRES_NEW** - Esta opção cria sempre uma nova transacção independentemente do contexto em que é chamada. Se for chamada num contexto transaccional, cria uma nova transacção e quando treminar faz *commit* ou *rollback* e volta à transacção do cliente. Esta opção deverá ser utilizada quando não se pretende que o método interfira com a transacção iniciada pelo cliente.
- **SUPPORTS** - Utiliza o contexto transaccional do cliente que utilizou o bean. Se o cliente tiver chamado no contexto de uma transacção, propaga a transacção do cliente, caso contrário não utiliza transacção.
- **MANDATORY** - Obriga a que o cliente que invoca o método tenha um contexto transaccional associado, caso contrário gera uma excepção.
- **NOT_SUPPORTED** - Não permite que o método seja executado por um cliente com transacção iniciada.Se o cliente tiver uma transacção iniciada o contentor de EJB irá suspender a transacção do cliente durante a execução do método.
- **NEVER** - Não permite que o cliente que executa o método tenha uma transacção iniciada. Se isso acontecer, irá lançar uma excepção.

No caso dos EJBs do tipo MDB apenas suportam os tipos de demarcação REQUIRED ou NOT_SUPPORTED. Nos MDBs é impossível obter o contexto transaccional dos clientes, ninguém invoca explicitamente o método onMessage do MDB.

Nos MDBs ao definir o tipo de demarcação REQUIRED ou NOT_SUPPORTED, apenas se está a informar o contontor de EJBs que caso se chame um Session Bean dentro do onMessage este deverá, ou não, ser executado no contexto transaccional do MDB.

Na listagem 4.38 é apresentado um EJB com o tipo de transacção REQUIRED onde quem é responsável pela demarcação transaccional é o contentor EJB.

```
1  @Stateless
2  @TransactionAttribute(TransactionAttributeType.REQUIRED)
3  public class ExemploEJB {
4
5    @EJB
6    private OutroEJB outroEJB;
7
8    public Pessoa createPessoa(Pessoa p) {
9      addLocalInfo(p);
10     outroEJB.addInfo(p);
11     return p;
12   }
13
14   public void addLocalInfo(Pessoa p) {
15     (...)
16   }
17 }
```

Listagem 4.38: EJB com tipo de transacção REQUIRED

De realçar que embora no exemplo da listagem 4.38 esteja presente de forma explícita o tipo de transacção REQUIRED, não seria necessário colocar esta instrução já que é o comportamento utilizado por omissão.

Como é possível ver, a classe foi marcada com a anotação @TransactionAttribute, fazendo com que todos os métodos da classe fiquem com o tipo de demarcação indicado. No entanto, é possível colocar métodos com outro tipo de demarcação dentro da mesma classe, para isso os métodos onde se pretende outro comportamento devm ser anotados com @TransactionAttribute.

No exemplo da listagem 4.38 temos a garantia que as operações executadas no método "createPessoa"são executadas no contexto de uma transacção graças ao contentor de EJB que faz toda a gestão de forma transparente para o desenvolvedor, conforme é possível ver no diagrama de sequência da figura 4.21.

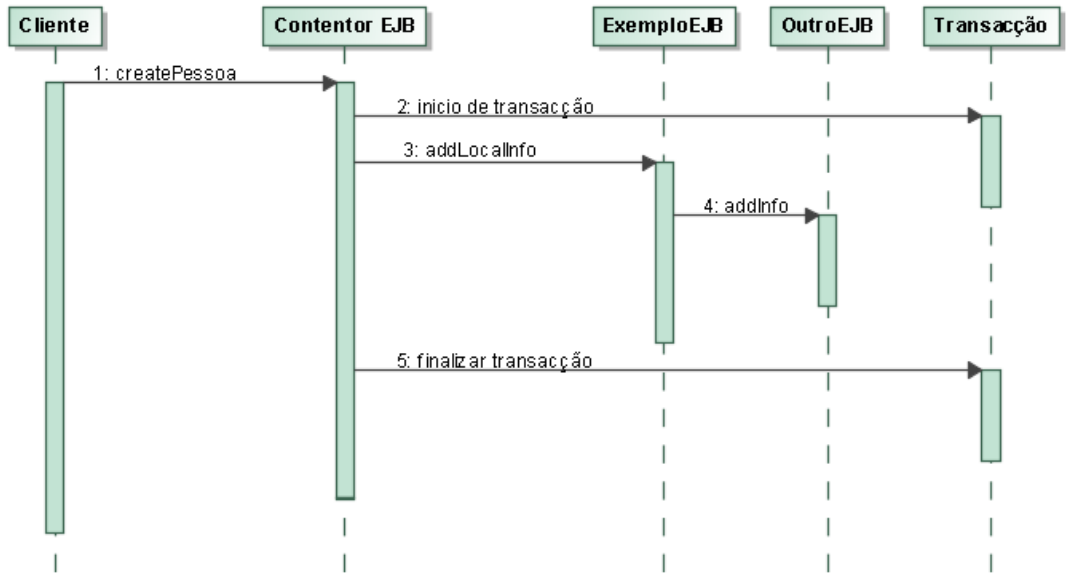


Figura 4.21: Diagrama de sequência de transacção

O comportamento ilustrado pelo diagrama de sequência da figura 4.21, é o comportamento padrão das transacções geridas pelo contentos EJB. Quando o cliente invoca o método createPessoa o contentor EJB irá criar ou utilizar a transacção do cliente dependendo se o cliente invocou o método em contexto transaccional ou não. Para alterar este comportamento, basta colocar uma anotação no método createPessoa com um dos tipos de demarcação transaccional anteriormente mensionados.

Embora nesta abordagem o contentor de EJB torna transparente para o desenvolvedor a demarcação transaccional (faz de forma automática o *begin*, *commit* ou *rollback* das transacções), o desenvolvedor pode sentir necessidade de fazer *rollback* de forma explícita em determinados contextos.

De seguida é apresentado um exemplo onde o desenvolvedor cria um EJB transaccional com o tipo de demarcação REQUIRED e no método1 faz rollback de forma explícita.

```
1  @Stateless
2  public class ExemploEJB {
3    @Resource
4    private SessionContext ctx;
5
6    public void metodo1(int flag) {
7      if (flag < 0)
8        ctx.setRollbackOnly();
9    }
10 }
```

Listagem 4.39: EJB transaccional, rollback explicito

Como é possível ver no exemplo da listagem **??**, o desenvolvedor não faz rollback directamente no EJB, o desenvolvedor tem de fazer rollback no contexto do EJB para informar o contentor que deve fazer rollback.
Para ir buscar o contexto de sessão, é utilizado injecção de depência com a anotação @Resource.

**Transacções geridas pelo desenvolvedor:**
Para fazer com que o contentor de EJB não faça a gestão de transacções de forma automática e deixe esse trabalho para o desenvolvedor, o EJB deve ser marcado com a anotação @TransactionManagement(TransactionManagementType.BEAN).

Para fazer a gestão das transacções de forma manual é utilizada a interface javax.transaction.UserTransaction. Esta interface é injectada pelo contentor de EJB, através da anotação @Resource.

Alguas das funcionalidades disponíveis por esta interface são:

- **begin** - Inicia uma nova transação associada à thread atual.
- **commit** - Confirma a transação actual.

- **rollback** - Indica que deve ser feito rollback da transação actual.
- **setRollbackOnly** - Marca a transacção actual para rollback.
- **getStatus** - Obtém o estado da transacção actual.
- **setTransactionTimeout** - Altera o tempo limite da transacção actual.

De seguida é apresentado um exemplo de como pode ser utilizado este tipo de gestão de transacções:

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class ExemploEJB {
  @Resource
  private UserTransaction trans;

  public void metodo1(int flag) {
    try {
      trans.begin();
      (...)
      if (flag >= 0)
        trans.commit();
      else
        trans.rollback();
    } catch (Exception e) {
      trans.rollback();
    }
  }
}
```

Listagem 4.40: EJB transaccional, rollback explicito

#### 4.3.6.11 Como utilizar EJB

Os clientes que pretendem utilizar um EJB não têm que estar a instânciar o EJB através do operador NEW. Os clientes podem utilizar injecção de dependências como é o caso do @EJB e @Inject ou utilizar o JINI para obter a instância para o EJB.

Na listagem 4.41 apresenta-se a definição de um simples EJB, onde estão definidas interfaces locais e remotas bem como acesso local ao EJB.

```
1 @Stateless
2 @Remote(ExemploEJBRemote.class)
3 @Local(ExemploEJBLocal.class)
4 @LocalBean
5 public class ExemploEJB implements ExemploEJBLocal,
     ExemploEJBRemote {...}
```

Listagem 4.41: Definição de um simples EJB Stateless

Para aceder ao EJB ExemploEJB, o cliente poderá utilizar umas das opções em baixo ilustradas:

- **@EJB:** Esta opção é especificamente destinada à injecção de instâncias de EJB no código do cliente.

```
1  @EJB ExemploEJB exemploEJB;
2  @EJB ExemploEJBLocal exemploEJBLocal;
3  @EJB ExemploEJBRemote exemploEJBRemote;
```

Listagem 4.42: Utilização de um EJB com @EJB

- **@Inject:** Utilizado para Injecção de depêndencia também poderá ser utilizado para instânciar EJBs.

```
1  @Inject ExemploEJB exemploEJB;
2  @Inject ExemploEJBLocal exemploEJBLocal;
3  @Inject ExemploEJBRemote exemploEJBRemote;
```

Listagem 4.43: Utilização de um EJB com @Inject

- **JINI:** Embora possa ser utilizado para EJB com acesso local, o JINI é mais utilizado quando um cliente está fora do contentor de EJBs e não pode utilizar a injecção de dependências. Neste caso é utilizado JINI para acessos remotos.

```
1  Context ctx = new InitialContext();
2  ExemploEJBRemote exemploEJBRemote = (ExemploEJBRemote
      ) ctx.lookup("java:global/EAP_AppGestao/EJB_Gestao
      /ClienteSessionBean!ejb.ExemploEJBRemote");
```

Listagem 4.44: Utilização de um EJB com JINI

### 4.3.7 Java Persistence API (JPA)

O JPA é uma ferramenta ORM (Object-Relational Mapping) que é utilizada para fazer a correspondência entre objectos e tabelas. Com o JPA é possível fazer o mapeamento de bases de dados relacionais em classes, objectos e atributos Java. Desta forma os desenvolvedores podem manter uma visão orientada aos objectos.

No package javax.persistence estão um conjunto de anotação que podem ser utilizadas para fazer o mapeamento entre ojectos e tabelas.

#### 4.3.7.1 Anotações JPA

Existe um evalado número de anotações que podem ser utilizador para garantir que as bases de dados são criadas com todas as restrições como estivessem a ser escritas em SQL.
Na figura 4.22 é apresentada uma imagem com a tabela Cliente, onde existe uma relação de um para muitos com encomendas.

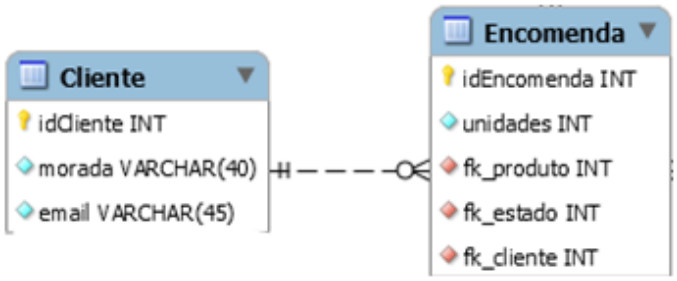


Figura 4.22: Modelo EA - tabela Cliente

Na listagem 4.45 é apresentado o código Java que representa a tabela Cliente.

```
1  @Entity
2  @NamedQuery(name="Cliente.findAll", query="SELECT c FROM
       Cliente c")
3  @Table(name="T_Cliente")
4  public class Cliente implements Serializable {
5    private static final long serialVersionUID = 1L;
6
7    @Id
8    private int idCliente;
9    @Column(length = 40, nullable = false)
10   private String email;
11   @Column(length = 45, nullable = false)
12   private String morada;
13
14   @OneToMany(mappedBy="cliente")
15   private List<Encomenda> encomendas;
16
17   // Constructors, getters, setters
18 }
```

Listagem 4.45: JPA - Entidade Cliente

**Anotações:**

- **@Entity:** Para que a classe seja idêntificada como sendo uma entidade e não uma simples classe POJO, deve ser marcada com a anotação @Entity.
- **@NamedQuery:** Com esta anotação é possível definir consultas estáticas em JPQL (Java Persistence Query Language) que posteriormente são traduzidas para SQL. Estas consultas são mais eficientes que as consultas dinâmicas.
- **@Table:** Permite indicar qual o nome da tabela na base de dados associada à entidade. Ou seja, neste exemplo embora o desenvolvedor trabalhe com o nome de entidade Cliente, quando for mapeado na base de dados será com no nome T_Cliente.

- **@Id:** Tendo em conta que o JPA é o mapeamento para tabelas relacionais, a anotação @Id indica que este campo será a chave primária da tabela.
- **@Column:** Permite definir as propriedades de cada coluna. Indicar a dimensão, nome, se permite null ou não entre outras propriedades na base de dados.
- **@OneToMany:** Indica o tipo de relação entre entidades, neste caso existe uma relação de um para muitos com a entidade Encomendas.

Como foi referido anteriormente, existe um evalado número de anotações que devem ser revistas para a construção de uma base de dados consistente.

#### 4.3.7.2 Persistence Unit:

A especificação JPA tem como requisito a definição de um ficheiro "XML"denominado de persistence unit na pasta "src/main/resources/". Este ficheiro contém toda a informação necessária para se conectar à base de dados (URL, driver JDBC, utilizador e senha). Permite definir outras propriedades como por exemplo a forma como o esquema da base de dados é gerado.
O elemento <provider> define o provedor de persistência, no nosso caso, o PersistenceProvider. No persistence-unit estão todas as entidades que devem ser geridas pelo gestor de entidades.

Na listagem 4.46 é apresentado um ficheiro onde estão definidas duas ligações a bases de dados diferentes.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/
   ns/persistence" xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp
   .org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/
   persistence/persistence_2_1.xsd">

  <persistence-unit name="GestaoPU" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.
       PersistenceProvider</provider>
    <jta-data-source>jdbc/gestao</jta-data-source>
    <class>model.Cliente</class>
    <class>model.Encomenda</class>
    <class>model.Estado</class>
  </persistence-unit>

  <persistence-unit name="EncomendasPU" transaction-type="
     JTA">
    <provider>org.eclipse.persistence.jpa.
       PersistenceProvider</provider>
    <jta-data-source>jdbc/encomendas</jta-data-source>
    <class>model.ProdutoEncomenda</class>
  </persistence-unit>

</persistence>
```

Listagem 4.46: Exemplo de PersistenceUnit.XML

No <jta-data-source> é utilizado o jdbc "jdbc/gestao"e "jdbc/encomendas"que foram previamente definidos no servidor Glassfish, como se mostra na figura 4.23.

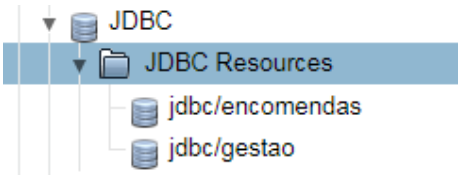


Figura 4.23: Glassfish JDBC

#### 4.3.7.3 JPA - Entity Manager

O Entity Manager (EM) é uma peça fundamental para o JPA. O EM gere o ciclo de vida das entidades bem como o seu estado, além de consultar entidades dentro de um contexto de persistência. O EM é responsável por criar e remover instâncias de entidades persistentes e encontrar entidades através da sua chave primária. Também pode definir se uma entidade tem um comportamento optimista ou pessimista no que respeita ao controlo de concorrência.

A instância de um Entity Manager pode ser obtida através de injecção de depências utilizando a anotação @PersistenceContext passando como atributos o nome da unidade de persistência (ficheiro XML), que foi abordado no ponto anterior.

De seguido é apresentado um exemplo de como instânciar um Entity Manager.

```
1 @PersistenceContext(unitName = "GertorPU")
2 private EntityManager em;
```

Listagem 4.47: ânciar EntityManager

Neste caso o Entity Manager irá ser instânciado com as propriedades que foram definidas no ficheiro "persistence.xml"dentro da tag «persistence-unit name="GestaoPU»".

#### 4.3.7.4 Utilização EJB com JPA

Na listagem 4.48 é apresentado um bean stateless que utiliza JPA para a persistência de informação.

```
1  @Stateless
2  public class ExemploEJB {
3    @PersistenceContext(unitName = "iselPU")
4    private EntityManager em;
5    public void create() {
6      Student s = new Student("Pedro Alberto", 32071, "MEIC");
7      Teacher t = new Teacher("Walter Vieira", 123);
8      em.persist(s);
9      em.persist(t);
10   }
11 }
```

Listagem 4.48: EJB com JPA

Desta forma temos a garantia que o método "create"irá fazer as duas inserções de forma atómica já que o contentor de EJB irá executar o método "create"no contexto de uma transacção.

Embora no exemplo apenas se demonstre o método persist do EntityManager existem outros métodos bastante úteis tais como:

- **void persist(Object entity):** Para persistir informação na base de dados.
- **void remove(Object entity):** Para remover informação na base de dados.
- **void lock(Object entity, LockModeType lockMode):** Para indicar o tipo de bloqueio no acesso a uma entidade. (Optimista ou pessimista)
- **boolean contains(Object entity):** Verifica se contém uma entidade.

# 5

# Protótipo

## 5.1 Enquadramento

Neste capítulo será apresentado um problema, que consiste na elaboração de um Sistema de Informação Empresarial que deverá ser resolvido utilizando as tecnologias Open Source e Microsoft estudadas ao longo do projecto, nomeadamente MYSQL e JEE 8 para Open Source e WCF e SQL Server para a Microsoft.

O problema apresentado, foi um trabalho final do ano lectivo 2016/17 da unidade curricular de Arquitectura de Sistemas de Informação (ASI), do terceiro semestre do Mestrado em Engenharia Informática do ISEL.

## 5.2 Projecto

A empresa ASITech, Lda dedica-se à venda de produtos de eletrónica doméstica. Os clientes realizam as suas encomendas através de computadores pessoais localizados em vários pontos de venda. Porque o número de clientes que podem aceder em simultâneo ao sistema é muito elevado, optou-se pela seguinte arquitetura física:

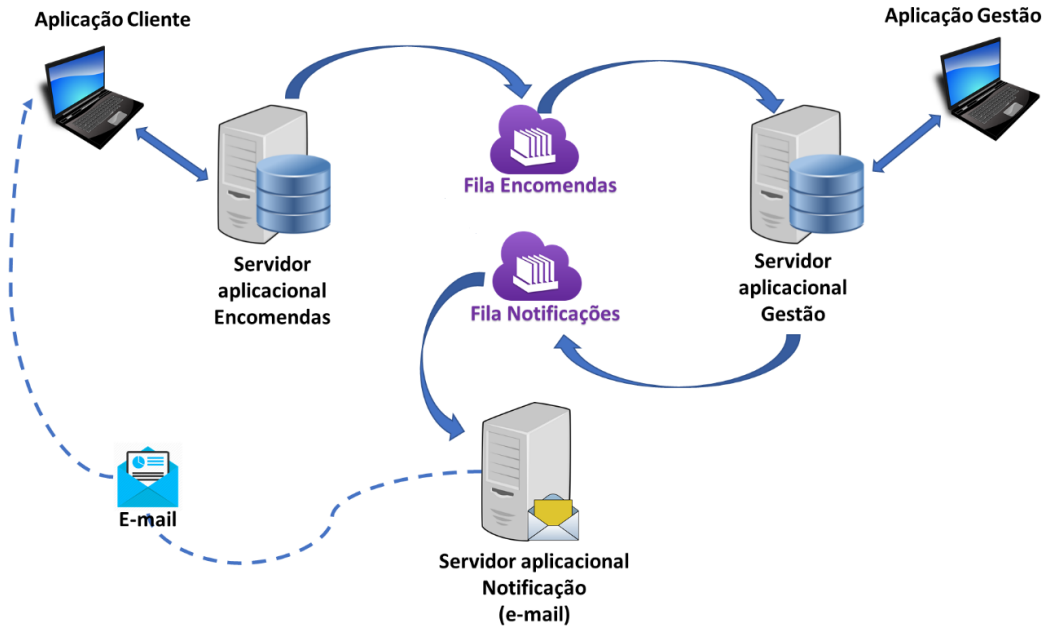


Figura 5.1: Arquitetura física

**Requisitos:**

O servidor aplicacional "Encomendas" aceita pedidos dos clientes para a realização de encomendas. Esses pedidos são fornecidos ao servidor aplicacional "Gestão" onde as encomendas são processadas. As mudanças de estado das encomendas originam notificações aos clientes que são enviadas ao servidor aplicacional "Notificação" que as enviará por e-mail para os clientes. O servidor "Encomendas" deve funcionar mesmo que os servidor "Gestão" e "Notificação" não estejam em funcionamento e o servidor "Gestão" deve funcionar mesmo que o servidor "Notificação" não esteja activo.

A informação sobre cada produto consta de um código, uma designação, um preço, uma quantidade disponível em stock e indicação do respectivo fornecedor. Por questões de eficiência, esta informação encontra-se fraccionada entre as os SGBDs "Encomendas" e "Gestão", sendo no primeiro relevante informação sobre código de produto, designação, preço e quantidade disponível (este último com carácter informativo apenas) e no segundo, informação sobre código de produto, preço, quantidade disponível e fornecedor).
Sobre cada fornecedor interessa manter o número de fornecedor, o nome e a morada, informação toda relevante apenas para efeitos de gestão.
A informação sobre cada cliente consta de um número de cliente (único), uma

morada e um endereço de e-mail. Por questões de eficiência, esta informação encontra-se replicada entre os dois SGBDS, devendo as duas réplicas estar sincronizadas.
Cada encomenda de um cliente deve ser associada a um estado de processamento (a processar, em expedição, expedida, impossível de satisfazer) sendo armazenada permanentemente no SGBD "Gestão", embora a sua informação inicial possa estar temporariamente armazenada na Fila Encomendas.

Para além das aplicações que servem os clientes da empresa, existe uma aplicação de gestão que deve permitir gerir clientes, realizar encomendas a fornecedores, aceitar entregas, etc. Esta aplicação utiliza os serviços do servidor aplicacional "Gestão".

Os clientes não têm garantia de que os produtos encomendados serão fornecidos (por não existir a quantidade pretendida em stock) mas deverão ser notificados do resultado do processamento das encomendas.

O sistema deve manter uma lista de perguntas frequentes e respetivas respostas. Cada pergunta é caraterizada por um identificador único (atribuído automaticamente), data de colocação e um texto relativo à pergunta. Cada resposta é caracterizada por um identificador único (atribuído automaticamente), pela referência para a pergunta a que respeita, pela data de colocação, por uma indicação de que a resposta foi dada pela empresa ou por um cliente e pelo texto da resposta. As perguntas apenas podem ser inseridas no SGBD Encomendas, mas as respostas podem ser inseridas quer no SGBD Encomendas (respostas de clientes), quer no SGBD Gestão (respostas da empresa). Pretende-se que em ambos os servidores coexista a informação sobre perguntas e respostas, mas que as manipulações desta informação em cada um dos locais seja feita com total autonomia, razão porque se admite que em alguns intervalos (em geral, pequenos) os dois sistemas possam estar fora de sincronismo.

### 5.2.1 Requisitos do sistema

Tabela 5.1: Requisitos do sistema

| Requisitos | Descrição |
|---|---|
| RQ01 | Servidor aplicacional “Encomendas” estará preparado para receber pedidos síncronos por parte dos clientes para realizar encomendas; |
| RQ02 | O servidor aplicacional “Gestão” estará preparado para receber pedidos assíncronos por parte do servidor aplicacional “Encomendas” e processar os mesmos; |
| RQ03 | O servidor aplicacional “Notificação” estará preparado para receber pedidos assíncronos por parte do servidor aplicacional “Gestão” e processar os mesmos; |
| RQ04 | Todas as mudanças de estado das encomendas irão originar uma notificação de email para o cliente; |
| RQ05 | O servidor aplicacional “Encomendas” deverá funcionar mesmo que os servidor aplicacional “Gestão” e “Notificação” não estejam em funcionamento; |
| RQ06 | O servidor aplicacional “Gestão” deverá funcionar mesmo que o servidor aplicacional “Notificação” não esteja a funcionar; |
| RQ07 | Informação de produto deverá estar fracionada nos SGBD "Encomendas"e "Gestão"; |
| RQ08 | No SGBD "Encomendas"a informação relativa ao stock será meramente informativa; |
| RQ09 | Informação relativa a clientes deverá estar replicada sincronamente nos SGBD’s "Encomendas" e "Gestão"; |
| RQ10 | Os clientes não têm garantia de que os produtos encomendados serão fornecidos (por não existir a quantidade pretendida em stock); |
| RQ11 | Os SGBD’s "Encomendas" e "Gestão" devem armazenar perguntas frequentes e respectivas respostas. Essa informação deverá coexistir em ambos os servidores. Admite-se que em alguns intervalos (em geral, pequenos) os dois sistemas possam estar fora de sincronismo. |
| RQ12 | As perguntas apenas podem ser inseridas no SGBD Encomendas; |
| RQ13 | As respostas podem ser inseridas quer no SGBD Encomendas (respostas de clientes), quer no SGBD Gestão (respostas da empresa); |

### 5.2.2 Camada de Dados (CD)

#### 5.2.2.1 Esquema Lógico Global

Após o levantamento de todos os dados necessários para o modelo de dados e assumindo que:

- Um produto só tem um fornecedor;
- Uma encomenda só contém um produto específico;
- Só será guardado o último estado da encomenda (Sem histórico);

Foi elaborado o seguinte esquema global:

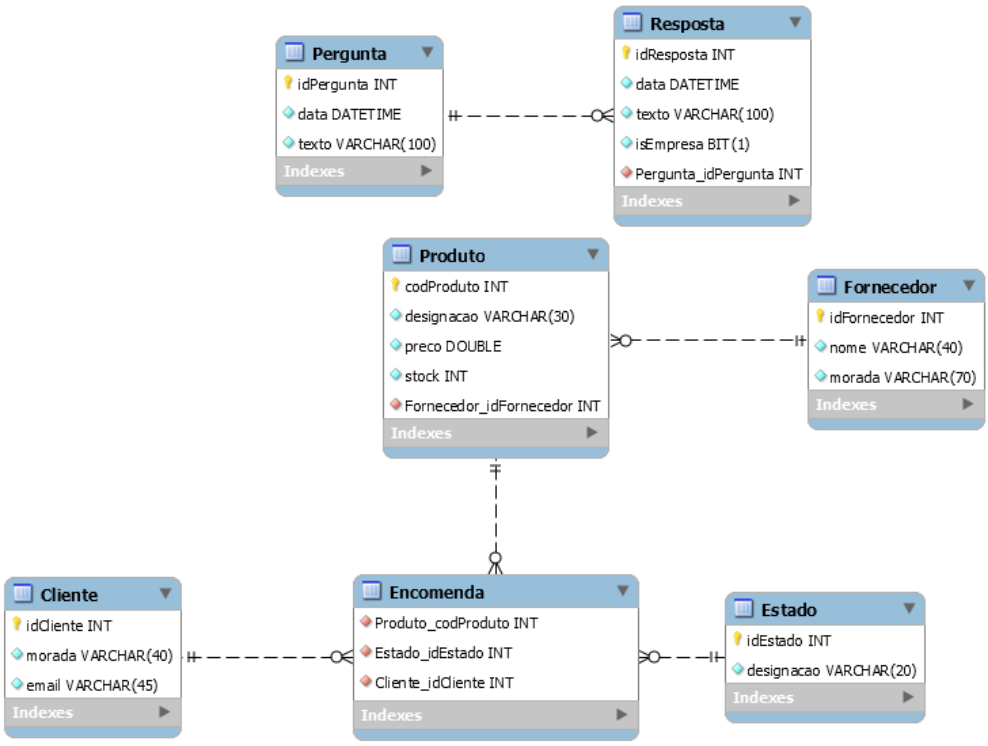


Figura 5.2: Esquema lógico global

Com o esquema lógico global acima ilustrado, é possível ficar com uma visão global de todas as entidades e atributos necessários para o desenvolvimento da camada de dados.

#### 5.2.2.2 Esquema de Fragmentação

Segundo o requisito [RQ07], a informação de produto deverá estar fraccionada nos SGBDs "Encomendas" e "Gestão", sendo no primeiro relevante informação sobre código de produto, designação, preço e quantidade disponível (este último com carácter informativo apenas) e no segundo, informação sobre código de produto, preço, quantidade disponível e fornecedor), foi necessário fazer uma fragmentação vertical para deixar em cada instância os atributos relevantes para esse contexto.

Tendo em conta que a informação de stock e preço é relevante para os SGBDs "Encomendas" e "Gestão", mas que apenas é actualizada no SGBD de "Gestão", foi feita uma separação destes dois atributos numa nova tabela designada de ProdutoInfo que existirá em ambos os SGBDs. Para os restantes atributos foi feita uma fragmentação vertical dando origem ao seguinte esquema de fraccionamento:

- **ProdutoEncomendas** (idProduto, designacao)
- **ProdutoGestao** (idProduto, unidades_encomendadas, encomendado)
- **ProdutoInfo** (codProduto, preco, stock)

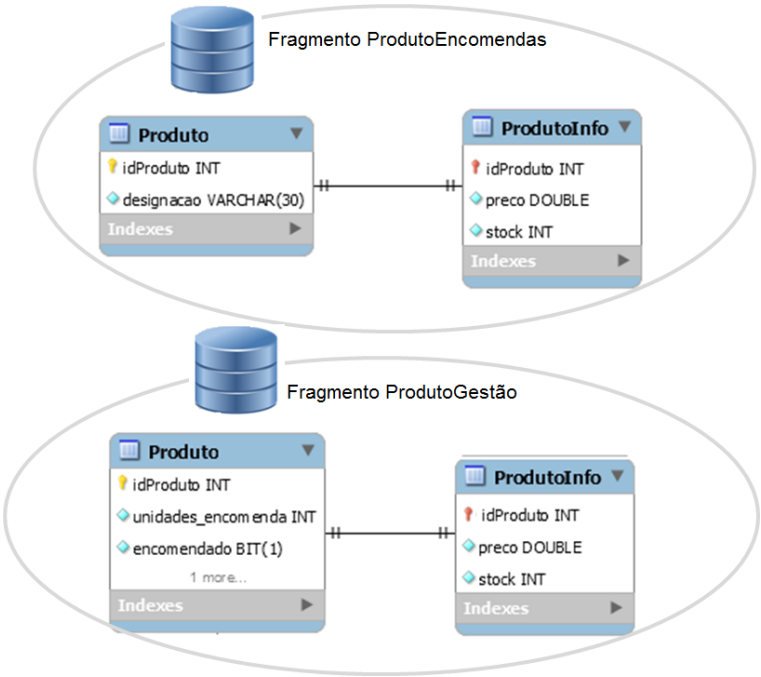


Figura 5.3: Fragmentação produto

#### 5.2.2.3 Esquema de distribuição

Nesta ponto serão apresentados os servidores e os respectivos esquemas lógicos que foram utilizados para o desenvolvimento da arquitectura proposta e desse modo responder aos requisitos solicitados para o desenvolvimento do sistema.

- **Nó onde estará o SGBD de Encomendas:**

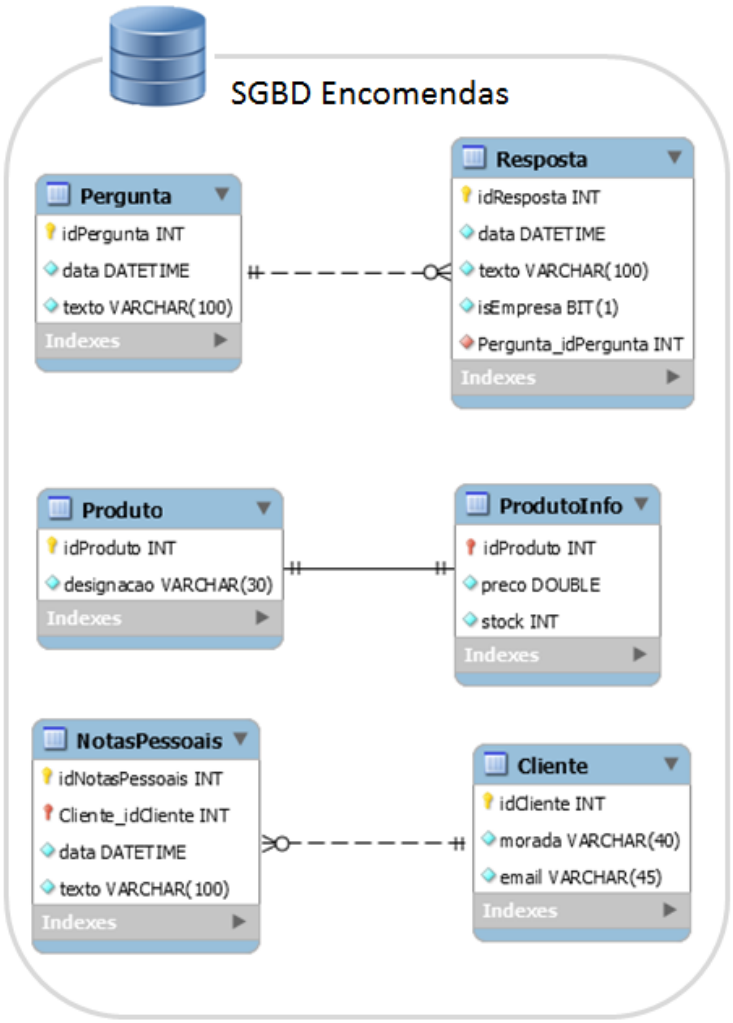


Figura 5.4: Esquema lógico SGBD Encomendas

- **Nó onde estará o SGBD de Gestão:**

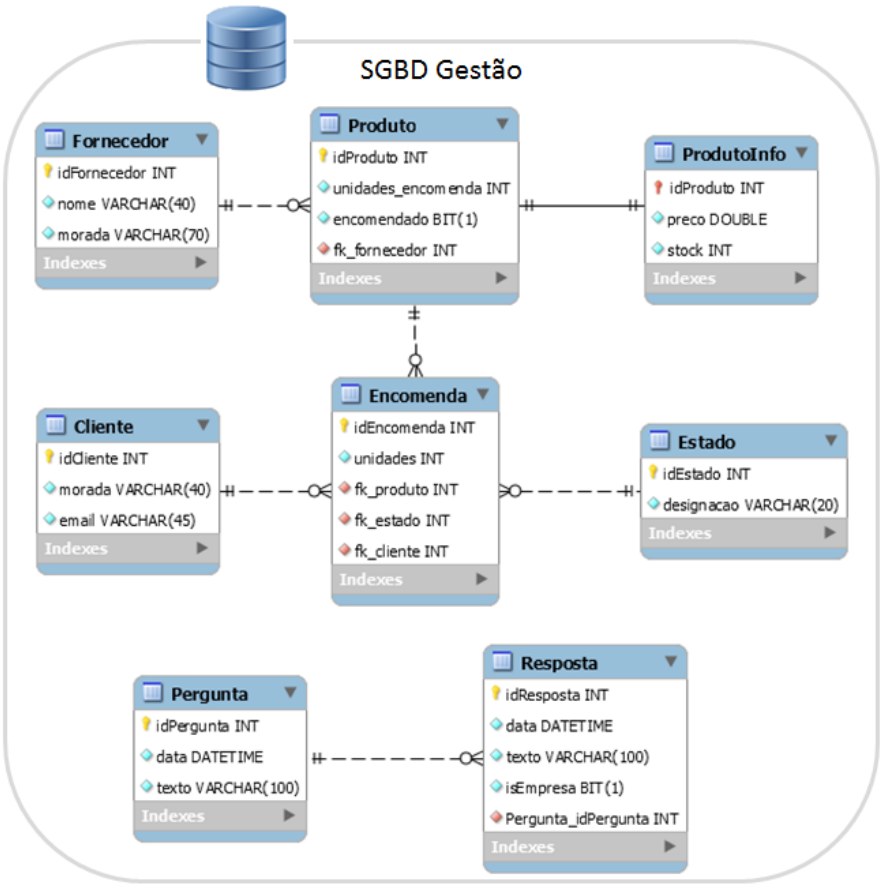


Figura 5.5: Esquema lógico SGBD Gestão

#### 5.2.2.4 Replicação de dados

Neste ponto vão ser apresentados os tipos de replicação que foram utilizados por forma a implementar o sistema, respeitando os requisitos do sistema.

- **Replicação produto:** Para a replicação de produto, foi necessário ter em conta os requisitos de sistema:
  - Os produtos apenas podem ser inseridos no SGBD de Gestão;
  - A informação de produto apenas pode ser actualizada no SGBD de Gestão.

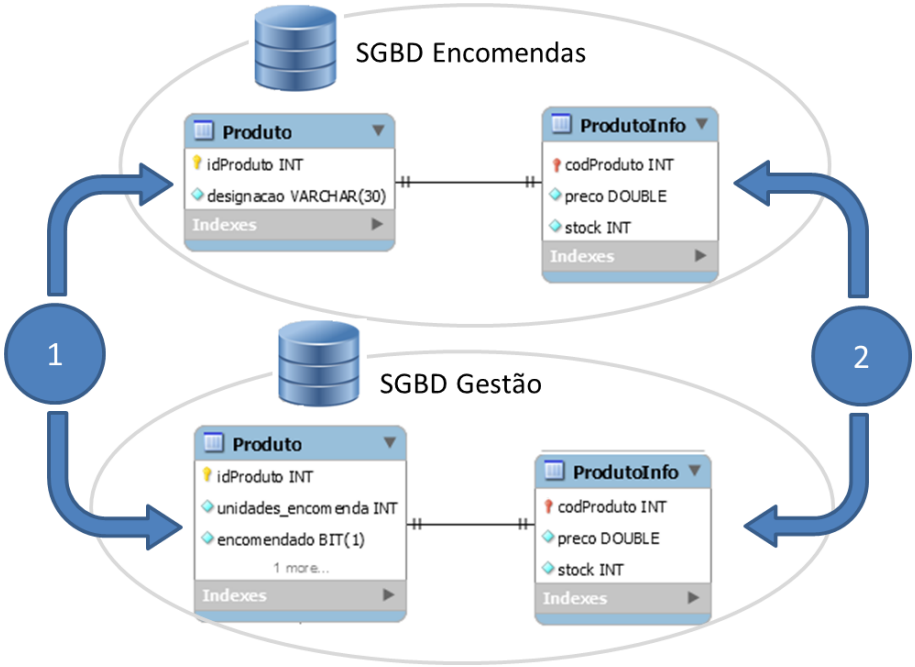


Figura 5.6: Replicação produto

**Solução MYSQL:**

1. A informação das tabelas de Produto são replicadas sincronamente através de transacções distribuídas geridas pela camada aplicacional.
2. A informação das tabelas de ProdutoInfo são replicadas de forma assincrona. Tendo em conta que a informação pode ser actualizada apenas no SGBD Gestão foi utilizada uma replicação do tipo *Master-Slave*.

**Solução SQL Server:**

1. A informação das tabelas de Produto são replicadas sincronamente através de um procedimento armazenado que utiliza transacções distribuídas geridas na camada de dados.
2. A informação das tabelas de ProdutoInfo são replicadas de forma assíncrona. Tendo em conta que a informação pode ser actualizada apenas no SGBD Gestão e o stock é meramente teórico foi utilizada uma replicação do tipo *Publisher-Subscriber*.

- **Replicação perguntas e respostas:** Para a replicação de perguntas e respostas, foi necessário ter em conta os requisitos de sistema:
  - As perguntas só são inseridas no SGBD de Encomendas;
  - As respostas podem ser inseridas no SGBD de Encomendas e Gestão.

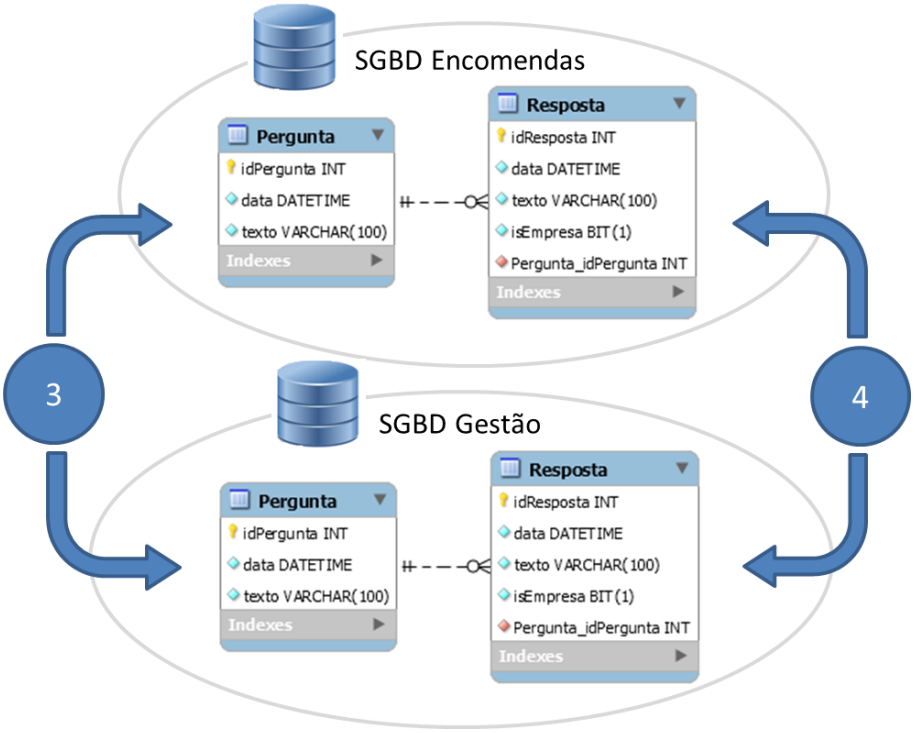


Figura 5.7: Replicação perguntas e respostas

**Solução MYSQL:**

3. A informação da tabela de Pergunta é replicada de forma assíncrona utilizando uma replicação do tipo *Master-Slave*, onde a tabela Pergunta do SGBD Encomendas tem o papel de *Master* e a tabela do SGBD Gestão tem o papel de *Slave*.
4. A informação da tabela de Resposta também é replicada de forma assíncrona, no entanto o tipo de replicação utilizado foi *Master-Master*, tendo em conta que é possível fazer actualizações em ambos os SGBDs.

**Solução SQL Server:**

3. A informação da tabela de Pergunta é replicada de forma assíncrona

utilizando *Publisher-Subscriber*, onde a tabela Pergunta do SGBD Gestão tem o papel de subscritor da tabela Pergunta do SGBD Encomendas que tem o papel de publicador.

4. A informação da tabela de Resposta também é replicada de forma assíncrona, no entanto o tipo de replicação utilizado foi *peer-to-peer*, tendo em conta que é possível fazer actualizações em ambos os SGBDs.

- **Replicação cliente:** Segundo os requisitos do sistema, a informação de cliente deverá estar sempre actualizada em ambos os servidores.

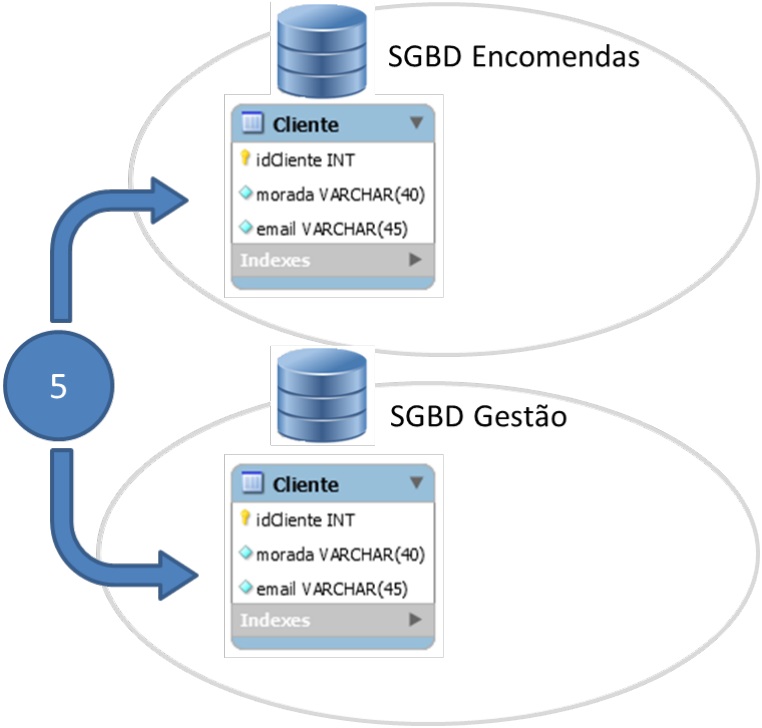


Figura 5.8: Replicação cliente

**Solução MYSQL:**

5. A informação da tabela de Cliente é replicada de forma síncrona recorrendo a transacções distribuídas ao nível da camada aplicacional.

**Solução SQL Server:**

5. A informação da tabela de Cliente é replicada de forma síncrona através de um procedimento armazenado que utiliza transacções distribuídas geridas na camada de dados, mas também poderia ter sido feita na camada aplicacional.

#### 5.2.2.5 Implementação SQL Server

Neste ponto serão apresentados os passos realizados na implementação da camada de dados em SQL Server.

1. Criação das bases de dados e respectivas tabelas em cada um dos servidores de acordo com os modelos EA demonstrados anteriormente.
2. Configurar cada um dos nós de Gestão e Encomendas para:
   - Aceitar acessos do exterior (Alterando as regras da Firewall do Windows).
   - Iniciar o serviço de Coordenador de Transações Distribuídas.
   - Permitir acesso ao Coordenador de Transações Distribuídas através da rede, tal como ilustra a figura 5.9.

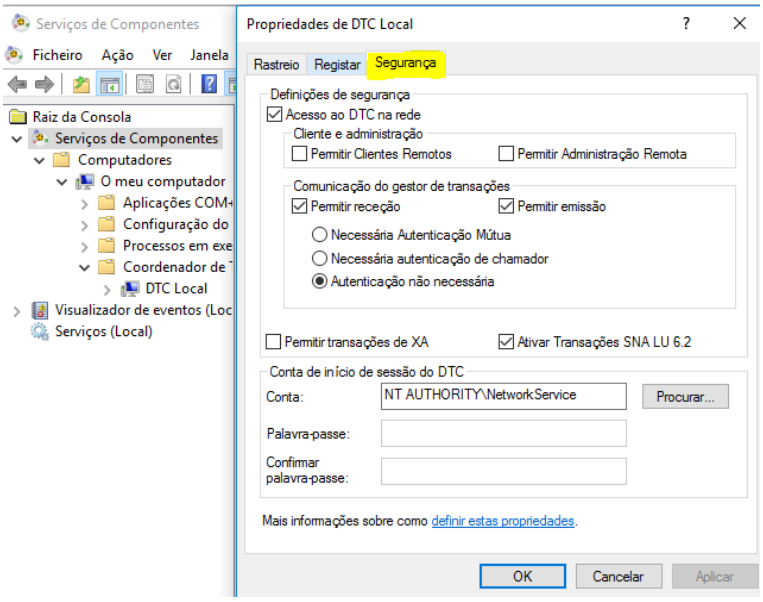


Figura 5.9: Configuração para acesso remoto ao DTC

- Alterar configuração do SQL Server para aceitar protoloco TCP/IP, tal como ilustra a figura 5.10.

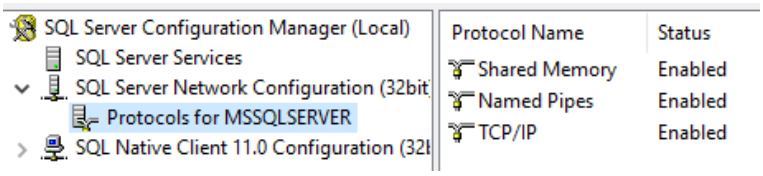


Figura 5.10: Configuração para protocolo TCP/IP

3. Criação dos *Linked Servers* e *Synonyms* em cada nó:

   - **Nó Encomendas:**

```
exec sp_addlinkedserver @server='Gestao',
   @srvproduct = 'SQL Server'

CREATE SYNONYM ClientesGestao for [Gestao].
   Gestao.dbo.Cliente
```

Listagem 5.1: Criação de Linked Servers e Synonyms no nó Encomendas

   - **Nó Gestão:**

```
exec sp_addlinkedserver @server = 'Encomendas',
   @srvproduct = 'SQL Server'

CREATE SYNONYM ProdutoEncomenda for [Encomendas
   ].Encomendas.dbo.Produto
CREATE SYNONYM ProdutoInfoEncomenda for [
   Encomendas].Encomendas.dbo.ProdutoInfo
```

Listagem 5.2: Criação de Linked Servers e Synonyms no nó Gestão

4. Criar os procedimentos armazenados "InserirCliente" e "ActualizarCliente" no nó Encomendas e os procedimentos armazenados "InserirProduto" e "ActualizarProduto" no nó Gestão utilizando Transacções distribuídas. Desta forma é garantida a replicação síncrona de Clientes e Produtos. Exemplo disso é a listagem 5.3.

```
CREATE PROCEDURE InserirCliente
(@morada varchar(40),@email varchar(45))
AS
  declare @idCli int
  begin try
    SET XACT_ABORT ON
    BEGIN DISTRIBUTED TRANSACTION
      INSERT INTO [dbo].[Cliente] ([morada],[email])
       VALUES (@morada, @email)
      set @idCli = (select idCliente from Cliente where
         [email]=@email)
      print @idCli
      INSERT INTO ClientesGestao ([idCliente],[morada],[
         email])
       VALUES (@idCli, @morada, @email)
    COMMIT TRANSACTION
  end try
  begin catch
    Rollback
  end catch
```

Listagem 5.3: Procedimento armazendo com transacções distribuídas

5. Criar uma replicação do tipo *Publisher-Subscriber* para fazer a replicação das tabelas de ProdutoInfo e Pergunta.
   A tabela ProdutoInfo do nó Gestão, terá o papel de publicador, no caso da tabela Pergunta, será a tabela do nó Encomendas que terá o papel de publicador.
   Para cada uma das tabelas foi feito o seguinte:

   - Definir o nó que terá o papel de distribuidor, indicar o local onde será gerada a BD distribuída e o ficheiro de log.
   - Definir o caminho onde serão guardados os *snapshots* do publicador.
   - Criar um novo publicador, atribuindo um nome para a publicação, indicando as tabelas que se pretendem replicar e as credênciais do Windows onde serão gerados os *snapshots*.
   - Criar um novo subscritor, indicando qual a publicação e de que nó se

pretende subscrever, indicar a base de dados do subscritor para onde serão replicados os dados, indicar as credênciais do Windows do nó onde serão gerados os *snapshots* e as credências do SQL Server do nó onde está o distribuidor.

Tal como é possível observar na figura 5.11 em cada servidor ficou configurado um publicador e um subscritor.

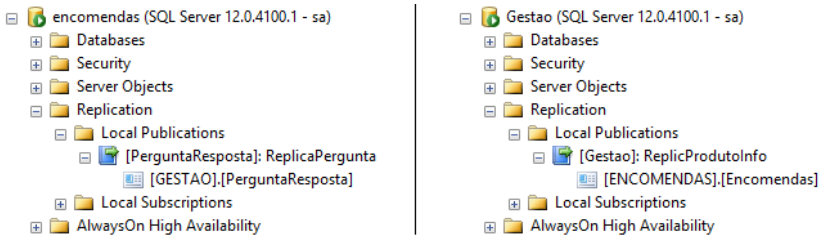


Figura 5.11: Replicação Publisher-Subscriber

6. Criar uma replicação do tipo *peer-to-peer* para fazer a replicação da tabela de Respostas. Para a criação deste tipo de replicação foi necessário ter em conta que:

   - A chave da tabela Resposta de cada um dos nós terá que ter um crescimento diferente para evitar colisões de chaves. Neste caso, a chave da tabela Respostas do nó Gestão, tem um crescimento de 2 em 2 e tem inicio em 0, ou seja, apenas admite número pares. Já a tabela Respostas do nó Encomendas, tem um crescimento de 2 em 2 e tem inicio em 1, ou seja, apenas admite número impares.
   - Se as tabelas de Resposta não tiverem tido inserções, pode ser criado o processo de replicação *peer-to-peer* sem utilizar nenhum *backup*, caso já tenham sido feitas alterações, é necessário restaurar uma das tabelas com o *backup* da outra tabela, para garantir que o processo de replicação é inciado com as tabelas alinhadas. Tal como é possível observar na figura 5.12 em cada servidor ficou configurado a replicação *peer-to-peer*.

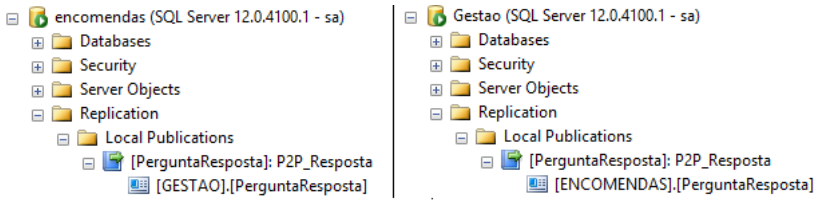


Figura 5.12: Replicação Peer-to-Peer

#### 5.2.2.6 Implementação MYSQL

Neste ponto serão apresentados os passos realizados na implementação da camada de dados em MYSQL.

1. Criação das bases de dados e respectivas tabelas em cada um dos servidores tal como demonstra a figura 5.13 de acordo com os modelos EA demonstrados anteriormente.

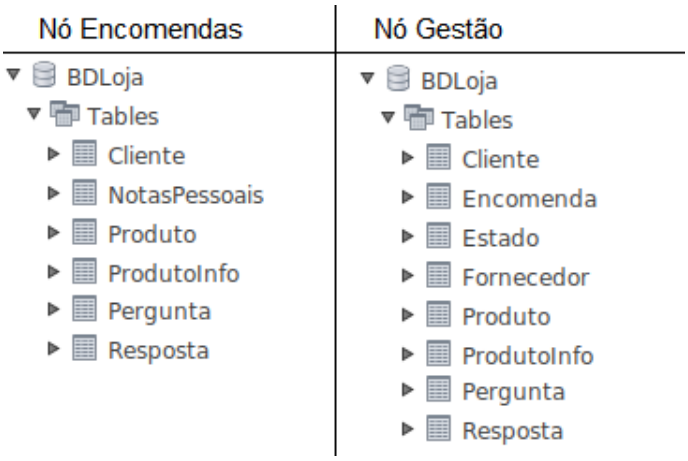


Figura 5.13: Bases de dados e tabelas nos nós MYSQL

2. Para a configuração da replicação *Master-Master* e *Master-Slave* foi necessário seguir os seguintes passos:

   - Parameterizar o ficheiro de configuração do MYSQL ("/etc/mysql/mysql .conf.d/mysqld.cnf") para permitir acessos desde o exterior, bem como indicar o local onde será gerado o ficheiro de log binário.

**Todos os Nós:**

```
1    bind-address = 0.0.0.0
2    federated
3    log_bin = /var/log/mysql/mysql-bin.log
4    expire_logs_days = 10
5    max_binlog_size = 100M
```

Listagem 5.4: Configuração para acessos remotos

- Parameterizar o ficheiro de configuração, indicando quais as tabelas ou bases de dados que devem ser replicadas ou ignoradas que estão no log binário.

**Nó Encomendas:**

```
1    server-id = 2
2    replicate_do_table = BDLoja.Resposta
3    replicate_do_table = BDLoja.ProdutoInfo
4    replicate_ignore_table = BDLoja.Pergunta
5    replicate_ignore_table = BDLoja.Produto
6    replicate_ignore_table = BDLoja.Cliente
7    replicate_ignore_table = BDLoja.Fornecedores
8    replicate_ignore_table = BDLoja.Encomendas
9    replicate_ignore_table = BDLoja.Estados
```

Listagem 5.5: Definir identificador do servidor Encomendas

**Nó Gestão:**

```
server-id = 1
replicate_do_table = BDLoja.Pergunta
replicate_do_table = BDLoja.Resposta
replicate_ignore_table = BDLoja.Cliente
replicate_ignore_table = BDLoja.NotasPessoais
replicate_ignore_table = BDLoja.Produto
replicate_ignore_table = BDLoja.ProdutoInfo
```

Listagem 5.6: Definir identificador do servidor Gestão

3. Criar em cada nó um utilizador com permissão para fazer a replicação dos dados.

```
GRANT REPLICATION SLAVE ON *.* TO 'user_replica'@'%'
   IDENTIFIED BY 'P4ssw0rd!';
```

Listagem 5.7: Criar utilizador com permissão para fazer a replicação

4. No Nó Gestão executar a instrução da listagem 5.8 para obter informação do nome do ficheiro de log binário e da sua posição para utilizar no No Encomendas.

```
show master status;
# Output-File = mysql-bin.000001
# Output-Position = 10565
```

Listagem 5.8: Obter nome do log file e posição no Nó Gestão

5. No Nó Encomendas, iniciar a replicação colocando a informação obtida na instrução da listagem 5.8. Para iniciar a replicação deve ser executada a instrução da listagem 5.9.

```
STOP SLAVE;
CHANGE MASTER TO MASTER_HOST = '192.168.56.105',
MASTER_USER = 'user_replica', MASTER_PASSWORD = '
   P4ssw0rd!',
MASTER_LOG_FILE = 'mysql-bin.000001',
MASTER_LOG_POS = 10565;
START SLAVE;
```

Listagem 5.9: Iniciar replicação no Nó Encomendas

6. No Nó Encomendas executar a instrução da listagem 5.10 para obter informação do nome do ficheiro de log binário e da sua posição para utilizar no Nó Gestão.

```
1 show master status;
2 # Output-File = mysql-bin.000001
3 # Output-Position = 9075
```

Listagem 5.10: Obter nome do log file e posição no Nó Encomendas

7. No Nó Gestão, iniciar a replicação colocando a informação obtida na instrução da listagem 5.10. Para iniciar a replicação deve ser executada a instrução da listagem 5.11.

```
1 STOP SLAVE;
2 CHANGE MASTER TO MASTER_HOST = '192.168.56.102',
3 MASTER_USER = 'user_replica',
4 MASTER_PASSWORD = 'P4ssw0rd!',
5 MASTER_LOG_FILE = 'mysql-bin.000001',
6 MASTER_LOG_POS = 9075;
7 START SLAVE;
```

Listagem 5.11: Iniciar replicação no Nó Gestão

8. A replicação síncrona tal como foi mencionado anteriormente terá de ser feita na camada aplicacional.

### 5.2.3 Camada Aplicacional (CA)

Neste ponto são apresentadas as implementações da Camada Aplicacional em *Open Source* e Microsoft. Ao contrário do ponto 5.2.2 Camada de Dados onde foi feita uma comparação quase que ponto por ponto entre as duas implementações, aqui será feita a apresentação de toda a CA em *Open Source* e depois toda a implementação da CA em Microsoft. Foi tomada esta decisão já que na CA a lógica de construção é totalmente diferente ao contrário do que acontece com a CD.

#### 5.2.3.1 Implementação com JEE

Para o desenvolvimento da CA em JEE com Java Beans foi utilizado o IDE Eclipse com a versão Oxygen e foi utilizado o servidor aplicacional Glassfish versão 5.

Como mencionado anteriormente, existem três servidores aplicacionais, um para a aplicação Encomendas, um para a aplicação Gestão e outro para aplicação Notificação. Tal como é possível observar na figura 5.14.

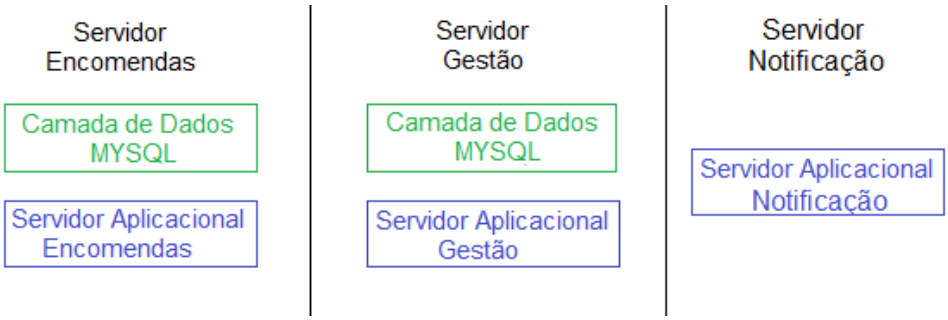


Figura 5.14: Servidores da solução

Na figura 5.15 está presente a organização da Camada Aplicacional. É possível observar que a CA foi dividida em duas sub-camadas, a Camada de Lógica de Negócio onde se encontram os *Enterprise Java Beans* e *Message-driven beans* e da Camada de Acesso a Dados onde se encontram as classes DAO (*Data Access object*).

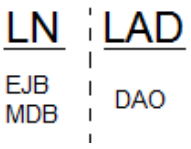


Figura 5.15: Separação da Camada Aplicacional

Para o desenvolvimento dos três servidores aplicacionais, foi seguido o seguinte padrão:

- **EAP_"Nome"** - Foi criado um projecto do tipo *Enterprise Application Project* que contém os recursos necessários para o desenvolvimento de aplicações empresariais. Este projecto por sua vez foi publicado no servidor aplicacional Glassfish.
- **EJB_"Nome"** - Neste projecto é onde se encontra toda a lógica aplicacional. Contém os EJB, MDB, as classes entidade do modelo de dados, classes DAO, classes Mapper.
- **EJB_"Nome"Cliente** - Neste projecto constam as interfaces remotas para a utilização dos EJB, bem como as classes DTO.

*** De referir que nos nomes dos projectos onde aparece "Nome" será "Encomendas", "Gestao" e "Notificacao" de acordo com o servidor onde se encontra o projecto.**

Nos servidores Encomendas e Gestão, onde os EJBs têm necessidade de interagir com a camada de dados, foi utilizado JPA como ORM para fazer o mapeamento das tabelas da CD para as classes da CA.

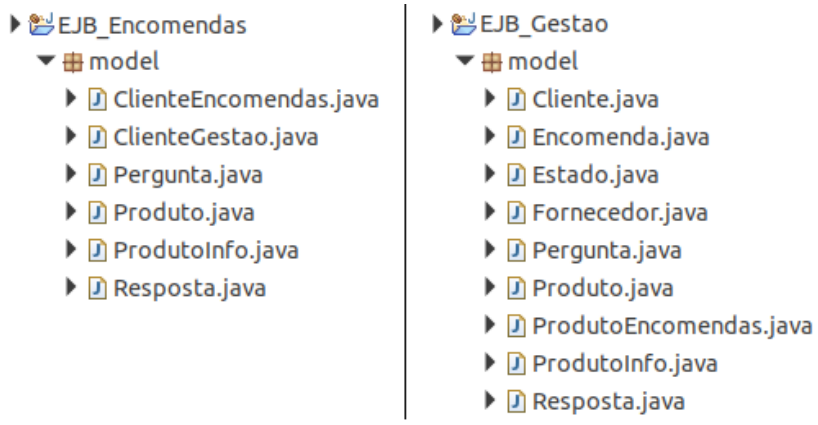


Figura 5.16: Classes que representam entidades

Na figura 5.16 estão as classes que representam as entidades utilizadas em cada um dos servidores aplicacionais. Tal como é possível verificar na imagem, no servidor aplicacional Encomendas foi feito o mapeamento da tabela cliente da CD

de Encomendas, mas também o mapeamento da tabela cliente da CD de Gestão. O mesmo acontece para a tabela produto no servidor aplicacional de Gestão. Isto acontece, porque é na camada aplicacional que será feita a replicação síncrona da inserção de um novo cliente e de um novo produto.

Com a utilização do JPA além de ser feito o mapeamento das tabelas em classes Java, é também posível tirar partido do *EntityManager* que permite criar, atualizar, excluir e fazer consultas sobre as entidade. Na listagem 5.12 é possível observar a estrutura do ficheiro "persistence.xml". Este ficheiro é um ficheiro de configuração padrão do JPA e tem de estar presente na pasta "META-INF"dentro do arquivo JAR que contém os beans de entidade.

```
1  <persistence-unit name="EncomendasPU" transaction-type="JTA
       ">
2  <provider>org.eclipse.persistence.jpa.PersistenceProvider</
       provider>
3  <jta-data-source>jdbc/encomendas</jta-data-source>
4    <class>model.ClienteEncomendas</class>
5    <class>model.(...)</class>
6  </persistence-unit>
7
8  <persistence-unit name="GestaoPU" transaction-type="JTA">
9  <provider>org.eclipse.persistence.jpa.PersistenceProvider</
       provider>
10 <jta-data-source>jdbc/gestao</jta-data-source>
11   <class>model.ClienteGestao</class>
12   <class>model.(...)</class>
13 </persistence-unit>
```

Listagem 5.12: Ficheiro persistence.XML

Como é possível observar, o ficheiro "persistence.xml"define configurações como:

- **persistence-unit:** Neste atributo é atribuído um nome único para posteriormente ser passado ao *EntityManager* para este conseguir interagir com a base de dados correcta.
- **provider:** Este atributo especifica a implementação subjacente do JPA *EntityManager*.

- **jta-data-source:** Neste atributo está o nome JINI da base de dados para o qual esta unidade de persistência é mapeada. Este nome foi definido no servidor Glassfish como irei apresentar de seguida.

- **class:** Este atributo contém as classes java que são mapeadas em tabelas da base de dados.

Para definir o nome JINI da base de dados foi necessário criar no servidor Glassfish as "*JDBC Connection Pools*" necessárias bem como as "*JDBC Resources*". Como é possível ver na figura 5.17 foram criadas duas *pools* de conexões, uma para a ligação à base de dados Encomendas e outra para a base de dados Gestão.

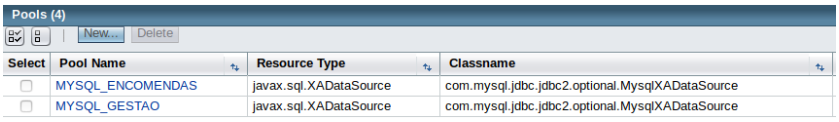


Figura 5.17: Connection Pools Glassfish

Para criação das "*JDBC Connection Pools*" para além de indicar o tipo de SGBD utilizado, no caso MYSQL foi necessário indicar o local onde se encontra o SGBD bem como as credênciais para o seu acesso, como é possível obeservar na figura 5.18.

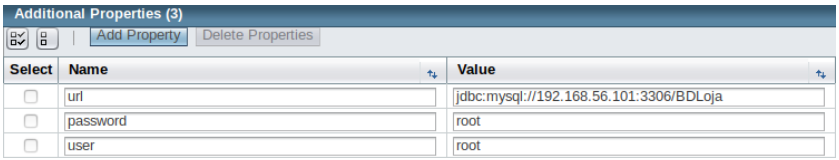


Figura 5.18: Configuração Connection Pools Glassfish

Na figura 5.19 é possível ver as duas ligações feitas às bases de dados Encomendas e Gestão com os respectivos nomes JINI.
Para cada uma destas ligações foi associada uma das *pools* criadas anteriormente.


Figura 5.19: JDBC Resource Glassfish

### Enterprise Java Beans

Irei agora apresentar os *Enterprise Java Beans* utilizados no desenvolvimento do projecto que é responsável pela camada de lógica de negócio. Foram utilizados *Java Beans Stateless* e *Message-driven beans*. Na figura 5.20 estão presentes todos os *Enterprise Java Beans* utilizados. Foram utilizados *Java Beans Stateless* para que o sistema fosse escalável e os *Message-driven beans* para as comunicações assíncronas através das filas de mensagens.

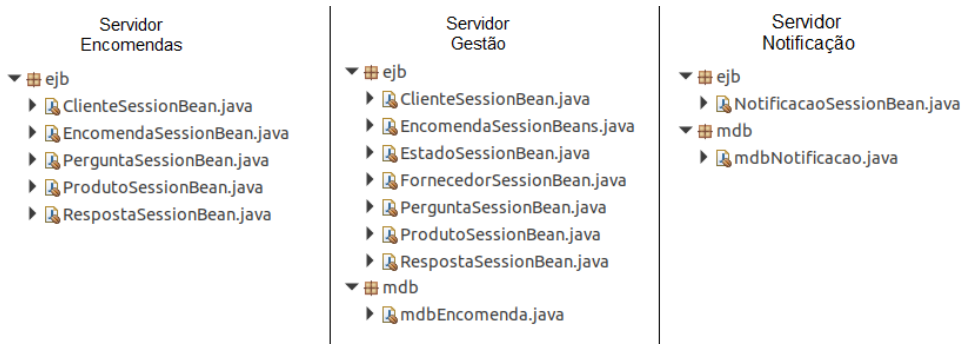


Figura 5.20: Enterprise Java Beans

Em praticamente todos os *Java Beans Stateless* foram criados métodos para salvar, obter um recurso por ID ou obter todos os recursos. Todos estes métodos acedem à CAD através das classes DAO.

Na listagem 5.13 está presente parte do código do EJB ClienteSessionBean do projecto do servidor Encomendas, ilustrando como foi desenvolvido o método *save*.

```
1  @Stateless
2  public class ClienteSessionBean implements
       ClienteSessionBeanRemote {
3    @PersistenceContext(unitName = "EncomendasPU")
4    private EntityManager emEncomendas;
5    @PersistenceContext(unitName = "GestaoPU")
6    private EntityManager emGestao;
7
8    private ClienteEncomendas cliEnc;
9    private ClienteGestao cliGest;
10   private ClienteMapper clienteMapper;
11
12   @Override
13   @TransactionAttribute(TransactionAttributeType.REQUIRED)
14   public ClienteDTO save(ClienteDTO c) throws Exception {
15     cliEnc = clienteMapper.ClienteDTOtoClienteEncomendas(c);
16
17     ClienteEncomendasDAO daoClienteEnc = new
         ClienteEncomendasDAO(emEncomendas);
18     cliEnc=daoClienteEnc.save(cliEnc);
19     c.setIdCliente(cliEnc.getIdCliente());
20
21     cliGest = clienteMapper.ClienteDTOtoClienteGestao(c);
22
23     ClienteGestaoDAO daoClienteGest = new ClienteGestaoDAO(
         emGestao);
24     daoClienteGest.save(cliGest);
25     return c;
26   }
27   @Override
28   public ClienteDTO findById(int id) { ... }
29 }
```

Listagem 5.13: EJB ClienteSessionBean

O padrão seguido por todos os EJB é semelhante ao da listagem 5.13, através de injecção de dependência com recurso à anotação "@PersistenceContext" é iniciado o *EntityManager* para manipulação das entidades da base de dados. Os métodos disponibilizados pelo EJB podem receber um objecto DTO com a informação que se pretende gravar na base de dados, podem receber um ID para executar uma acção sobre um recurso em especifico ou não receber nenhum parâmetro, como é o caso dos EJBs que têm o método "getAll()".

Sempre que se trata de um EJB que recebe um objecto DTO como parâmetro, é utilizado um *Mapper* para fazer o mapeamento para a classe entidade. Depois é utilizada uma classe DAO que é responsável pela interação com a base de dados, sendo passado à classe DAO o *EntityManager* que deverá ser utilizado para interagir com a BD.

Na listagem 5.13, são utilizados dois *EntityManager* e duas classes DAO num mesmo método transaccional já que se trata de uma inserção em dois servidores diferentes de forma síncrona.
De realçar o facto do método *save* estar marcado com a anotação "@TransactionAttribute(TransactionAttributeType.REQUIRED)" ainda que este seja o valor por omissão. Desta forma é garantido que quando o método *save* é invocado no contexto de uma transacção, será utilizada a transacção do cliente, caso contrário será criada uma nova transacção.
A gestão da transacção é feita pelo contentor de EJB, sendo que se uma das classes DAO falhar, o contentor de EJB irá dar ordem de *rolled back*, se ambas as classes DAO persistirem a informação sem problemas, será dada ordem de *commit*.
Isto acontece porque o contentor de EJB verifica se todos os participantes da transacção votarão no *commit*, ou seja, se terminaram o seu trabalho sem que ocorresse uma excepção. Caso ocorra uma excepção é dada ordem de *rolled back*, caso contrário é feito *commit*.

Na listagem 5.14 é apresentado o exemplo da implementação do método "save" da classe "ClienteEncomendasDAO".

```
1  public class ClienteEncomendasDAO {
2    public ClienteEncomendas save(ClienteEncomendas t) throws
         Exception {
3      if(t.getIdCliente() == 0) {
4        entityManager.persist(t);
5        entityManager.flush();
6      }else {
7        if(!entityManager.contains(t)) {
8          if(entityManager.find(ClienteEncomendas.class, t.
               getIdCliente())==null)
9            throw new Exception("Erro ao actualizar o
                 ClienteEncomendas");
10       }
11       t= entityManager.merge(t);
12     }
13     return t;
14   }
15 }
```

Listagem 5.14: Método save de ClienteDAO

Como é possível observar na listagem 5.14, com a utilização do *EntityManager* o método ficou bastante simples, fazendo apenas a verificação se o ID é igual a zero, nesse caso será uma nova inserção utilizando para esse efeito o método *persist* do *EntityManager*. Caso contrário, será uma actualização e para isso é obtido o objecto a actualizar através do método *find* e por fim é feita actualização com os novos valores através do método *merge*.

No que diz respeito aos *Message-driven beans*, é possível observar na figura 5.20 que foram definidos dois *Message-driven beans* o MDBEncomendas no servidor Gestão e o MDBNotificacao no servidor de Notificação. Houve a necessidade de definir estes dois MDB para possibilitar a comunicação assíncrona entre os Servidores.

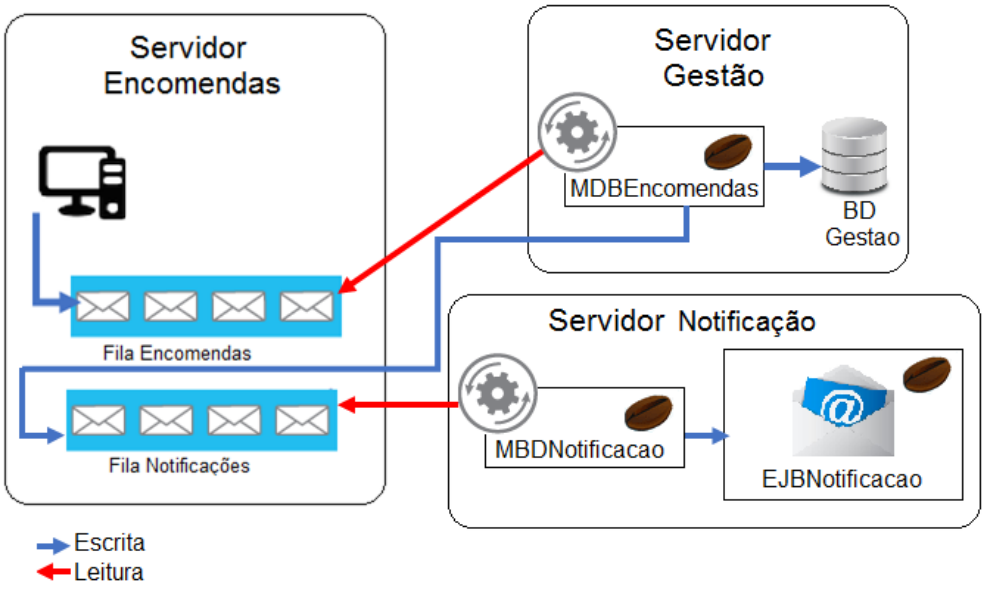


Figura 5.21: Comunicação assíncrona

Na figura 5.21 é possível observar a forma como a comunicação é feita com recurso às filas de mensagens. Idealmente as filas de mensagens deveriam estar num servidor à parte para garantir que em caso de problemas em qualquer um dos três servidores do sistema, os outros dois continuariam a funcionar e com acesso às filas de mensagens. Neste projecto, e tendo em conta as limitações de hardware optei por colocar as filas de mensagens no servidor Encomendas.

No servidor Encomendas, quando um cliente faz uma encomenda a mensagem é enviada para a fila Encomendas. Por sua vez o MDBEncomendas definido no servidor Gestão tem o método "OnMessage" à escuta na fila de mensagens de Encomendas, assim sendo, sempre que é inserida uma mensagem na fila de Encomendas, se o servidor de Gestão estiver operacional a mensagem será consumida. Após o consumo da mensagem é registada a encomenda na base de dados de Gestão e é colocada uma mensagem na fila de Notificações.

No caso do MDBNotificacao, sempre que existe uma alteração de estado de uma encomenda, é colocada uma mensagem na fila de Notificações, o servidor Notificação tem o método "OnMessage" à escuta na fila de mensagens de Notificação, assim sendo, sempre que é inserida uma mensagem na fila de Notificações, se o servidor de Notificação estiver operacional a mensagem será consumida. Depois do consumo da mensagem, é enviado um email para o cliente a informar o estado do pedido.

De seguida irei apresentar um exemplo de um *Message-driven beans*. Irei mostrar como inserir mensagens na fila e como as consumir.
Como foi referido anteriormente, as filas ficarão alojadas no servidor de Encomendas, ou seja, as filas têm de ser acedidas remotamente. No acesso às filas de menssagens despendi algum tempo, já que na documentação do Glassfish e JEE não encontrei de forma clara a forma de fazer esta configuração.
A solução que encontrei para a configuração dos três servidores para que conseguissem aceder às filas de forma remota, passam por colocar a seguinte configuração na consola de administração do Glassfish:

- **Servidor Encomendas**: Na figura 5.22 mostro os menus que tive que aceder para indicar que o tipo do serviço é *EMBEDDED*, e que a porta que estará à escuta neste servidor é a 7676.


Figura 5.22: Configuração JMS - Servidor Encomendas

- **Servidor Gestão e Notificação**: Na figura 5.23 mostro os menus que tive que aceder em ambos os servidores para indicar que o tipo do serviço é *REMOTE*, e que o host é o IP do servidor Encomendas e a porta 7676.

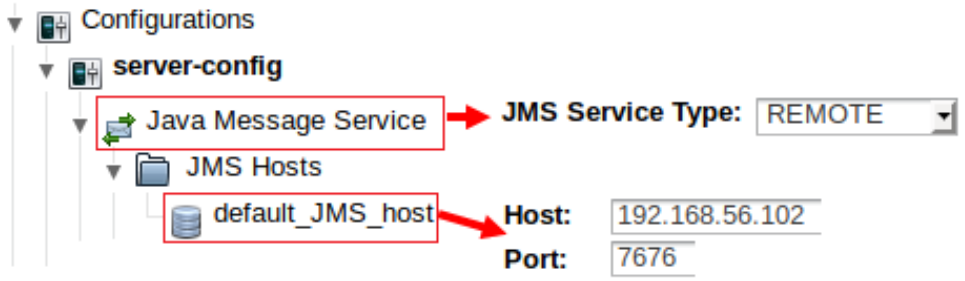


Figura 5.23: Configuração JMS - Servidor Gestão e Notificação

- **Servidor Encomendas, Gestão e Notificação**: Na figura 5.24 mostro os menus a aceder para configurar *Destination Resource* e *Connection Factories*. Na

figura apenas aparece a fila de Encomendas, mas tive de fazer exactamente o mesmo para a fila de Notificação.


Figura 5.24: Configuração JMS - Destination Resource e Connection Factories

Na listagem 5.15 apresento o exemplo do código necessário para colocar uma mensagem na fila de Encomendas.
Como é possível observar, a configuração da fila e da conexão são feitas através de injecção de depêndencia com recurso à anotação "@Resource".
Quando o método "saveEncomenda"é chamado, é criada uma conexão e uma sessão para essa mesma conexão. Depois é criado um produtor e uma mensagem para aquela sessão. Depois de fazer o mapeamento do objecto DTO na mensagem a enviar para a fila, o produtor faz o envio da mensagem para a fila.

```
@Stateless
public class EncomendaSessionBean implements
    EncomendaSessionBeanRemote {
  @Resource(mappedName="newEncomendaQueue")
  private Queue dest;
  @Resource(mappedName="newEncomendaConnectionFactory")
  private ConnectionFactory queue;

  @Override
  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  public void realizarEncomenda(EncomendaDTO encomendaDTO)
  { saveEncomenda(encomendaDTO); }

  public void saveEncomenda(EncomendaDTO encomendaDTO){
    Connection connect = null;
    try {
      connect = queue.createConnection();
      Session session = connect.createSession(true, 0);
      MessageProducer prod = session.createProducer(dest);
      MapMessage msg = session.createMapMessage();
      msg.setInt("idCliente", encomendaDTO.getIdCliente());
      msg.setInt("idProduto", encomendaDTO.getIdProduto());
      msg.setInt("quantidade",encomendaDTO.getQuantidade());
      prod.send(msg);
    } catch (JMSException je) {
      System.out.println("ERRO : saveEncomenda");
    } finally {
      if (connect != null) {
        try { connect.close(); }
        catch (Exception e) { }
      }
    }
  }
}
```

Listagem 5.15: MDB Envio de mensagem para a fila

Na listagem 5.16 apresento o exemplo do código necessário para ficar à escuta de mensagens na fila de Notificações.
Como é possível observar na anotação "@MessageDriven" são indicados parâmetros de configuração para a *Destination Resource* e *Connection Factory* que foram criados no Glassfish. Ao se tratar de um *Message-driven beans* a classe implementa a interface "MessageListener" que tem o método "onMessage". É o método "onMessage" que está à escuta na fila de Notificação. Ao entrar uma mensagem na fila, o método "onMessage" é iniciado.

```
1  @MessageDriven(
2  activationConfig = {
3  @ActivationConfigProperty(propertyName = "destination",
       propertyValue = "newNotificacaoQueue"),
       @ActivationConfigProperty(propertyName = "
       destinationType", propertyValue = "javax.jms.Queue")},
4  mappedName = "newNotificacaoQueue")
5  public class mdbNotificacao implements MessageListener {
6    @EJB
7    private NotificacaoSessionBean notificacaoBean;
8
9    @TransactionAttribute(TransactionAttributeType.REQUIRED)
10   public void onMessage(Message message) {
11   try {
12     MapMessage notificacao = (MapMessage) message;
13     idEncomenda=notificacao.getInt("idEncomenda");
14     idCliente= notificacao.getInt("idCliente");
15     ( ... )
16     estado = notificacao.getString("estado");
17
18     notificacaoBean.enviarEmail(new EncomendaDTO(idEncomenda
         , idCliente, emailCliente, moradaCliente, idProduto,
         quantidade, estado));
19   } catch (Exception e) { e.printStackTrace(); }
20   }
21 }
```

Listagem 5.16: MDB Recepção de mensagem da fila

Para testar a aplicação foram criadas duas aplicações consola no Servidor Encomendas e no Servidor Gestão. No Servidor Notificação não foi necessário criar qualquer aplicação de teste já que o consumo das mensagens e envio do email é feito de forma automática.
Como é possível observar na figura 5.25 os menus de cada uma das aplicações têm as operações requeridas no enunciado do problema.

Aplicação Encomendas

```
****************************************
****************** MENU ****************
****************************************
# Opções #
   1 - Registar cliente;
   2 - Consultar dados cliente;
   3 - Apagar registo cliente;
   4 - Listar todos os produtos;
   5 - Procurar produto especifico;
   6 - Realizar encomenda
   7 - Fazer uma pergunta;
   8 - Consultar uma pergunta;
   9 - Listar todas as perguntas;
   10 - Dar uma resposta;
   11 - Consultar uma resposta;
   12 - Consultar todas as respostas;
```

Aplicação Gestão

```
****************************************************
****************** MENU ********************************
****************************************************
# Opções #
   1 - Listar todos os clientes;
   2 - Listar todos os estados;
   3 - Listar todos os produtos;
   4 - Listar todos os fornecedores;
   5 - Listar todas as encomendas;
   6 - Listar informação de um cliente em especifico;
   7 - Listar informação de um produto em especifico;
   8 - Listar informação de um fornecedor em especifico;
   9 - Listar informação de uma encomenda em especifico;
   10 - Inserir fornecedor;
   11 - Inserir produto;
   12 - Encomendar produto;
   13 - Alterar estado de encomenda;
   14 - Receber encomenda (Actualiza stock);
   15 - Listar todas as perguntas;
   16 - Listar todas as respostas;
   17 - Consultar pergunta por ID;
   18 - Consultar resposta por ID;
   19 - Adicionar uma resposta;
   0 - Para terminar;
```

Figura 5.25: Menu aplicação Encomendas e Gestão

Dentro do método "Main"de cada uma das aplicações de teste, foi necessário recorrer ao JINI para ter acesso aos EJBs. Na listagem 5.17 é possível observar o troço de código necessário para aceder à interface remota dos EJB de produto e cliente.

```
1    Context ctx = new InitialContext();
2 Object obj = ctx.lookup("java:global/EAP_AppGestao/EJB_
    Gestao/ClienteSessionBean!ejb.ClienteSessionBeanRemote")
    ;
3 ClienteSessionBeanRemote t = (ClienteSessionBeanRemote)
    PortableRemoteObject.narrow(obj,
    ClienteSessionBeanRemote.class);
4
5 obj = ctx.lookup("java:global/EAP_AppGestao/EJB_Gestao/
    ProdutoSessionBean!ejb.ProdutoSessionBeanRemote");
6 ProdutoSessionBeanRemote produto = (
    ProdutoSessionBeanRemote) PortableRemoteObject.narrow(
    obj, ProdutoSessionBeanRemote.class);
```

Listagem 5.17: Obter EJB com JINI

#### 5.2.3.2 Implementação com WCF

Para o desenvolvimento da CA em WCF foi utilizado o Microsoft Visual Studio 2017.
Como mencionado anteriormente, e à semalhança do que foi feito na CA com JEE, existem três servidores aplicacionais, um para a aplicação Encomendas, um para a aplicação Gestão e outro para aplicação Notificação.

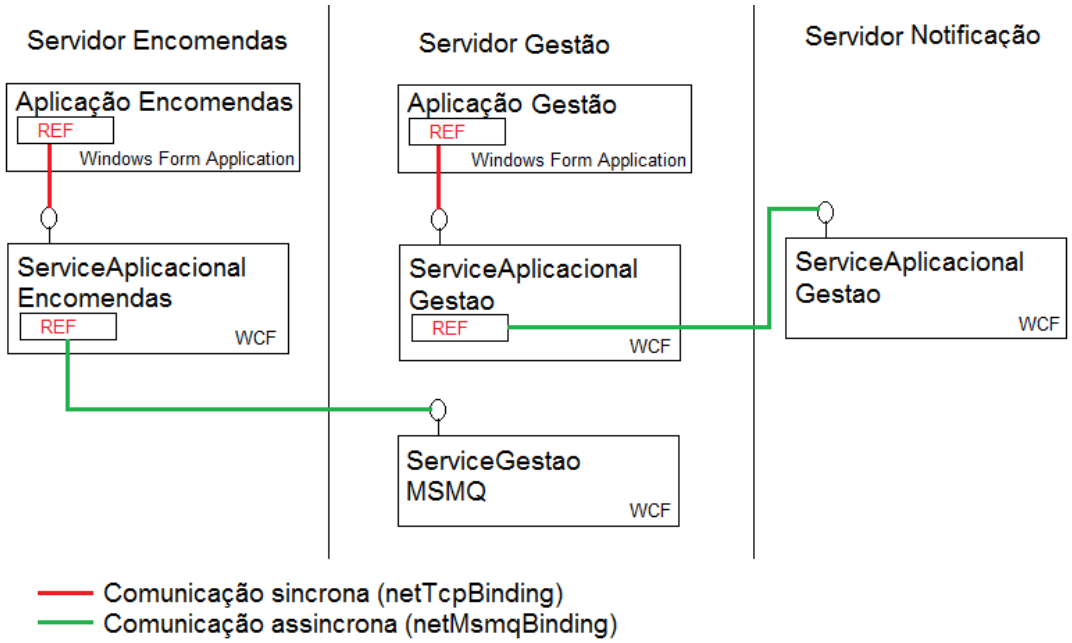


Figura 5.26: Servidores da solução

Tal como é possível observar na figura 5.26, a implementação do projecto com tecnologias Microsoft passou pela criação de serviços WCF onde ficou a lógica de negócio. Como foi explicado anteriormente, os serviços WCF implementam uma interface com um conjunto de operações que são disponiblizadas para os clientes que utilizem o serviço. Na figura 5.27 estão as interfaces e operações que foram definidas para a implementação do projecto.

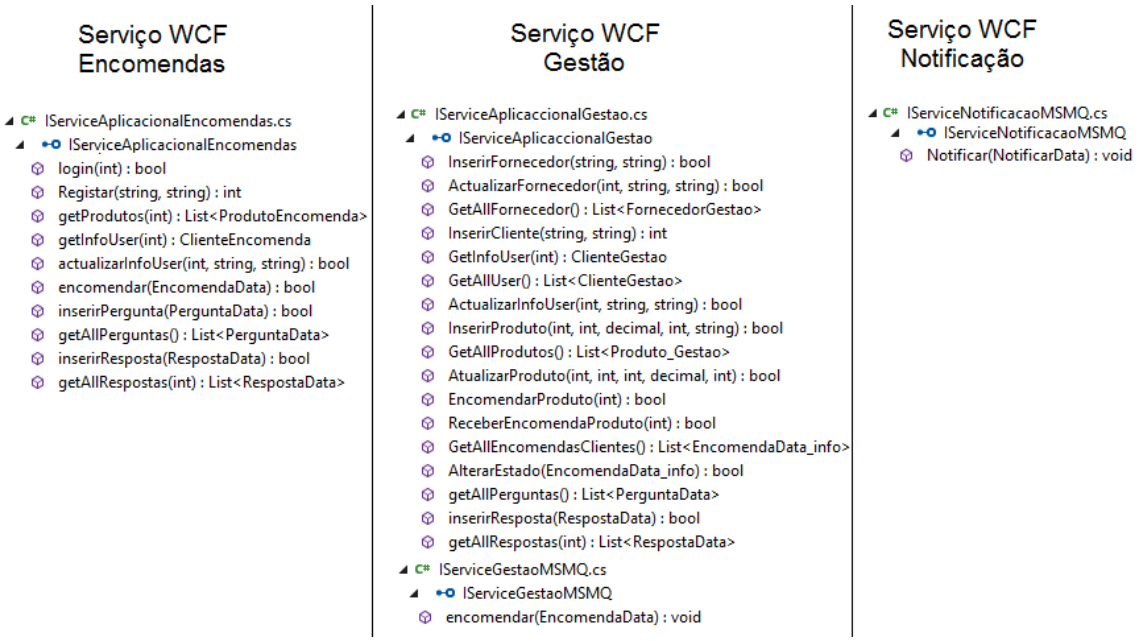


Figura 5.27: Interfaces e operações dos Serviços WCF

**Servidor Encomendas:**
No servidor Encomendas foi implementado um serviço WCF designado de "ServiceAplicacionalEncomendas" que define uma série de operações presentes no contrato "IServiceAplicacionalEncomendas"como ilustra a figura 5.27. No contrato também defini um conjunto de *DataContracts* para serem usados na passagem de informação entre o cliente e o servidor aplicacional "Encomendas".

O endpoint do Serviço Aplicacional Encomendas é composto por:

- **Address:** "net.tcp://192.168.56.105:8733/ServidorAplicacional_Encomendas"
- **Binding:** "netTcpBinding"
- **Contract:** "IServiceAplicacionalEncomendas"

De referir que todas as operações à excepção da operação "encomendar" são operações síncronas entre o cliente que utiliza a aplicação e o serviço WCF.
No caso particular da operação "encomendas", esta tem como objectivo garantir

que o pedido efectuado pelo cliente é inserido na fila de mensagens que será consumida pelo servidor aplicacional "Gestão". Para isso, é criada uma transacção com o nível de isolamento ReadCommited (por default) e só depois enviada a mensagem para a fila conforme ilustra a listagem 5.18.

```
1  [OperationBehavior(TransactionScopeRequired = true,
2          TransactionAutoComplete = true)]
3  public bool encomendar(EncomendaData encData)
4  {
5    PRX_Gestao.ServiceGestaoMSMQClient prx = new PRX_Gestao.
       ServiceGestaoMSMQClient();
6    PRX_Gestao.EncomendaData enc = new PRX_Gestao.
       EncomendaData();
7    enc.idCliente = encData.idCliente;
8    enc.idProduto = encData.idProduto;
9    enc.quantidade = encData.quantidade;
10   prx.encomendar(enc);
11   prx.Close();
12   return true;
13 }
```

Listagem 5.18: Operação encomendar produto

**Servidor Gestão:**
No servidor Gestão foram implementados dois serviços WCF designados de "ServiceAplicacionalGestao" e "ServiceGestaoMSMQ" que definem as operações presentes no contrato "IServiceAplicacionalGestao" e "IServiceGestaoMSMQ" respectivamente, como ilustra a figura 5.27. No contrato também defini um conjunto de *DataContracts* para serem usados na passagem de informação entre o cliente e o servidor aplicacional "Gestao".

Os endpoints dos Serviços Aplicacionais de Gestão são composto por:

- **Address:** "net.tcp://192.168.56.104:8735/ServidorAplicacional_Gestao"
- **Binding:** "netTcpBinding"
- **Contract:** "IServiceAplicacionalGestao"

- **Address:** “net.msmq://192.168.56.104:8735/private/filaencomenda”
- **Binding:** “netMsmqBinding”
- **Contract:** “IServiceGestaoMSMQ”

No serviço WCF "ServiceAplicacionalGestao", toda a comunicação entre a aplicação de Gestão e o serviço WCF é feita de forma síncrona.
No caso particular da operação “AlterarEstado”, esta tem como objectivo garantir que é efectuada a inserção de uma mensagem na fila de notificações que será consumida pelo servidor aplicacional “Notificação”, e que a informação é actualizada no SGBD. Para isso, é criada uma transacção com o nível de isolamento ReadCommited (por padrão) e é feita a tentativa de inserção e envio da mensagem.

O serviço WCF "ServiceGestaoMSMQ", disponibiliza uma única operação "encomendar" que tem como único objetivo proceder à encomenda de um determinado produto para um determinado cliente e alterar o estado da mesma. Esta operação é feita recorrendo a uma transacção para garantir que a inserção na base de dados da encomenda retirada da mensagem da fila de encomendas e a colocação da notificação na fila de notificações é feita de forma atómica, ou seja, qualquer uma destas operações só fica concluída se todas tiverem sucesso.

Se tudo correr bem a aplicação irá colocar uma mensagem na fila de mensagens de notificações de modo a notificar o cliente da alteração de estado da encomenda.
Na figura 5.28 seguinte podemos observar a fila de encomendas já criada no sistema.

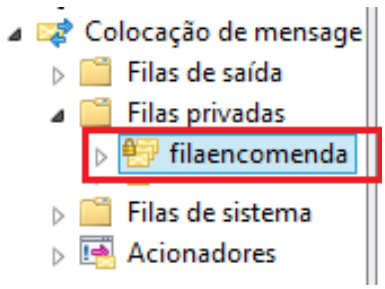


Figura 5.28: Fila de encomendas

Na figura 5.29 seguinte podemos observar o diagrama de sequência da operação encomendar um produto por parte de um cliente.

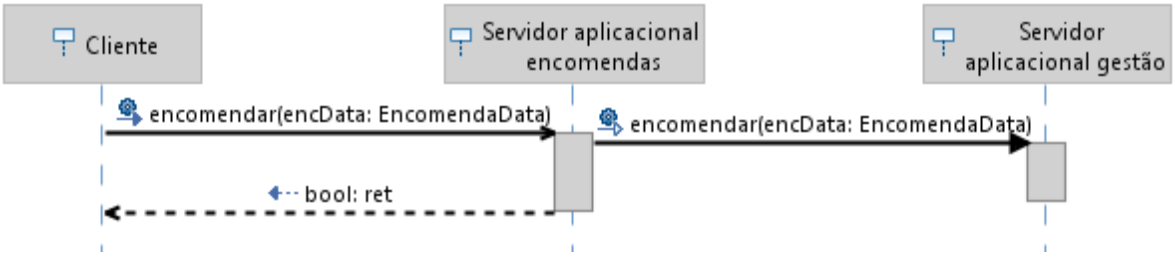


Figura 5.29: Diagrama de sequência operação encomendar

**Servidor Notificação:**
O servidor aplicacional de notificações foi implementado para notificar o cliente das alterações de estado das encomendas.

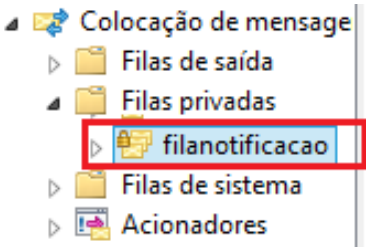


Figura 5.30: Fila de notificação

Este serviço é assíncrono, e consome as mensagens da fila de notificações colocadas pelo serviço aplicacional gestão.

O endpoint do Serviço WCF de Notificação é composto por:

- **Address:** "net.msmq://192.168.56.106/private/filanotificacao"
- **Binding:** "netMsmqBinding"
- **Contract:** "IServiceNotificacaoMSMQ"

O servidor aplicacional notificação, ao consumir uma mensagem da fila envia um email para o cliente que realizou a encomenda com a indicação do estado em que se encontra a encomenda.

**Construção dos serviços WCF:**
Tendo em conta que para o desenvolvimento dos serviços WCF anteriormente mencionados segui sempre o mesmo padrão, irei apresentar de seguida os passos para o desenvolvimento do serviço WCF "ServiceGestaoMSMQ". Optei por apresentar o serviço "ServiceGestaoMSMQ" e não outro, porque é um serviço WCF pequeno, no que respeita ao número de linhas de código, por utilizar filas de mensagens e persistência dos dados, ou seja, é um serviço WCF completo.

**Construção do contrato "IServiceGestaoMSMQ":**

```
1  public interface IServiceGestaoMSMQ
2  {
3    [OperationContract(IsOneWay =true)]
4    void encomendar(EncomendaData encData);
5  }
6  [DataContract]
7  public class EncomendaData
8  {
9    [DataMember(Order = 1, IsRequired =true)]
10   public int idCliente { get; set; }
11   [DataMember(Order = 2, IsRequired = true)]
12   public int idProduto { get; set; }
13   [DataMember(Order = 3, IsRequired = true)]
14   public int quantidade { get; set; }
15 }
```

Listagem 5.19: Contrato IServiceGestaoMSMQ

Tal como é possível observar na listagem 5.19, foi criada uma interface designada de "IServiceGestaoMSMQ" e marcada com a propriedade "[ServiceContract(Session Mode = SessionMode.Required)]" para indicar que este contrato apenas aceita clientes com sessão. Clientes sem sessão não podem aceder a este serviço WCF.
Também é possível observar que o único método desta interface está marcada com a propriedade "[OperationContract(IsOneWay =true)]", desta forma é definido no contrato, que esta operação está disponível para os cliente do serviço WCF e que quando o cliente fizer uma chamada a esta operação não irá ficar bloqueado à espera de retorno.
Por fim, é também definida uma nova classe e marcada com a propriedade "[DataContract]" para colocar no contrato a estrutura de dados a utilizar para interagir com o serviço WCF.

**Implementação do Serviço "ServiceGestaoMSMQ":**

```
1  [ServiceBehavior(InstanceContextMode = InstanceContextMode.
       PerSession,
2  TransactionAutoCompleteOnSessionClose = true)]
3  public class ServiceGestaoMSMQ : IServiceGestaoMSMQ
4  {
5    [OperationBehavior(TransactionScopeRequired =true)]
6    public void encomendar(EncomendaData encData)
7    {
8      //Persistir encomenda na BD de Gesto
9      //Enviar mensagem para fila de Notificao
10   }
11 }
```

Listagem 5.20: Implementação do Serviço ServiceGestaoMSMQ

Tal como é possível observar na listagem 5.20, a implementação do serviço WCF "ServiceGestaoMSMQ", passa pela implementação da interface "IServiceGestaoMSMQ". A classe foi marcada com as propriedades "[ServiceBehavior( InstanceContextMode = InstanceContextMode.PerSession, TransactionAutoCompleteOnSessionClose = true)]" para indicar o tipo de instancia, e indicar que as transacções são completas de forma automática no final da sessão, não sendo necessário estar a finalizar as transacções de forma explicita.
O método implementado, é marcado com a propriedade "[OperationBehavior(

TransactionScopeRequired =true)]" para indicar que é obrigatório o uso de transacção. Se o cliente propagar a transacção para o serviço WCF, será utilizada a transacção do cliente, caso contrário é utilizada uma nova transacção.

Atenção: Para a utilização do MSMQ é necessário activar nas funcionalidades do Windows as filas de mensagens, tal como ilustra a figura 5.31.

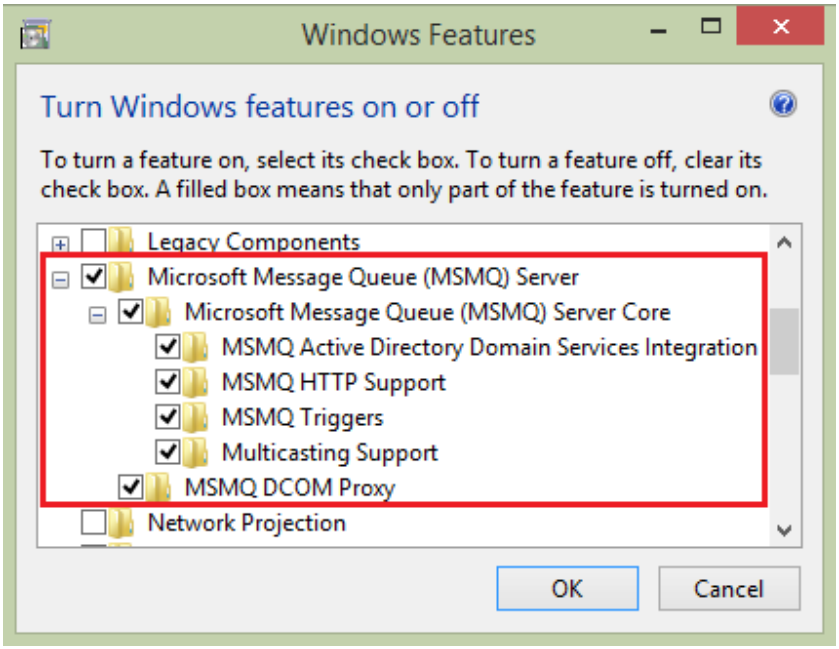


Figura 5.31: Activar MSMQ

**Configuração do ABC do serviço:** Para a configuração do ABC (*Address, Binding e Contract* ) foi necessário colocar a ficheiro "App.config" com a configuração abaixo indicada.

```
1 <endpoint address="net.msmq://192.168.56.104/private/
    filaencomenda"
2    binding="netMsmqBinding" bindingConfiguration="
       GestaoBinding"
3    contract="ServidorAplicacional_Gestao.IServiceGestaoMSMQ
       " />
```

Listagem 5.21: Configuração do ABC

**Entity Framework:**
Para a persistência dos dados, foi utilizado o Entity Framework como ferramenta ORM. No Visual Studio foi utilizada a metodologia *Database First*, obtendo o modelo entidade associação da base de dados a mapear para objectos, conforme ilustra a figura 5.32.

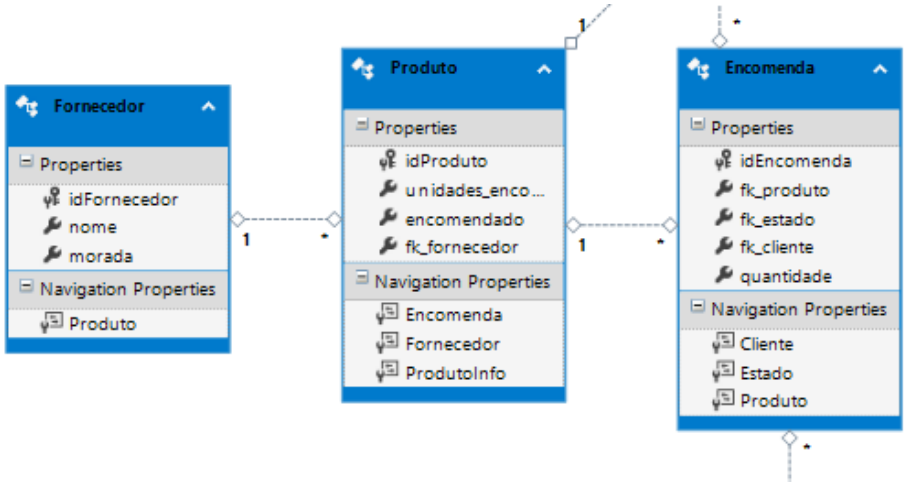


Figura 5.32: Entity Framework vista grafica

Ao carregar em cima de cada um dos atributos das tabelas da figura 5.32 também é possível definir várias propriedades, estas propriedades também podem ser alteradas programáticamente, mas sem dúvida que com a vista gráfica o processo de parameterização é mais simples.

| | |
|---|---|
| **Code Generation** | |
| Getter | Public |
| Setter | Public |
| **General** | |
| Concurrency Mode | None |
| Default Value | (None) |
| Documentation | |
| Entity Key | **True** |
| Name | **idEncomenda** |
| Nullable | **False** |
| StoreGeneratedPattern | **Identity** |
| Type | **Int32** |

Figura 5.33: Entity Framework propriedades

Na listagem 5.22 é possível ver a utilização do Entity Framework para actualizar os dados relativos a um fornecedor.

Numa primeira fase vou buscar a informação relativa ao fornecedor que tem o ID igual ao idFornecedor. Depois altero o estado do contexto para modificado, e altero os campos de nome e morada. Por fim basta indicar ao contexto que salve as alterações.

```
1  public bool ActualizarFornecedor(int idFornecedor, string
       nome, string morada)
2  {
3  bool ret = true;
4  using (var ctx = new GestaoEntities())
5  {
6    var fornecedor = (from f in ctx.Fornecedor
7    where f.idFornecedor == idFornecedor
8    select f).FirstOrDefault();
9
10   ctx.Entry(fornecedor).State = System.Data.Entity.
       EntityState.Modified;
11   fornecedor.morada = morada;
12   fornecedor.nome = nome;
13   try
14   {
15     ctx.SaveChanges();
16   }
17   catch (Exception)
18   {
19     ret = false;
20   }
21 }
22 }
23 return ret;
```

Listagem 5.22: Exemplo de uso do EF

Para testar a aplicação foram criadas duas aplicações *Windows Form* uma no Servidor de Encomendas e outra Servidor Gestão. No Servidor Notificação não foi necessário criar qualquer aplicação de teste já que o consumo das mensagens e envio do email é feito de forma automática pelo serviço WCF. Como é possível observar na figura 5.34 os menus de cada uma das aplicações são sugestivos e têm todas as operações para cumprir os requisitos do sistema.

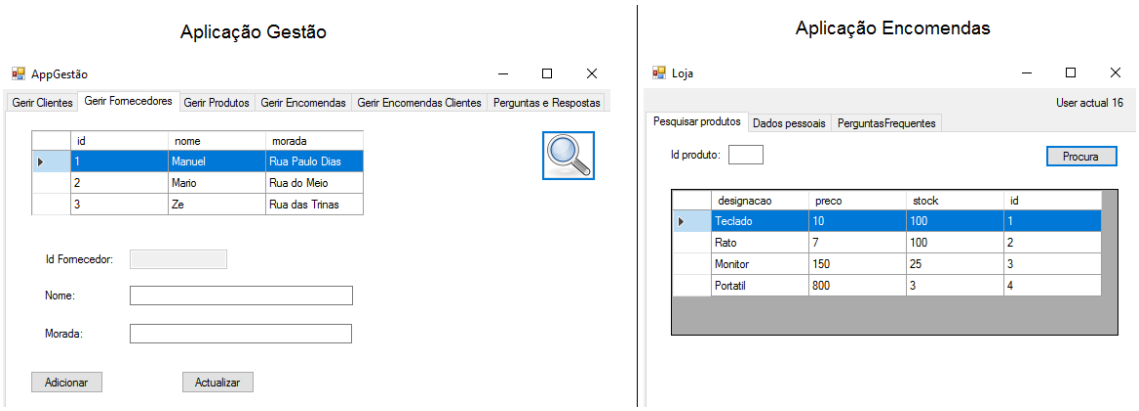


Figura 5.34: Aplicação Gestão e Encomendas

Dentro de cada um dos projectos das aplicações ilustradas na figura 5.34, foram adicionadas as referências para os serviços WCF desenvolvidos. Depois de obter a referência para os serviços WCF, foi apenas necessário utilizar o *proxy* para chamar cada uma das operações.

# 6

# Resultados

No que diz respeito aos resultados obtidos e comparando o desenvolvimento do prótotipo tanto em Microsoft com WCF e SQL Server como em Open Source com JEE e MYSQL, posso afirmar que para os requisitos do protótipo desenvolvido qualquer uma das opções é viável, ou seja, foi possível desenvolver os dois protótipos com sucesso.

No entanto, existem diferenças entre as tecnologias utilizadas, que devem ser tidas em conta em futuros projectos já que a escolha incorrecta da tecnologia pode levar à impossibilidade da realização de alguns possíveis requisitos de um projecto.

## 6.1 Camada de Dados

No que diz respeito à CD, foram utilizadas as tecnologias SQL Server e MYSQL. Embora o projecto tenha sido realizado com sucesso com ambas as tecnologias, existem grandes diferenças entre as tecnologias.

- **Construção de base dados relacional:** Não existe diferenças significativas entre as duas tecnologias, apenas ligeiras diferenças na sintaxe.
- **Construção de procedimentos armazenados, gatilhos ou vistas:** Não existe diferenças significativas entre as duas tecnologias, apenas ligeiras diferenças na sintaxe.
- **Transacções distribuídas:** Neste ponto existem grandes diferenças entre as tecnologias, O SQL Server permite iniciar e participar numa transacção distribuída. O MYSQL apenas permite participar numa transacção distribuída como um gestor de recursos.
- **Replicações:** Neste ponto também existem grandes diferênças tanto nas replicações síncronas como nas assíncronas.
  - **Síncronas:** No SQL Server é possível replicar sincronamente com recurso às transacções distribuídas. No MYSQL são necessárias soluções mais robustas como é o caso dos clusters.
  - **Assíncronas:** Tanto o SQL Server como o MYSQL possibilitam este tipo de replicação, no entanto o SQL Server tem um Wizard muito intuitivo que ajuda a fazer toda a confguração, ao passo que o MYSQL obriga a vários passos mais morosos e complexos para a configuração da replicação.
- **Custo monetário:** Embora este estudo não incida sobre a comparação da poupança monetária que pode ser feita, o MYSQL é um software gratuito, ao passo que o SQL Server tem um custo associado.

## 6.2 Camada Aplicacional

No que diz respeito à CA, foram utilizadas as tecnologias WCF e JEE com EJB . Embora o projecto tenha sido realizado com sucesso com ambas as tecnologias, existem algumas diferenças entre as tecnologias.

- **Controlo transaccional:** No que diz respeito ao controlo transaccional, tanto o JEE com EJB ou o WCF oferecem a possibilidade da utilização de tansacções ACID.

Ambas as tecnologias oferecem mecanismos para que o programador se concentre simplesmente na lógica de negócio, deixando a gestão do controlo transaccional ser tratado pela tecnologia. Quando isto acontece o programador tem apenas que utilizar anotações no caso do JEE ou atributos no caso do WCF para fazer o controlo transaccional de forma declarativa.
Se o programador pretender também poderá fazer o controlo transaccional programaticamente em ambas as tecnologias.

Ainda em relação às transacções, de realçar que ambas as tecnologias permitem a propagação de transacções.

- **Controlo de concorrência:** No que diz respeita ao controlo de concorrência, ambas as tecnologias oferecem opções para que o programador se possa focar na logica da aplicação e deixar que a tecnologia faça o controlo transaccional.
  Recorrendo a atributos em WCF ou anotações em JEE o programador pode indicar que determinada operação é *thread-safe* por aceder a recursos partilhados e pode indicar que outras operações que são apenas de leitura podem ser acedidas de forma concorrente.

- **Disponibilidade e escalabilidade:** No que diz respeito à disponibilidade e escalabilidade, ambas as tecnologias têm soluções para suportar essas caracteristicas.
  No WCF são utilizados serviços com tipo de instanciação *per-call*, no caso do JEE com EJB são utilizados os EJB *Stateless*. Em ambas as soluções temos a garantia que não existe partilha de estado, podendo desta forma acrescentar outros serviços *per-call* ou EJB *Stateless* iguais para aumentar a disponibilidade ou escalabilidade do sistema.

- **Comunicação assíncrona:** Ambos os sistemas possibilitam comunicação assíncrona com recurso a filas de mensagens.
  No WCF é necessário ter atenção à escolha do binding e à construção do ficheiro de configuração, mas não é necessário nenhum código especifico para a utilização das filas de mensagens (MSMQ).
  No JEE utilizando os *Message-driven bean* é necessário a utilização de código e anotações especificas para a utilização das filas de mensagens.

- **Ferramentas ORM:** No que diz respeito às ferramentas estudadas, JPA para JEE e Entity Framework para o WCF posso salientar que ambas as ferramentas possibilitam o *"reverse engineering"* que facilita bastante o trabalho

do programador.
No entanto, na minha opinião o Entity Framework tem uma utilização mais amigável, já que possibilita a visualização do modelo da BD e a alteração das propriedades dos atributos das tabelas num ambiente grafico. Já o JPA, não encontrei nenhuma opção grafica para a configuração dos atributos.

# 7

# Conclusões

Com a elaboração deste trabalho foi possível aprofundar e desenvolver competências que tinham sido adquiridas em algumas das unidades curriculares deste curso de Mestrado. No entanto, este projecto obrigou-me a desenvolver novas competências, como foi o caso da escrita do relatório e a possibilidade de trabalhar com novas tecnologias como foi o caso do JEE e MYSQL que serviram de comparação com as tecnologias da Microsoft WCF e SQL Server que foram estudas em algumas unidade curriculares do curso.

De realçar o facto do meu trabalho ser bastante extenso já que envolveu um estudo teórico dos aspectos a ter em conta no desenvolvimento de sistemas de informação empresariais, estudo das tecnologias MYSQL, SQL Server, JEE com EJB e WFC e o desenvolvimento de dois sistemas de informação empresariais com recurso a diferentes tecnologias.
Tendo em conta a extensão do trabalho, aprendi que se deve definir claramente o objectivo final do trabalho e cumprir sempre as metas estabelecidas ao longo do projecto, sob pena de comprometer a entrega do mesmo.

O presente trabalho mostra que a maior parte dos aspectos a ter em conta no desenvolvimento SIE são possíveis de resolver com sucesso tanto em JEE com EJB e MYSQL como com WCF e SQL Server, no entanto, pode haver requisitos do sistema que apenas possam ser cumpridos por uma das tecnologias. (Por exemplo,

se houver um requisito que obrigue a replicação sincrona na camada de dados, não será possível optar pelo MYSQL). Por esse motivo devem ser analisados os requisitos do SIE a desenvolver para saber qual a melhor tecnlogia a utilizar.

Neste trabalho além dos exemplos apresentados nos capítluos 3 e 4, que mostravam como resolver alguns dos problemas no desenvolvimento de SIE foram desenvolvidos no capítulo 5 dois projectos de um SIE em tecnologias diferentes mas com com um mesmo objectivo final.

Após conclusão dos dois projectos, verifiquei que foi possível satisfazer todos os requisitos apresentados. No que respeita ao esforço despendido a nível de tempo, o projecto *Open Source* foi mais moroso, em parte pela minha falta de experiência em JEE mas também porque penso que no que toca a configurações os produtos da Microsoft são mais intuitivos do que os *Open Source* e por esse motivo o projecto da Microsoft foi desenvolvido com maior rapidez.

Uma outra comparação que seria interessante de ser feita no futuro, seria em termos de custos. Ou seja, se o valor pago pelas licenças da Microsoft, compensam as horas a mais que possam ser gastas no desenvolvimento de SIE. Também seria interessante comparar soluções de *failover clustering* e *disaster recovery* em ambas as tecnologias.

# Referências

[1] Walter Vieira. Slides ASI - ISEL 2017

[2] António Gonçalves. Beginning Java EE 7 - Apress 2013

[3] Juval Löwy. Programming WCF Services - O'Reilly 2010

[4] Documentação MYSQL. https://dev.mysql.com/doc/ - 2018

[5] ISO/IEC 9126-1:2001. https://pt.wikipedia.org/wiki/ISO/IEC_9126 - 2018

[6] Transacções distribuidas. https://docs.oracle.com/database/121/TTJDV/jta.htmTTJDV237 -2018

[7] Replicação MySQL. https://www.devmedia.com.br/mysql-replicacao-de-dados/22923 - 2018

[8] Transacções EJB com Glassfish. https://www.byteslounge.com/tutorials/ejb-multiple-datasource-transaction-example - 2018

[9] two-phase commit. http://lig-membres.imag.fr/krakowia/Files/MW-Book/Chapters/Transact/transact-body.html -2018

[10] Airton Lstori. Replicação MySQL com PHP. https://pt.slideshare.net/MySQLBR/replicao-mysql-e-php -2018

[11] Sacha Krakowiak. Middleware Architecture with Patterns and Frameworks. Creative Commons license -2009

[12] D. Gelernter. Generative Communication in Linda. In Proceedings of ACM Transactions on Programming Languages and Systems. 1985

[13] MYSQL Galera. https://www.fromdual.com/galera-cluster-for-mysql - 2018

# Organização do CD

Este documento é acompanhado de um CD que contém a seguinte estrutura de pastas:

- **Windows**:
  - **SQL Server**: Código fonte para implementação da camada de dados.
  - **WCF**: Código fonte para implementação da camada de aplicacional.
- **Open Source**:
  - **MYSQL**: Código fonte para implementação da camada de dados.
  - **JEE com EJB**: Código fonte para implementação da camada de aplicacional.
- **Vídeos**:
  - **Windows**: Vídeo que demonstra o funcionamento do projecto em Windows.
  - **Open Source**: Vídeo que demonstra o funcionamento do projecto em Open Source.