INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Criação e atualização de aplicações para área de retalho por MDA

ANDRE DE SOUSA RIBEIRO
Outubro de 2015

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Criação e atualização de aplicações para área de retalho por MDA

André de Sousa Ribeiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Arquiteturas, Sistemas e Redes

Orientador: Ana Maria Dias Madureira Pereira

Co-orientador: Paulo Manuel Baltarejo de Sousa

Porto, Outubro 2015

Resumo

Nos últimos anos tem-se verificado a constante evolução dos mercados em plataformas na Internet como forma de melhoria não só dos serviços prestados, mas também para o aumento de vendas de produtos e respetiva internacionalização dos mesmos. Este aumento da procura por este tipo de *softwares*, assim como a constante evolução e atualização dos mesmos tem contribuído para que estas aplicações evoluam em termos de funcionalidades e complexidade. Isto contribui cada vez para a dificuldade de formar equipas capazes de manter e desenvolver estes sistemas sem que comprometa em grandes custos para as organizações.

Neste sentido surgiram diversas ferramentas que permitem criar soluções pré desenvolvidas de aplicações na Internet denominadas de "*E-commerce applications*". Estas plataformas, apesar do utilizador não ter obrigatoriamente que deter conhecimentos de programação para proceder à sua instalação, são bastante restritas tanto aos serviços que podem ser usados, e na sua escalabilidade, visto que normalmente correm em servidores específicos e por vezes as configurações necessárias para instalação tornam-se bastante complexas de ser efetuadas.

Pretende-se no âmbito desta dissertação de mestrado propor um modelo de uma arquitetura de um sistema baseado em mecanismos MDA para a área de retalho, particularmente em ambientes de *e-commerce*. Serão inicialmente sistematizados os principais tipos de e-commerce numa perspetiva de evolução histórica. Será igualmente enquadrado o MDA no desenvolvimento de um sistema de *e-commerce*. Neste sentido, serão equacionadas as diferenças entre o modelo típico de desenvolvimento de *software* e o desenvolvimento de software orientado pelas metodologias do MDA.

No processo de especificação e desenvolvimento do modelo proposto será realizada uma análise de requisitos, assim como, a proposta do modelo da arquitetura de um sistema baseado em mecanismos MDA, tendo como orientação os requisitos e arquitetura definida na fase de análise. Finalmente no sentido de analisar o resultado esperado para um sistema orientado por metodologias definidas por MDA, serão realizado alguns testes no sistema desenvolvido de forma a analisar o seu desempenho e validar a sua adequabilidade no âmbito do processo de desenvolvimento de sistemas *e-commerce*

Palavras-chave: MDA, Framework, Internet, Gerador, E-commerce, Modelos

Abstract

In recent years there has been the constant evolution of Internet platforms in markets as a way to improve not only the services provided, but also to increase product sales and respective internationalization. This increased demand for this type of software, as well as the constant evolution and update them has contributed to these applications evolve in terms of functionality and complexity. This contributes increasingly to the difficulty of forming teams capable of maintaining and developing these systems without compromise in large costs for organizations.

In this sense, they emerged several tools that let you create pre solutions developed applications on the Internet called "E-commerce applications". These platforms, although the user does not have necessarily had any programming knowledge to carry out its installation, are quite restricted both services you can use, and scalability, as usually running on specific servers and sometimes the necessary settings for Installation become quite complex to be made.

It is intended as part of this master's thesis propose a model of an architecture of a system based on MDA mechanisms for the retail area, particularly in e-commerce environments. They will initially systematized the main types of e-commerce in a historical perspective of evolution. It will also framed the MDA to develop an e-commerce system. In this sense, it is equated differences between the typical model of software development and software development methodologies guided by the MDA.

The specification and development process of the model an analysis of requirements will be held, as well as the model proposed architecture of a system based on MDA mechanisms, with the guidance requirements and architecture defined in the analysis phase. Finally in order to analyze the expected result for a system-oriented methodologies defined by MDA will be conducted some tests on the system developed to analyze its performance and validate their suitability within the e-commerce systems development process.

Keywords: MDA, Framework, Web, Generator, E-commerce, Models

Agradecimentos

Concluída a dissertação, gostaria de deixar algumas palavras de agradecimentos às pessoas que tornaram a sua realização possível.

Em primeiro lugar, há minha namorada pelo apoio e incentivo nos bons e maus momentos pois sem ela não seria possível esta concluída.

Em segundo ao meus pais que desde o momento que entrei no mestrado me suportaram com um apoio fundamental para levar o barco até ao fim e nunca desistir dos meus objetivos na vida.

Por fim, aos meus orientadores, Ana Madureira e Paulo Baltarejo, pela partilha de conhecimento e disponibilidade para me orientar no desenrolar da dissertação.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Código Fonte

Acrónimos e Símbolos

Lista de Acrónimos

ACEPI Associação do Comércio Eletrónico e Publicidade Interativa

ADM Architecture Driven Modernization

ASP (Microsoft) Active Server Pages

B2B Business to Business

B2C Business to Commerce

B2E Business to Employee

C2B Consumer to Business

C2C Consumer to Consumer

CIM Computation Independent Model

CWM Common Warehouse Metamodel

EC Electronic Commerce

e-CRM Electronic Customer Relationship Management

EDI Electronic Data Interchange

EFT Electronic Funds Transfer

ERP Enterprise Resource Planning

FTL Freemarker Template Language

HTTP Hypertext Transfer Protocol

JVM Java Virtual Machine

LDAP Lightweight Directory Access Protocol

LINQ Language Integrated Query

MBE Model Based Engineering

MDA Model Driven Architecture

MDD Model Driven Development

MDE Model Driven Engineering

MGC Motor de Geração de Código

MOF MetaObject Facility

MTF Model Transformation Framework

MVC Model View Controller

OMG Object Management Group

PIM Plataform Independent Model

PSM Plataform Specific Model

SI Sistemas de Informação

SQL Structured Query Language

TI Tecnologias de Informação

UI User Interface

UML Unified Modeling Language

WCF Windows Communication Foundation

XML Extensible Markup Language

1 Introdução

O surgir das Tecnologias de Informação (TI) potenciaram não só o aparecimento de novas indústrias como as consultoras de sistemas de informação ou as relacionadas com os negócios da Internet, mas também tem posto em causa alguns modelos tradicionais de desenvolvimento de negócio.

A Internet mudou a forma como fazemos compras. Mesmo as pessoas que não fazem regularmente compras usam a pesquisa *online* para ver preços, especificações de produtos e disponibilidade antes de realmente ir ao destino comprar. Um estudo do Statista [1] indica que em 2000, 1.024 biliões de utilizadores usaram a Internet em todo o mundo e até 2015 o número cresceu exponencialmente para 3.17 biliões. Um outro estudo da mesma empresa indica que em 2011 o número de compradores *online* a nível mundial era de 792.6 milhões e prevê que em 2016 seja de 1.321 biliões [2]. O sector do retalho *online* está em evolução e substitui cada vez mais as tradicionais lojas de venda de retalho.

A crescente complexidade dos sistemas de informação e a dificuldade de os manter, tem sido um aspeto que tem gerado mais discussão, no sentido de encontrar soluções para este problema. Um sistema de informação bem-sucedido é aquele que é produzido dentro do prazo e nos custos estimados; fiável (sem erros e disponível) e que pode ser mantido facilmente e com baixo custo. Concluiu que atualmente ainda são poucos os sistemas que apresentam estes requisitos [3].

Segundo Alberto Silva e Carlos Videira [3] as principais fases do processo de desenvolvimento de *software* são a Conceção, Implementação e Manutenção, sendo que a Manutenção engloba 67% dos custos das diversas tarefas de desenvolvimento de software.

Durante muito tempo pensou-se que o simples facto de aumentar o tamanho da equipa de

desenvolvimento seria o suficiente para acompanhar o processo de criação, evolução e manutenção de *software*. Mas como se tem vindo a notar, o crescimento da necessidade de desenvolvimento de *software*, assim como futuras atualizações e manutenção aumentou drasticamente.

Com o início do desenvolvimento dos Sistemas de Informação (SI), todas as empresas criavam os seus próprios produtos de *software*, e eram responsáveis por todo o ciclo de vida do *software*. Contudo, à medida que se vai podendo medir os valores investidos em cada parte do ciclo de vida, esta realidade começa a alterar-se. As organizações esforçam-se por melhores resultados do seu investimento em desenvolvimento, mantendo o *software* operacional durante o maior período de tempo possível, aumentando assim o período de manutenção dele. Além disso, a disponibilização de ferramentas e linguagens de desenvolvimento *open source* atraiu as atenções, visto que é possível manter código desenvolvido por outros, ou vice-versa. O número de dependências de *software* externo, assim como a ligação com ferramentas empresariais também aumentaram.

Contudo, a manutenção é uma atividade dispendiosa. A manutenção de *software* inclui a correção de erros, alterações e melhoramentos ao *software* operacional. Por esta razão tinha que haver uma solução que fosse ao encontro do melhoramento do processo de desenvolvimento e manutenção de *software* por forma a reduzir tempo e custos. Muitos dos processos que se encontram nas aplicações são muito repetitivos, mudando apenas o contexto ao qual são aplicados. Nesse sentido, surge a necessidade de alterar a tradicional forma de desenvolvimento de *software*, introduzindo conceitos de sistemas orientados por modelos e transformações, de forma a ir de encontro a vários tipos de negócio. O sistema passa a ser padronizado em modelos que, através de *inputs* do utilizador e processado por transformações, geram código, de forma a ir ao encontro das necessidades do negócio.

Através de um esforço da organização conhecida por Object Management Group (OMG), surge uma metodologia para ir de encontro às necessidades de desenvolvimento de *software* [4]. Uma metodologia capaz de reduzir o tempo e o custo do desenvolvimento de *software*, através de uma arquitetura orientada a modelos aplicáveis a qualquer domínio de negócio. Estes modelos possibilitam reduzir o tempo de desenvolvimento e manutenção e evitar processos repetitivos de desenvolvimento de código. Tendo em conta as necessidades pretende-se no âmbito desta dissertação propor um sistema desenvolvido por Model Driven Architecture (MDA) que permita auxiliar e reduzir o tempo e custo da produção de *software* na área do sector do retalho nas mais diversas fases de conceção do mesmo [5].

1.1 Contribuições

Pretende-se no âmbito desta dissertação de mestrado propor um modelo de uma arquitetura de um sistema baseado em mecanismos MDA para a área de retalho, particularmente em ambientes de *e-commerce*. Serão inicialmente sistematizados os principais tipos de e-commerce numa perspetiva de evolução histórica. Será igualmente enquadrado o MDA no desenvolvimento de um sistema de *e-commerce*. Neste sentido, serão equacionadas as diferenças entre o modelo típico de desenvolvimento de *software* e o desenvolvimento de software orientado pelas metodologias do MDA.

No processo de especificação e desenvolvimento do modelo proposto será realizada uma análise de requisitos, assim como, a proposta do modelo da arquitetura de um sistema baseado em mecanismos MDA, tendo como orientação os requisitos e arquitetura definida na fase de análise. Finalmente no sentido de analisar o resultado esperado para um sistema orientado por metodologias definidas por MDA, serão realizado alguns testes no sistema desenvolvido de forma a analisar o seu desempenho e validar a sua adequabilidade no âmbito do processo de desenvolvimento de sistemas *e-commerce*.

1.2 Objetivos

Esta dissertação tem como objetivo a criação de um Motor de Geração de Código (MGC). Esta ferramenta irá permitir aos programadores otimizar o seu trabalho, poupando assim tempo na fase de desenvolvimento, assim como nas fases de manutenção e atualização. Este para o conseguir utilizará mecanismos orientados ao desenvolvimento de aplicações por MDA. Permitirá aos utilizadores descrever o sistema a ser implementado através de uma linguagem informar que servirá para documentar o sistema e ao mesmo tempo como *input* do MGC. O sistema irá utilizar o *input* definido pelo utilizador e os módulos disponíveis e selecionados pelo utilizador para definir qual a informação a ser gerada.

O MDA possibilita um desenvolvimento de sistemas homogénicos que permitem ser aplicados nos mais diversos tipos de negócio. No contexto desta dissertação iremos aplicar a suas metodologias para gerar artefactos de *software* no âmbito de sistemas de retalho online.

1.3 Estrutura do Documento

Estando definidos os objetivos e contribuições o segundo capitulo irá descrever o **Estado da Arte** onde serão englobado e enquadrada a Introdução aos sistemas de retalho, a sua evolução ao longo dos anos, os vários tipos de E-Commerce (EC) existentes assim de como proceder à criação de um EC. Este capítulo também irá fazer a introdução ao MDA, descrever a sua visão, explicando os seus conceitos e definições. Uma comparação entre o tradicional modelo de desenvolvimento de *software* e o desenvolvimento por MDA também será descrito neste capítulo, sob forma de entender como através do compromisso feito pelo MDA e através das várias vertentes, é possível definir estratégias para desenvolver um sistema de EC otimizado. Serão por fim descritas ferramentas existentes de modelação por MDA, assim como as ferramentas e tecnologias que serão usadas para desenvolver o sistema de MDA nesta dissertação.

No terceiro capítulo com o nome de **Mecanismo de Geração de Código,** será responsável por descrever o desenvolvimento do sistema produzido, sendo que este será dividido em 3 secções: a análise, onde se encontram identificados os requisitos funcionais e não funcionais, a arquitetura do sistema proposto e o modelo de dados que irá suportar a plataforma desenvolvida; a implementação, onde são descritos os detalhes de implementação e da solução proposta para os objetivos desta dissertação.

O quarto capítulo com o nome **Validação do Mecanismo de Geração de Código**, será onde o motor de geração de código será posto a prova simulando a produção de artefactos de um protótipo de sistema EC. Desta forma será possível constatar os resultados obtidos, provando assim a sua utilidade nesta área de negócio.

Por fim no quinto capítulo com o nome de **Conclusão e trabalho futuro,** irão constar as conclusões do estudo efetuado no âmbito desta dissertação e fazer referência a trabalho futuro a ser realizado com base nos resultados obtidos.

2 Estado da Arte

Em qualquer trabalho de investigação e desenvolvimento torna-se necessário na fase de levantamento de requisitos estabelecer um primeiro contato com os conceitos relacionados com o tema em questão. Com essa pesquisa e procura é possível apurar quais as melhores práticas de desenvolvimento e implementação do que se pretende realizar. Este processo de investigação, que consiste em pesquisar o máximo possível de informação e de desenvolvimentos na área de estudo em questão, denomina-se de Estado da Arte.

Assim sendo, serão abordados neste capítulo os estudos e pesquisas efetuados relacionados com os dois principais temas desta dissertação: MDA e o setor do Retalho. Serão também referidas as tecnologias e ferramentas usadas para a conceção deste trabalho.

2.1 Sistema de Retalho

O setor do retalho está em constante evolução. Os retalhistas lidam hoje com as forças complexas da globalização e diversificação que agitam o setor. Nesta secção, será feita uma pequena abordagem da história do setor e da atualidade, focando a importância que as TI têm tido neste setor. Nesse sentido, será feito um *overview* a sistemas de comércio eletrónico, desde as suas vantagens, arquitetura e processo de desenvolvimento.

2.1.1 Perspetiva Histórica

A história do retalho é de evolução constante, sendo que o desenvolvimento social e tecnológico tem contribuído para este processo evolutivo. David Roth organiza a história deste setor em cinco períodos distintos, que vão desde o início da humanidade até aos dias de hoje [6].

- **Comprador como "Hunter Gatherer" (Pré-História)** - Desde sempre que o ser humano evoluiu no sentido de encontrar e adquirir de algo que necessita para a própria sobrevivência ou que simplesmente "quer". Nesta altura, as peles de animais eram usadas para diversos fins, no entanto, o impacto mais duradouro foi dar origem à Hudson's Bay Company em 1670. A colheita de peles de animais que eram abundantes na América do Norte permitiu à empresa controlar o comércio de peles durante vários séculos e acumular muita riqueza;
- **Comprador Neolítico (8,000AC a 2,000 AC)** - Uma vez tornadas eficientes as sociedades "*hunter/gatherer*", foi introduzida a agricultura que fornece a plataforma para a segunda etapa evolutiva para o retalho. O desenvolvimento de ferramentas para a agricultura, a domesticação de animais permitiu a produção de excedentes para venda ou troca que, por sua vez, gerou uma infraestrutura de centros de comercialização e distribuição. É também neste período que surge o primeiro mecanismo de cálculo conhecido, o que se torna num avanço significativo para a gestão do setor;
- **Dinheiro (2,000 AC a 1600DC)** - O comércio como a produção e a distribuição de produtos agrícolas cresceu rapidamente. No entanto, sendo realizado maioritariamente pelo sistema de escambo[1] e, sendo cada vez maior a diversidade de produtos, surgiu a necessidade de outro método de valor – o "dinheiro" (na forma de moedas, pérolas, entre outros);
- **Produção em massa (Século XVII a século XIX)** - A industrialização provocada pela Revolução Industrial entre os séculos XVII e XIX trouxe a produção em massa de produtos a baixo custo e, com isso, uma nova abordagem para o retalho a nível de lojas (especializadas em diversos tipos de produtos);
- **Compra moderna (Desde século XX)** - Este período surgiu em meados do século XX com a introdução do conceito de loja como um local onde milhares de produtos diferentes podem ser adquiridos num único local. Além disso, com a introdução das primeiras formas de

[1] Prática que permite realizar uma troca comercial sem o envolvimento de moeda ou objeto que se passe por esta, e sem equivalência de valor.

entretenimento e hospitalidade o conceito de “compra” tornou-se uma atividade de lazer. Desde então, outros formatos foram surgindo como o supermercado do centro comercial ou o hipermercado. Mas talvez um dos mais importantes seja a introdução da compra *online*.

2.1.2 A evolução da compra *online* e o E-Commerce

Centralizando agora a história no sentido da loja *online,* torna-se importante referir como esta surgiu tal como a conhecemos nos dias de hoje. Para tal contribuiu o aparecimento dos computadores, dos cartões de crédito e, por fim, a Internet [6].

Foi a J. LYONS & CO, uma casa de chá, restaurante e hotel britânica a pioneira na introdução de um computador no setor do retalho. O computador Lyons Electronic Office [7] , construído em 1951, foi projetado pelo Dr. John Pilkerton e John Simmons para lidar com as contas da empresa e logística. Na década de 1960, os computadores no setor do retalho começaram a ser usados para gerir os dados dos clientes e acelerar os processos de contabilidade, assim como o grupo Lyons havia feito na primeira tentativa. Walmart [8] e outras lojas de descontos como a Kmart [9] e Target [10] , foram as primeiras a adotar a tecnologia do computador e, em meados da década de 1980 e início de 1990 pequenas lojas já tinham computadores. Em geral, usavam-no como sistemas que permitissem controlar o *stock* da loja e, mais tarde, com algum *software* que já permitisse gerir as folhas de pagamento dos funcionários e as contas do dia-a-dia. Com isto, o tempo de resposta para os sistemas de *stock* melhorou significativamente, assim como a gestão dos próprios funcionários tornou-se mais simples.

Em 1946, John Biggins, um banqueiro de Nova York, apresenta o cartão “CHARGE-IT”. Este cartão foi usado para comprar bens de uma loja, sendo o papel de compra enviado para o Banco Biggins [11]. O banco pagou o dinheiro aos comerciantes, cobrando o mesmo valor ao cliente que apresentou o papel de compra. A proliferação dos cartões de crédito incentivou uma mentalidade “compre agora, pague depois”, alimentando assim mais gastos neste setor. Além disso, com o surgir das compras *online*, o cartão de crédito tornou-se o formulário padrão de pagamento que tem alimentado o crescimento das transações de comércio eletrónico.

Por fim, a introdução e evolução da Internet mudou a forma como fazemos compras. A sua origem data 1960 quando o Governo dos Estados Unidos iniciou a investigação em redes informáticas, mas, foi a partir de meados da década de 1990 que ganhou reconhecimento e, desde então, tem crescido

exponencialmente. A Internet oferece aos clientes um acesso sem precedentes à informação, a capacidade de consumir, compartilhar e criar conteúdos e envolver-se num comércio como nunca antes.

Em 1994/1995 surgem duas grandes empresas neste setor, o eBay [12] e o Amazon [13], as primeiras a permitir transações eletrónicas e, com isso, consideradas pioneiras no conceito de Electronic Commerce ou, como muitas vezes é apresentado E-Commerce (EC). Mas, o conceito de EC não surgiu pela primeira vez neste contexto. Este foi originalmente concebido para descrever o processo de realização de transações comerciais eletronicamente usando a tecnologia da Electronic Data Interchange (EDI) e Electronic Funds Transfer (EFT). Estas tecnologias, que apareceram pela primeira vez no final dos anos 1970, permitiram a troca de informações e a realização de transações eletrónicas entre empresas, tipicamente na forma de ordens de compra e faturas eletrónicas. O EDI e o EFT foram assim as tecnologias que lançaram as bases para o que hoje conhecemos como E-commerce e, com a evolução da Internet e introdução da Internet no início de 1990, as aplicações de EC rapidamente se expandiram [14].

Atualmente, a Amazon e o eBay estão no topo das vinte marcas do setor do retalho mais valiosas do Mundo, estando a Amazon em primeiro lugar com uma avaliação, em 2013, de 40,334 milhões de euros e o eBay em quarto com 15,653 milhões de euros [6].

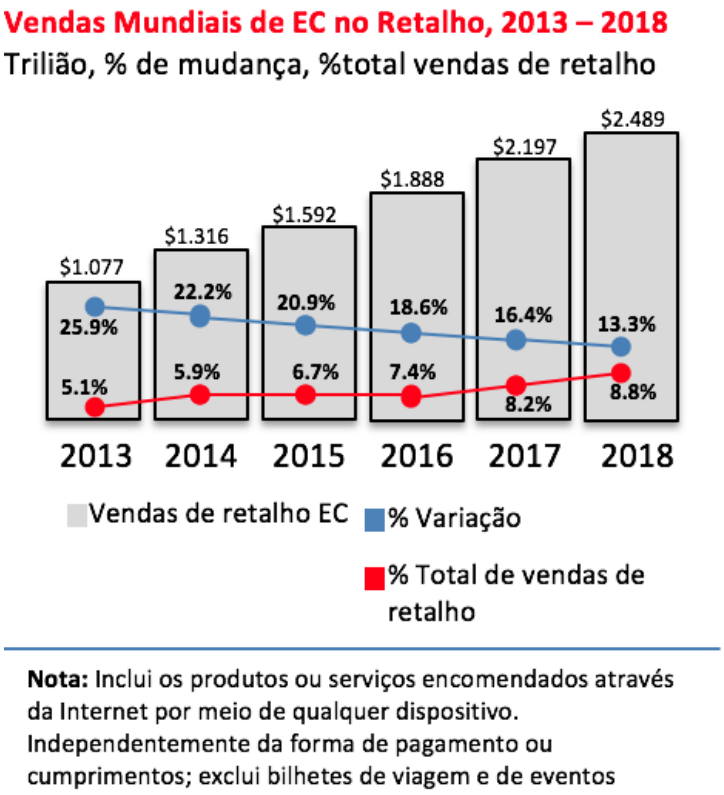

Figura 1 – Retail Ecommerce Sales Worldwide [16]

Portugal não ficou de fora deste ciclo de evolução. Um estudo da Associação do Comércio Eletrónico e Publicidade Interativa (ACEPI) afirma que 59% da população total já está ligada à Internet, um crescimento de 14% nos últimos 2 anos. No que diz respeito a compras *online*, entre Setembro de 2011 e Fevereiro de 2012 os portugueses compraram no valor de 1,630 milhões de euros. Em média e, no espaço de 6 meses, realizam 8 compras *online* por pessoa, gastando uma quantia média de 427 euros por pessoa. O estudo afirma também que os portugueses estão na linha da frente a nível das compras *online*: 97% dos utilizadores recorre à Internet para pesquisar informação sobre os bens que pretende adquirir, e 78% efetuam realmente a compra *online* [17].

Outro estudo mais recente, "European B2C E-Commerce Report 2015", feito pela European Ecommerce e que está resumidamente apresentado na Figura 2, indica que em Portugal 6,9 milhões de pessoas estavam ligadas à Internet, o que equivale a uma taxa de 66%. Por outro lado, 3.2 milhões de euros de consumidores portugueses adquiriram bens e/ou serviços *online* em 2014. No total, gastaram 2.9 biliões de euros *online*, o que resultou numa despesa média de 911 euros por comprador *online* [18].

Comparando ambos os estudos nota-se um aumento dos valores ao longo destes últimos anos, não só no acesso à Internet, mas também nos valores associados às compras *online*. Este último estudo afirma ainda que o E-Commerce cresceu 13% em 2014, o que corresponde a uma aposta por parte das empresas nas vendas *online*.

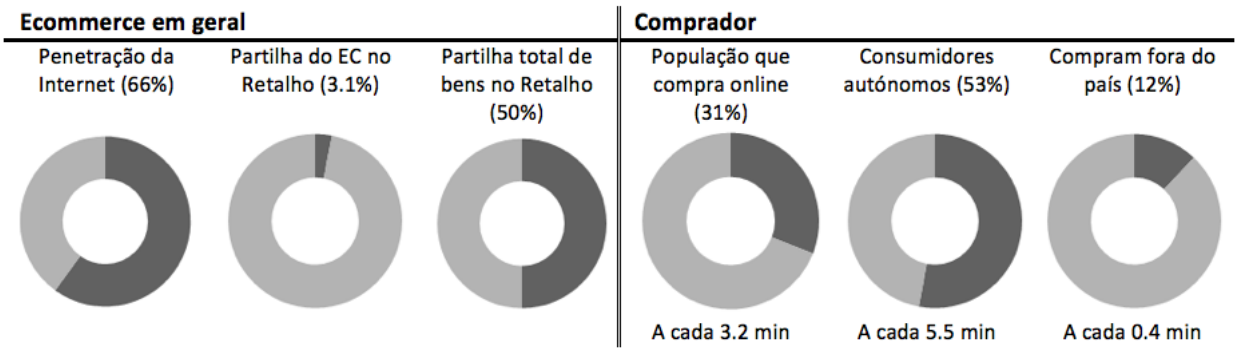

Figura 2 – Dados de Portugal do estudo "European B2C E-Commerce Report 2015"(Adaptado de [18])

2.1.3 Visão Geral de sistemas E-Commerce

EC simboliza o uso da Internet para comprar, vender, transportar dados de comércio, bens ou serviços [19]. Este tipo de sistemas tem como grande diferencial a possibilidade de abranger tanto mercados grandes como nichos específicos. Em termos gerais, engloba o processo *online* de desenvolvimento, *marketing*, venda, entrega, atendimento e pagamento por produtos e serviços comprados por empresas virtuais de clientes conectados à rede, com o apoio de uma rede mundial de parceiros comerciais.

Além disso, com o rápido crescimento da Internet e outras tecnologias de informação, os sistemas EC oferecem funcionalidades e novas formas de fazer negócios na qual as empresas não podem ignorar. Permite a uma empresa alcançar vantagens competitivas de: redução de custos com base em custos reduzidos de publicidade/promoção; diferenciação do produto, personalização de produtos e resposta atempada ao mercado; foco no cliente e melhoria do serviço prestado ao cliente. Além das empresas, consumidores e a própria sociedade beneficiam deste tipo de sistemas. Na Tabela 1 é possível observar em detalhe as inúmeras vantagens que este tipo de sistemas pode trazer [20].

Tabela 1 – Benefícios de sistemas EC (adaptado de [20])

Benefício	**Descrição**
Promoção do produto	Reforço da imagem e marca do produto, permitindo a interatividade e personalização dos conteúdos e publicidade, com base no perfil do cliente ou de entrada.
Economização de custos	Através da utilização da Internet como meio de transmissão de informação é possível diminuir o custo da prestação de informações aos clientes, incluindo pessoal, telefones e custos de impressão.
Informação oportuna	Devido à sua natureza instantânea, é possível a redução do tempo do ciclo necessário para produzir e entregar informação e serviços.
Tempo de expedição diminuído	Os clientes enviam as suas expedições eletronicamente para a base de dados, o que permite diminuir o tempo associado à remessa no sistema de correio.
Consistência da informação	Permite a consistência e a precisão da informação através da partilha de informações e uso de formulários eletrónicos para fazer negócios.
Melhor atendimento ao consumidor	A capacidade de fornecer respostas *online* para os problemas através de guias ou secções com perguntas e respostas comuns já predefinidas e a interação com o correio eletrónico 24 horas por dia, 365 dias por ano, ajuda na promoção da confiança e retenção de clientes.
Melhor relação com o consumidor	De cada vez que um cliente entra numa pagina da Internet para fazer uma compra ou ver determinado produto, todo este tipo de informação é gravado para que, no futuro, o atendimento possa ser mais inteligente e ir de encontro às expetativas do mesmo.
Customização e oferta de produtos	A capacidade de customização de produtos *online* de acordo com as necessidades do cliente. Grande oferta de produtos e serviços e a capacidade de fazer compras a qualquer momento e em qualquer lugar.
Melhoria dos serviços públicos	Com o aparecimento dos E-Government permite que sejam fornecidas de forma mais fácil e acessível determinado tipo de informação pública, podendo assim ser acedida mais rapidamente.
Melhoria do nível de vida	Permite o acesso a produtos/serviços com um preço mais acessível. Permite o acesso de pessoas residentes em locais mais rurais ou países em desenvolvimento a uma gama e variedade de produtos/serviços que até então não tinham.

Apesar de todas as vantagens, os sistemas EC têm também algumas limitações, as quais é possível observar em detalhe na Tabela 2 [20].

Tabela 2 – Limitações de sistemas EC [20]

Limitação	**Descrição**
Largura de banda insuficiente	Em países menos desenvolvidos, a largura de banda é insuficiente para estes tipos de serviços e o acesso à Internet ainda é caro.
Integração com outro tipo de sistemas	Dificuldades na integração da Internet e *software* de EC com alguns sistemas, aplicações e bases de dados já existentes.
Falta de padrões de segurança	Falta de normas universalmente aceites para a qualidade, segurança e confiabilidade deste tipo de sistemas.
Resistência à mudança	Resistência parte das pessoas em mudar de uma loja física para uma virtual. Falta de confiança e perceção de que este tipo de sistemas são caros e podem não ser garantidos.

2.1.4 Tipos de E-Commerce

Existem vários tipos de sistemas EC, no entanto diferentes autores fazem caracterizações diferentes, uns considerando a existência de mais tipos que outros. Apresentam-se de seguida alguns tipos de sistemas EC [21]:

- **Business-To-Business** - Um sistema Business-to-Business (B2B) refere-se a atividades de negócios *online* entre as empresas, ou seja, tanto os vendedores como os compradores são organizações. Produtos e serviços são vendidos tipicamente para outras empresas que os utilizam nos seus processos de fabrico de novos serviços e produtos. Neste tipo de negócio, os processos de venda implicam a criação de uma maior relação entre os dois negócios e uma negociação feita em vários níveis de atuação e negociação. Com o surgir da Internet, as empresas têm investido bastante em sistemas B2B de modo a economizarem mais custos, reduzirem os prazos de entrega e manterem-se competitivas.

  Como exemplos deste tipo de negócio temos a Intel que vende micro processadores para Hewlett Packard [22] ou ainda a John Deere [23] que vende máquinas de cortar relva para a Home Depot [24].

- **Business-To-Consumer** - Um sistema Business-to-Commerce (B2C) engloba todas as atividades de negócios *online* em que as empresas oferecem produtos e serviços aos consumidores. Os vendedores são organizações e os compradores são indivíduos, ou seja, os produtos deste tipo de negócio são vendidos diretamente ao consumidor final e tipicamente o processo é mais curto. O consumidor é encorajado a comprar o produto no momento imediato. Tipicamente num sistema EC o consumidor consulta a informação do produto no portal e paga a encomenda antes de a receber no processo de "*checkout*", usando para isso cartão de débito ou crédito ou outro mecanismo digital. Este tipo de sistemas é caracterizado por uma concorrência intensa, baixas barreiras de entrada no mercado e um baixo grau de fidelização dos clientes.

  As vendas no setor o retalho através de canais *online*, serviços financeiros, serviços de viagens, e produtos digitais (por exemplo, serviços de música e filme por *streaming*) são formas populares de sistemas B2C. Exemplos concretos deste tipo de negócio são a Amazon [13] que vende livros *online* para os consumidores, a Dell [25] que vende computadores *online* para os clientes ou ainda a Best Buy [26] que vende impressoras para clientes através da Internet.

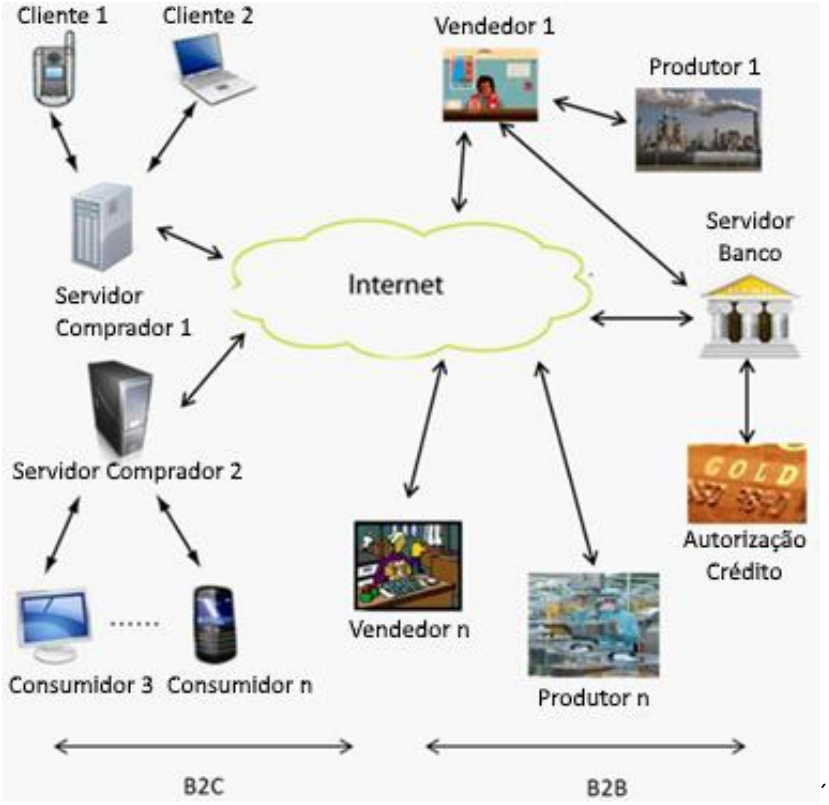

Figura 3 – Diferenças entre negócios B2B e B2C [21]

A pretende mostrar a diferença entre um sistema B2B e B2C, colmatando assim as diferenças expostas anteriormente.

- **Consumer-To-Business** - Num sistema Consumer-To-Business (C2B) os consumidores (pessoas) comercializam produtos e serviços para empresas. Este modelo de negócio é uma inversão do modelo B2C onde empresas oferecem produtos e serviços aos consumidores.

  Exemplos deste tipo de negócio é a Fotolia [27], onde fotógrafos e *designers* vendem obras para as empresas ou a Survey Scout [28], onde as pessoas se oferecem para responder a questionários das empresas e as mesmas pagam às pessoas por este serviço;

- **Consumer-To-Consumer** - Num sistema Consumer-To-Customer (C2C), um indivíduo vende produtos ou serviços a outros indivíduos.

  Exemplos deste tipo de negócio são as páginas de Internet de leilão *online,* como o eBay [12] e Craigslist [29], onde os consumidores podem comprar e vender produtos e/ou serviços que

utilizam sistemas de pagamentos *online* como o PayPal [30] para enviar e receber dinheiro com facilidade e segurança;

- **Business-To-Employee** - Um sistema Business-To-Employee (B2E) é geralmente usado pelas organizações para transmitir informações e serviços internamente para os seus colaboradores. Este tipo de negócio ajuda também as organizações na redução de custos administrativos e eliminando as despesas relacionadas com a impressão, *e-learning* e viagens.

  Exemplos deste tipo de negócio são a divulgação de anúncios, *e-learning*, solicitação de viagens *online* ou pedidos de fornecimento online;

- **E-Government** - Um sistema E-Government é usado para fornecer informações e serviços públicos aos cidadãos e (considerado Government-To-Citizen) e para parceiros de negócios e fornecedores (considerado Government-To-Business). O E-Government é também uma forma eficaz de realização de transações comerciais com os cidadãos e as empresas e dentro dos próprios governos.

2.1.5 Retalho e sistemas de E-Commerce

Tal como já foi referido anteriormente, o setor do retalho encontra-se em constante evolução e a utilização de sistemas EC é prova disso. Têm surgido novos negócios de EC para retalho, mas também muitos retalhistas convencionais têm adotado estratégias *online*. Retalho realizado através da Internet é chamado de E-Tailing e os vendedores que realizam negócios *online* de retalho são chamados de retalhistas *online*. A pretende mostrar essa evolução e como o EC mudou a relação entre clientes e retalhistas [31].

Numa cadeia de abastecimento de retalho convencional, os clientes são responsáveis por comprar os produtos numa loja (local do retalhista), assumindo assim a linha final na cadeia de distribuição. A localização da loja e o *stock* de produtos são fatores importantes em lojas de retalho tradicional. Uma boa localização pode constituir numa rede de clientes maior/fixa, mas isso pode também acarretar custos (tais como aluguer do espaço, estabelecimento, etc.) que muitas vezes se refletem no preço de venda mais alto, diminuindo desta forma a competitividade. Por outro lado, o retalhista mantém um determinado nível de *stock* na loja, que é reabastecido por centros de distribuição regionais onde as mercadorias, que são provenientes de uma vasta gama de fornecedores, estão armazenadas. A nível de relação com o

cliente, sistemas deste tipo permitem manter o contacto “cara-a-cara” podendo assim proporcionar uma relação mais estável.

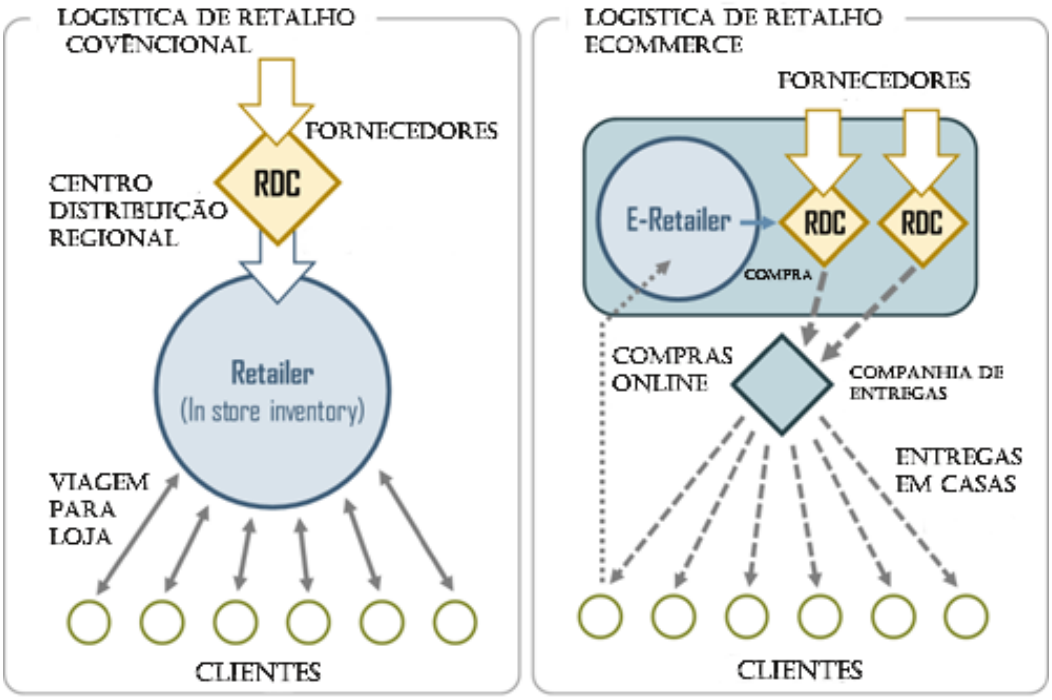

Figura 4 – Diferença entre lojas de retalho convencional e sistemas EC [31]

No que diz respeito ao uso de sistemas EC para retalho, alguns destes fatores comportam-se de maneira diferente. A nível de localização esta é mais flexível, permitindo a utilização de locais de menor custo que não teriam sido considerados no caso de uma loja convencional. No que diz respeito aos clientes, estes estão ligados “diretamente” à cadeia de abastecimento pois, efetuar uma encomenda de um produto, atinge “diretamente” o centro de distribuição. A nível de relação com o cliente, esta pode ser considerada por um lado menos estável, pois os contactos são estabelecidos quase de forma anónima, mas por outro permite uma resposta mais rápida e eficiente a dúvidas/queixas que possam surgir.

2.1.6 Modelos de negócio no E-Tailing

Por forma a entender melhor o E-Tailing, Turban et al. [32] colocam-no na perspetiva de um retalhista ou um fabricante que vende a consumidores individuais, ou seja, como um sistema do tipo B2C. Apresentam-se de seguida os modelos de negócio com base no canal de distribuição usado:

- Retalhistas que vendem em lojas físicas e *online*, como é o caso da QVC [33] e Lands' End [34];
- Fabricantes que vendem *online* e através dos retalhistas, como é o caso da Dell. O principal fator de sucesso deste modelo é a capacidade de oferecer produtos personalizados a um custo razoável;
- Retalhistas que apenas vendem *online*, como é o caso da Amazon;
- Retalhistas que *lojas online* para complementar suas atividades, como por exemplo a Walmart [8] e a HomeDepot [24]. No entanto, também se tem assistido a uma tendência inversa: por exemplo, a Apple abriu lojas físicas e a Dell [25] vende os seus produtos em locais de armazenamento de parceiros, como Best Buy [26] e Staples [35]. Através deste tipo de estratégias multicanal, as empresas oferecem várias opções para o cliente adquirir;
- *Shoppings* online, que incluem muitas lojas online num mesmo site. Neste caso, as lojas apresentadas podem pagar uma subscrição ou uma comissão para manutenção do *site* e publicidade. Exemplos deste tipo de negócio são por exemplo o *site* BedAndBreakfast [36] ou ainda o Yahoo! Small Business [37] e o BingShop [38];
- Vendas Flash. Em nenhum dos modelos acima referidos os vendedores conseguem oferecer grandes descontos através de um intermediário ou diretamente aos consumidores. Este tipo de modelo considera o uso de promoções num curto período de tempo que são praticadas para atrair pessoas que já estão numa loja, ou para fornecedores anunciarem uma venda para um dia ou por vários dias, ou ainda para vendas "doorbuster" entre certas horas num determinado dia. A Woot.com [39], uma empresa da Amazon.com, é um exemplo deste tipo de modelo relacionado com os negócios da própria empresa.

2.1.7 Desenvolvimento de um sistema E-Commerce

Para o desenvolvimento de um sistema EC deve-se ter em atenção um conjunto de opções técnicas que vão desde a arquitetura do sistema, à decisão das tecnologias de Internet a usar e à integração de possíveis sistemas. Mas é importante não esquecer os requisitos básicos deste tipo de sistemas.

Tabela 3 – Funcionalidades esperadas num sistema de EC [40]

Compradores	**Vendedores**
Catálogos com listagem de produtos	Proporcionar o acesso a um catálogo de produtos.
Adicionar produtos ao carrinho de compras	Fornecer o conceito de carrinho de compras eletrónico, no qual os clientes podem adicionar os produtos.
Pagar pelos produtos encomendados, através de alguma forma de crédito.	Verificar o crédito de um cliente e aprovar a respetiva.
Confirmar uma ordem, assegurando que o produto desejado está disponível	Providenciar a entrega do produto.
Controlar os pedidos efetuados	Fornecer meios para os clientes e visitantes se registarem no *site*, fazerem comentários ou para pedirem informações adicionais.
	Fornecer meios para *call center* e resolução de problemas via Internet.
	Analisar as compras do cliente a fim de personalizar as experiências do mesmo.
	Fornecer suporte pós-venda via Internet.
	Fornecer tradução, se necessário.

Um sistema de EC pode ser de vários tipos, como por exemplo prestador de serviços, fabricante/distribuidor, meios de comunicação, viagens/entretenimento ou ainda redirecionado para o setor do retalho, objeto de estudo desta dissertação. Mas, qualquer que seja o tipo em questão existem funcionalidades que são semelhantes a este tipo de sistemas e que necessitam de ser implementadas. Embora isso simplifique a tarefa de criar a arquitetura da aplicação, os requisitos devem ser sempre analisados com atenção, por forma a corresponderem às reais necessidades da empresa. A Tabela 3 apresenta um conjunto de funcionalidades esperadas, quer por parte de compradores, quer por parte de vendedores [40].

Para Turban et al. [40] o ciclo de desenvolvimento de um típico sistema EC e em geral do desenvolvimento de *software*, passa por cinco fases: Identificar, justificar e planear; Criar uma arquitetura; Selecionar uma opção de desenvolvimento; Instalar, testar, integrar e fazer *deploy* da aplicação.

Na Figura 5 é possível a relação entre cada uma dessas fases, sendo que será apresentada uma breve explicação de cada uma delas:

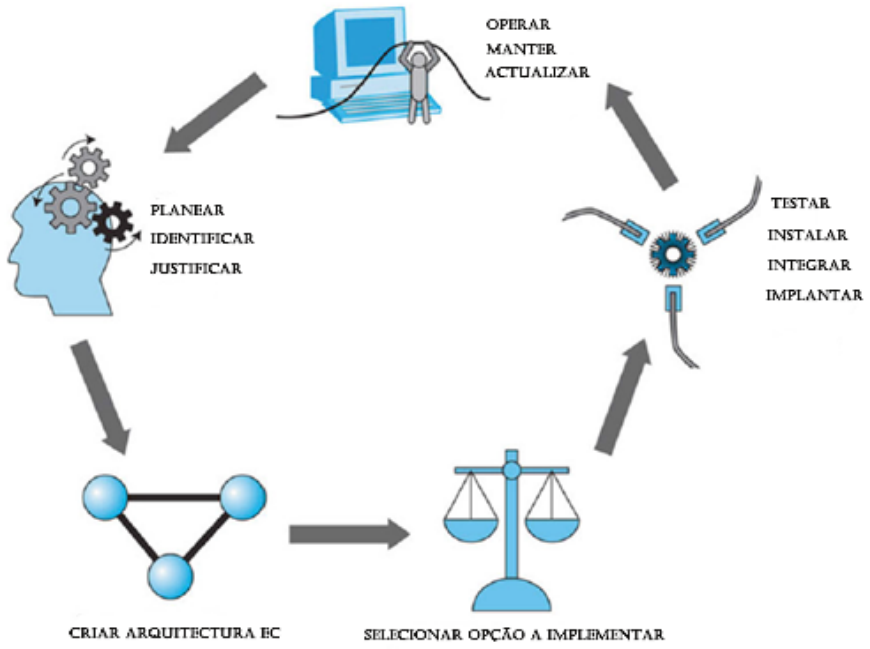

Figura 5 – Ciclo de vida do desenvolvimento de *software* (Adaptado de Turban et al. [40])

- **Identificar, justificar e planear -** Nesta fase, é necessário identificar os processos de negócio associados ao sistema e alinhá-lo com o plano de negócios da organização. Todos os requisitos esperados deverão ser levantados (ter em atenção as funcionalidades básicas referidas anteriormente), assim como o propósito de cada um deles. O resultado dessa etapa resume-se a um conjunto de decisões tomadas para implementação de uma determinada aplicação, com um calendário, orçamento e responsabilidades atribuídas;
- **Criar uma arquitetura -** Com o evoluir das necessidades as empresas começaram a disponibilizar o acesso às suas bases de dados através da Internet e, os métodos usados para interligar os *browsers*, servidores de páginas Internet, aplicações e bases de dados resultaram numa arquitetura *multi-tier*, hoje amplamente conhecida [41].

  Uma arquitetura *multi-tier* é, tal como o nome indica, uma arquitetura que divide o sistema em múltiplas camadas lógicas (camada de *front-end*, camada intermédia e camada de *back-end*), e

que podem ser desenvolvidas e mantidas de forma independente. A Figura 6 apresenta cada uma das camadas em questão e a forma como estas comunicam entre elas, de modo a enviar e a receber dados;

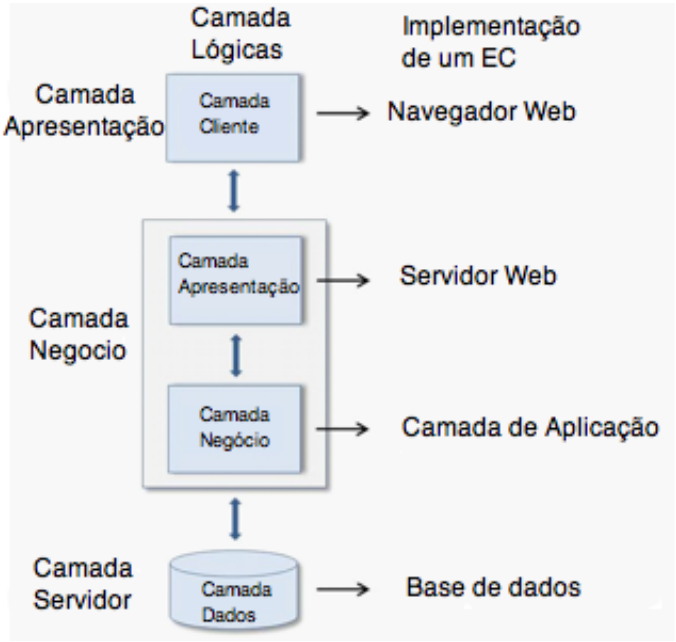

Figura 6 – Arquitetura *multi-tier* [41]

- **Camada cliente/*fron-end* -** A camada cliente exibe o conteúdo de páginas na Internet. Requer que o dispositivo tenha instalado um *browser* (Internet Explorer, Google Chrome, Mozzila Firefox e outros) em execução no computador do utilizador e comunica com a camada de apresentação da camada intermédia;

- **Camada de acesso a dados/*back-end*** - A camada de acesso a dados processa o pedido do utilizador e é responsável por gerir os dados envolventes. O Structured Query Language (SQL) é a linguagem de bases de dados padrão para a comunicação entre a camada de lógica de negócio e a camada de dados. Tecnologias de bases de dados para esta camada incluem o Microsoft SQL Server [49], o MySQL [50], o Oracle Database [51], o PostgreSQL [52] entre outras.

- **Camada intermédia** - A camada intermédia subdivide-se em duas partes: de apresentação e de lógica de negócio. De uma forma genérica a camada de apresentação permite a estruturação da User Interface (UI) da página na Internet e das interações entre o utilizador e a aplicação. É suportada por um servidor *web* (Microsoft IIS [42], Apache Web Server [43], JBOSS [44],

WebSphere [45] entres outros), que permite gerar as páginas na Internet através de pedidos de Hypertext Transfer Protocol (HTTP) e que comunica com a camada de lógica de negócio para realizar os pedidos do utilizador.

Por outro lado, a camada de lógica de negócios é suportada por um servidor de aplicação (por exemplo, Microsoft Windows Server [46], Oracle WebLogic Application Server [47] e IBM WebSphere Application Server [48]) e manipula a lógica de negócios necessária para processar os pedidos do utilizador, ou seja, é nesta camada que são processadas as encomendas, registados os utilizadores, entre outras normais capacidades de uma loja *online*. O servidor de aplicações geralmente automatiza uma série de serviços, tais como transações, segurança, persistência e serviços de mensagens;

- **Selecionar uma opção de desenvolvimento -** A escolha da estratégia de desenvolvimento de um sistema EC depende das necessidades da empresa e faz parte do ciclo de desenvolvimento de um sistema deste género. Como tal, será sempre importante para a escolha da estratégia de desenvolvimento ter em atenção determinados fatores. Clientes alvo, necessidades dos mesmos, produtos a serem vendidos, pagamento *online* ou não, tipos de pagamento, formas de entrega de produtos, promoção dos produtos, dados de marketing e análise feita pelo site e marca do próprio site ou como este se diferencia de outros são apenas algumas questão que devem ser equacionadas e que pesam na decisão final. Independentemente da complexidade da página na Internet, estão disponíveis três estratégias principais [53];
- **Desenvolvimento do sistema *in-house* -** O desenvolvimento *in-house*, ou seja, o desenvolvimento interno numa empresa subdivide-se ainda em mais opções: Construção a partir do zero, sendo esta raramente usada mas que, apesar de ser um processo lento e dispendioso é o que ainda fornece melhor ajuste às necessidades da organização; Construção a partir de componentes/software/serviços já desenvolvidos por outras entidades como ponte de começo de desenvolvimento. Estes componentes englobam servidores de páginas de Internet (Apache e Microsoft IIS) e linguagens de script para páginas na Internet como o PHP, o Microsoft Active Server Pages (ASP), o JavaServer Pages (JSP) entre outras, que permitem uma integração mais acessível com bases de dados e outros sistemas de *backoffices*; Uso de Enterprise Application Integration (EAI) é especialmente interessante quando diversos sistemas necessitam de ser integrados.

Este tipo de estratégia, apesar de proporcionar o desenvolvimento de soluções à medida, peca por uma possível falta de capacidade de reutilização de código e pela falta de interoperabilidade, ou seja, a capacidade de se ligar a pessoas, dados e sistemas;

- **Delegar o desenvolvimento do sistema a terceiros (*outsourcing*) -** O recurso a *outsourcing* é cada vez mais comum entre empresas. No caso do desenvolvimento de sistemas EC o uso de *outsourcing* pode ser feito através do recurso a *software houses* ou empresas de telecomunicações. Por exemplo, empresas como a IBM [54], a Oracle [55], a MCI [56] já fornecem serviços para o desenvolvimento, operação e manutenção deste tipo de sistemas. Este tipo de estratégia permite o acesso a sistemas deste tipo sem grande investimento, mas, por outro lado, não garante que o resultado final seja o expectável;
- **Comprar diretamente um sistema de EC para um determinado tipo de negócio -** A opção de adquirir um sistema de EC pronto a instalar e a utilizar requer sempre menos tempo e investimento do que a opção de desenvolvimento *in-house*, no entanto, não consegue atingir o nível de flexibilidade da anterior. Por outro lado, atualmente existe uma grande oferta no mercado deste tipo de soluções o que possibilita também uma maior gama de escolha. Outra desvantagem deste tipo de estratégia é o facto do produto adquirido deixar de existir/receber atualizações ou tornar-se obsoleto por não conseguir acompanhar os avanços constantes deste tipo de mercado.

O resultado desta etapa é, além da estratégia de desenvolvimento escolhida, a aplicação deve estar desenvolvida e pronta a ser instalada;

- **Instalar, testar, integrar e fazer *deploy* da aplicação -** Uma vez que o sistema já foi desenvolvido, o próximo passo envolve a instalação da aplicação no ambiente de *hardware* e de rede selecionada. Depois de instalada a aplicação torna-se necessário testar o melhor possível cada um dos módulos, através de testes individuais a cada módulo, testes de integração e, teste de aceitação e, por fim, testes de usabilidade. O resultado desta etapa é a disponibilização da aplicação, depois de previamente testada, aos utilizadores finais;

- **Operações, manutenção e atualizações -** Esta é a fase que requer mais tempo, esforço e investimento para a organização e diz respeito a operações de manutenção e atualização essenciais para manter o sistema atualizado e de acordo com o expectável para o utilizador final.

2.2 Arquitetura Model-Driven-Architecture (MDA)

O MDA ou Architecture-Driven Modernization (ADM) [57] como foi originalmente conhecido é um modelo de arquitetura de *software* que teve origem por volta de 2004 com o objetivo de trazer linhas de orientação para especificação de estruturação, sendo que esta é traduzida sob forma de modelos. Este tipo de arquitetura é conhecida como a área da engenharia vocacionada para o domínio do negócio, sendo que o seu objetivo chave consiste na reutilização do *software*. Tendo em conta que as organizações desenvolvem tipicamente sistemas parecidos com diferentes necessidades do cliente, apesar de o *software* ter que ser construído de raiz, existem múltiplos componentes que podem ser reutilizadas de sistemas anteriormente desenvolvidos e que podem ser reutilizados nos novos sistemas.

O MDA é focado na reutilização de código através de padronizações de comportamentos, sendo que a sua implementação não tem em conta a linguagem de programação nem o domínio ao qual é aplicado. A Figura 7 representa a imagem definida pelo MDA para representar a sua diversidade a nível de domínios de atuação. O MDA em si não é uma nova especificação da OMG, mas uma abordagem para o desenvolvimento de *software* que é suportado por outras especificações da OMG já existentes, tais como o Unified Modeling Language (UML), o Meta Object Facility (MOF) e o Common Warehouse Metamodel (CWM) [57].

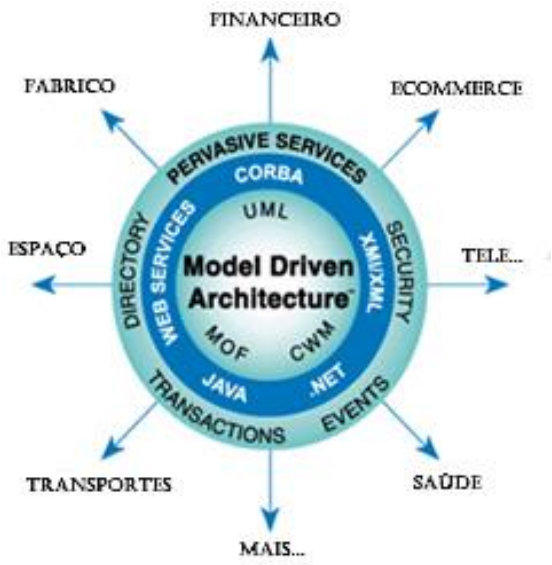

Figura 7 – Áreas de domínio do MDA (Adaptado de [57])

2.2.1 Visão

O mito de aplicações independentes, que nunca necessitam de atualização e correção, construída através de modelos de dados muito proprietários e que nunca eram usados para se integrar com outras aplicações, foi por muito tempo uma realidade, vindo apenas a desvanecer no final do Século XX.

É do conhecimento claro para os programadores que as aplicações deveriam ser feitas para durar, terem a possibilidade de se integrar com outras e de serem atualizáveis. Maioria destas características eram ignoradas e utilizadas apenas as especificações de primeira necessidade para a aplicação desempenhar o papel requerido. As aplicações eram apenas desenvolvidas para durar um determinado número de anos sem nunca se pensar em pormenores de infraestrutura e a constante mudança de requisitos.

Não tardou muito até que novos desafios trouxessem uma mudança na indústria das Tecnologias de Informação e Comunicação (TIC). Com as novas linguagens para sistemas de modelação a surgir, a fundir-se e a convergir com o aparecimento dos primórdios do Simple Object Access Protocol (SOAP), da Object Model Template (OMT), da Integrated Definition (IDEF), entre outras, passou assim a existir uma moda para as linguagens de modulação.

Na realidade, existem poucos motivos pelo qual não se deve fazer primeiro uma análise e um desenho cuidado do *software* antes de o desenvolver. A conceção, não só conduz a sistemas que são mais fáceis de desenvolver, integrar e manter, mas também porque têm a capacidade para automatizar pelo menos algumas partes da sua construção. No mundo do *software* é possível padronizar comportamentos através de modelos definidos e as suas normas, com o uso de Unified Modeling Language (UML), ou MetaObject Facility (MOF) e Common Warehouse Metamodel (CWM) introduzidos todos pela OMG. Através do mesmo mecanismo é possível até gerar ligações quando é necessário conectar com outras aplicações, tendo logo gerado todas as traduções e pontes. Este mesmo princípio pode ser repetido várias vezes, até quando existem novos processos ou novas variáveis a serem consideradas, volta a gerar o conteúdo tendo em conta essas mesmas alterações.

Este foi o desafio que o MDA se propôs, trazendo algo que permitisse ser interpretado pela máquina e que ao mesmo tempo fosse traduzido em modelos de informação de forma a manter uma flexibilidade a longo termo nas seguintes atuações [58]:

- **Implementação** - Permitir que novas infraestruturas e tecnologias recentes possam ser integradas com os desenhos existentes;
- **Integração** - Tendo em conta não só a execução, mas também o projeto estão disponíveis em tempo de integração e é possível automatizar a produção de pontes de integração de dados e conceção com novas infraestruturas;
- **Manutenção** - A disponibilidade do projeto num formato legível da máquina dá aos programadores acesso direto à especificação do sistema, tornando a manutenção mais simples;
- **Testes e Simulação** - Uma vez que os modelos desenvolvidos podem ser utilizados para gerar código, que pode igualmente ser confrontado e comparado com os requisitos, este pode ser testado em diversas infraestruturas e simular diretamente o comportamento do sistema a ser concebido.

2.2.2 Conceitos e definições

Apresenta-se de seguida um conjunto de conceitos importantes para perceber todo o funcionamento de um sistema baseado numa arquitetura MDA [58], [57]:

- **Sistemas -** O contexto do MDA é o desenvolvimento de sistemas de *software*, sendo que estes podem estar em desenvolvimento ou já em fase de término;

- **Model -** Modelo é nome formal que é atribuído a uma especialização de uma função, estrutura ou comportamento num contexto de um sistema através de um respetivo *viewpoint,* conceito este que será referido mais a frente. Um modelo é tipicamente representado por um conjunto de combinações de texto e/ou desenhos, usando para isso notação como o UML ou algum tipo de notação bem conhecida;

- **Model Driven -** Descreve um caso de estudo de desenvolvimento de *software* onde os modelos são usados como fonte principal para documentação, análise, desenho, construção, desenvolvimento e manutenção do sistema;

- **Arquitetura -** A arquitetura do sistema é a especificação das peças e conetores do sistema e as regras para a integração das partes usando os conetores. Dentro do contexto MDA estas partes, conetores e regras são expressas através de um conjunto de modelos inter-relacionados;

- **Viewpoints** - Um *viewpoint* é uma técnica de abstração que nos permite focar num determinado conjunto de particularidades de um sistema, enquanto descartamos todos os aspetos irrelevante. Um *viewpoint* pode ser representado por um ou vários modelos.

  O MDA especifica por omissão três tipos de *viewpoints* num sistema: *Computation Independent Viewpoint*, *Platform Independent Viewpoint* e *Platform Specific Viewpoint*. De seguida serão detalhados os respetivos *viewpoints* de forma entender qual a estrutura que o MDA pretende passar, como a forma mais correta numa arquitetura orientada a modelos:

  - **Computation Independent Viewpoint** - O *computation independent viewpoint foca-se no ambiente do Sistema, e nos requisitos necessários para o Sistema;* os detalhes da estruturação do sistema e o seu processamento permanecem nesta fase ainda escondidos ou por determinar;

  - **Plataform Independent Viewpoint** - A *platform independent viewpoint foca-se nas operações do Sistema enquanto escondem os detalhes necessários para uma plataforma em particular*. A platform independent view mostra que é possível uma parte completa de uma especificação não mudar entre várias plataformas particulares. Esta pode usar uma linguagem de modelação ou uma linguagem de programação específica para descrever como o sistema vai ser usado;

  - **Plataform Specific Viewpoint** - A *platform specific viewpoint combina a* platform independent viewpoint com focus em informação adicional a ser usada numa plataforma específica por um sistema.

  O MDA especifica ainda por omissão a utilização de três modelos no sistema correspondentes aos três *viewpoints* acima referidos. Estes modelos podem ser vistos como camadas de abstração, tendo em conta que, para cada um deles, podem ser definidos vários modelos, cada correspondendo a um *viewpoint* mais focado de um sistema:

- **Plataform Independent Model -** O Plataform Independent Model (PIM) é uma vista do sistema de um Plataform Independent Viewpoint. Uma das formas mais comuns de alcançar é utilizar um modelo do sistema o mais neutro possível, utilizando para isso, tecnologias suportadas por máquinas virtuais. Uma máquina virtual define um conjunto de componentes e serviços que são

especificados independentemente da plataforma de forma a poder ser executada de forma específica mesmo em plataformas diferentes;

- **Plataform Specific Model** - O Platform Specific Model (PSM) representa uma vista do sistema de um Plataform Specific Viewpoint. O PSM combina as especificações provenientes do PIM que lhe proporciona a informação necessária de como articular a transformação para uma determinada plataforma. São criados em função dos objetivos que temos para um determinado sistema, visto que podem assumir qualquer linguagem de programação, linguagem de *script* e todo e qualquer tipo de ficheiro de qualquer sistema operativo;

- **Computation Independent Model -** O Computation Independent Model (CIM) é uma vista do sistema de um Computation Independent Viewpoint. O CIM é normalmente chamado de modelo de domínio e o vocabulário usado é de fácil interpretação de forma a ser de fácil compreensão para pessoas e para coerência com as especificações do sistema;

  Na utilização e criação dos CIM supõe-se que o responsável principal seja uma pessoa que desconheça a forma que os modelos e artefactos vão ser usados, de forma a cumprir a especificidade para um requisito ao qual se enquadra as entradas declaradas no CIM;

  O CIM desempenha um papel de ponte entre as pessoas que são responsáveis e especialistas por analisar o domínio do negócio em questão e entre aqueles que efetivamente constroem os artefactos necessários para cumprir os requisitos desse mesmo domínio.

- **Plataform Model -** O Plataform Model é algo que contém toda a informação técnica que representa as diferentes partes que vão ser utilizadas pela plataforma para representar outra plataforma ou serviço. Também disponibiliza a informação necessária para ser usada para os PSM e ainda os diferentes tipos de elementos que vão ser usados para especificar o uso da plataforma pela aplicação.

  Um tipo de modelo de plataforma genérico pode ser usado para especificar um tipo de estilo específico de arquitetura;

- **Model Transformation** - O Model Transformation [59] é o processo de converter um modelo para outro modelo do mesmo sistema. A transformação combina o PIM com a informação adicional

para produzir o PSM. Após a geração do PSM este irá sofrer outra transformação de forma a poder originar código.

A Figura 8 representa de forma sugestiva a transformação de um modelo que se encontra independente da plataforma, PIM, juntamente com a informação definida para a transformação mais o processamento da ferramenta de transformação, PSM, irá dar origem a um modelo, código, com valor.

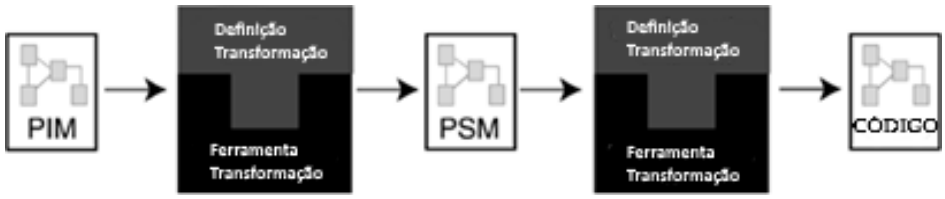

Figura 8 – Modelo de Transformação (Adaptado de [59])

Estas transformações são baseadas em regras bem definidas que fazem com que seja possível transformar um modelo em outro modelo. A característica mais importante nas transformações é que estas ao transformar um modelo de origem num modelo de destino nunca percam o significado, sendo que isto só é aplicável quando existe a possibilidade de representar o significado de igual forma nos dois modelos.

2.2.3 Estratégias de desenvolvimento de sistemas MDA

Independentemente da plataforma destino a que o sistema de MDA pretenda respeitar o primeiro passo será sempre o de representação do CIM através de uma linguagem bem conhecida de estruturação/modelação. Este modelo, após sofrer uma transformação no sistema, pode voltar a ser transformado para uma plataforma distinta conforme a implementação pretendida.

Figura 9 – Processo de transformação do MDA (Adaptado de [59])

Uma simples transformação é definida por três componentes representados na Figura 9, em que um modelo de origem através de um conjunto de regras de transformação bem definidas dá origem a outro modelo destino. De seguida será apresentada uma simples transformação de modelos [59].

Imaginemos um PIM que define um Cliente e os seus atributos **título**, **nome**, **data nascimento**, tipicamente os atributos num PIM são definidos como públicos e ao longo do tempo as suas propriedades podem mudar consoante a necessidade. A Figura 10 representa o PIM descrito.

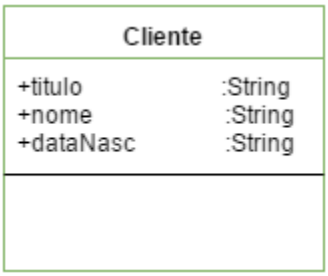

Figura 10 – Cliente PIM (Adaptado de [59])

Neste momento, temos a representação abstrata da plataforma que define um Cliente e as suas características. Caso o modelo de destino fosse por exemplo uma classe Java, os atributos definidos no PIM como públicos seriam vistos como uma má implementação em Java. No Java, os atributos de uma classe devem seguir normas de encapsulamento, utilizando para isso os atributos privados e modificadores de acesso para mudar os seus valores. Tendo em conta esse facto o resultado esperado seria algo com o representado na Figura 11.

Figura 11 – PSM Cliente (Adaptado de [59])

Tanto o PIM com o PSM representados são bastante parecidos, mas são ambos úteis pois contêm informação direcionada para diferentes tipos de programadores e são desenvolvidos em momentos diferentes do ciclo de desenvolvimento de *software*. Na transformação ocorrida neste exemplo, e como é espectável, é possível notar quais as regras que foram aplicadas na transformação. Neste exemplo, é

visível que o nome da classe no PIM é igual ao nome da classe no PSM, assim como os atributos têm o mesmo nome em ambos os modelos. Por fim, é possível ainda notar que para ir de encontro às boas práticas no Java os modificadores dos atributos foram alterados para privados, assim como foram gerados métodos para obter e alterar os valores mantendo a típica notação **get** e **set** evidenciada normalmente no Java.

Um sistema complexo pode consistir em vários modelos interrelacionados e organizados por várias camadas bem definidas onde os modelos são mapeados para outros modelos através de regras bem definidas. Deste conjunto global de modelos podem ocorrer várias transformações horizontais, dentro de uma camada de abstração em conjunto com todas as transformações que ocorrem nas camadas verticais. Aplicar uma arquitetura consistente pelos *viewpoints* do sistema é uma forma de ilustrar essas mesmas transformações horizontais. Por vezes um PSM pode também ser uma camada de abstração, assumindo assim o papel de um PIM, de modo que a informação seja utilizada para futuras transformações e para outras camadas. Para além da simples noção de CIM/PIM/PSM, existem dois conceitos no MDA que são os de modelos e transformações. Este mesmo padrão pode ser repetido e aplicado sucessivamente a modelos, assumindo papéis de PIM e PSM até obter um resultado com valor. Se expandirmos o exemplo anterior adicionando mais classes e associações entre elas começamos a ter perceção da complexidade a ir aumentando gradualmente e a utilidade do sistema por MDA a seguir o mesmo caminho. Adicionando mais duas classes Encomenda e Produto e as relacionarmos com a classe existente anteriormente Cliente teremos o PIM resultante será idêntico ao da Figura 12.

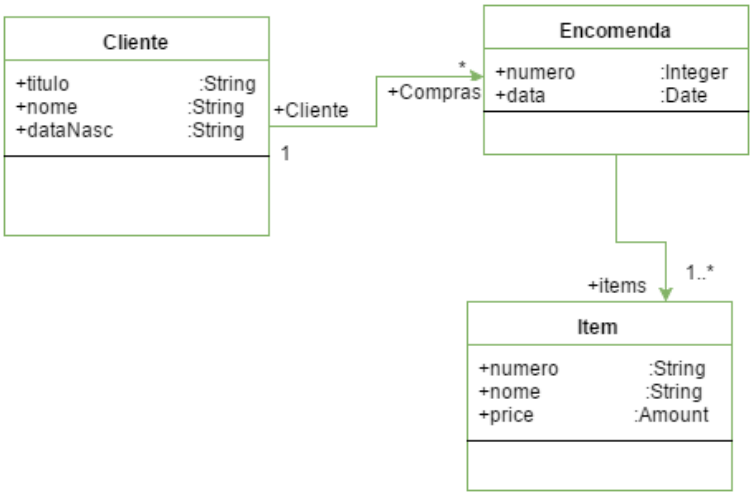

Figura 12 – PIM de relação entre entidades (Adaptado de [59])

Para adaptar as regras anteriormente definidas na situação atual, de forma a poder adicionar o comportamento para tratar as relações na linguagem de programação Java, foram adicionadas regras. Estas são representadas por relações de carnalidade 1-1, em que uma das classes passa a ter um objeto do outro tipo de classe da relação, e por relações de cardinalidade 1-n, onde a classe define um agregador para armazenar objetos da outra classe. Os modificadores para estes novos objetos também serão gerados automaticamente, sendo que o resultado desta transformação seria algo como a Figura 13.

No PSM gerado apresentado na Figura 13 já começamos a notar mais diferenças do PIM que no primeiro exemplo. Estas diferenças vão aumentando à medida que a complexidade das relações e quantidade de entidades definidas no PIM vão aumentando.

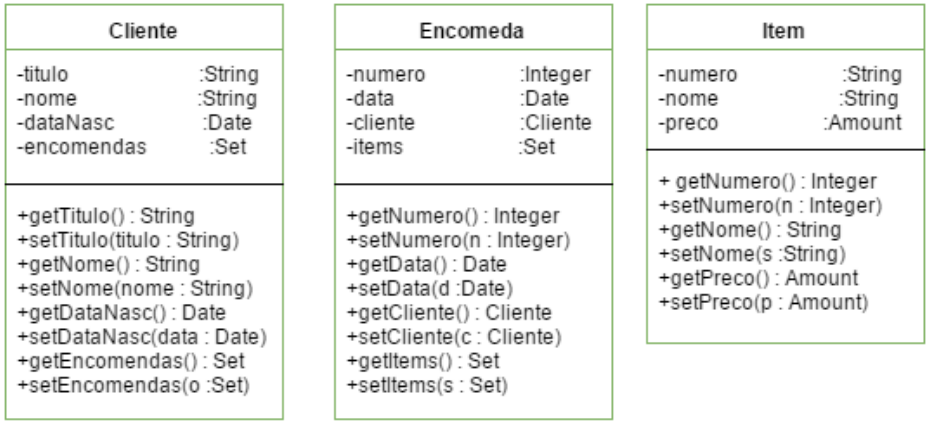

Figura 13 – PSM de relação entre entidades (Adaptado de [59])

2.2.4 Processo de Desenvolvimento por MDA vs Processo Desenvolvimento Tradicional

Esta secção, pretende descrever alguns dos principais problemas do desenvolvimento tradicional de *software* e mostrar algumas das vantagens que um sistema desenvolvido por MDA pode trazer em relação ao modelo tradicional.

Se por um lado a evolução do *hardware* tem vindo a aumentar ao longo dos anos a um nível exponencial e, no que diz respeito a velocidade dos processadores e sua quantidade tem vindo a aumentar, o mesmo não se tem notado ao nível do *software*. O progresso feito no desenvolvimento de *software* não pode ser medido em nível de velocidade e custos. Contudo, algumas melhorias no desenvolvimento têm acontecido. Evidência desse facto é a fiabilidade que existe nos dias que correm na construção de sistemas cada vezes maiores e mais complexos. No entanto, o desenvolvimento de *software* é uma área que ainda

se debate com muitos e grandes problemas. Desenvolver *software* é um trabalho intensivo e, com o aparecimento constante de novas tecnologias, acaba por ser repetido inúmeras vezes. Para além disso, os sistemas nunca são desenvolvidos com apenas uma tecnologia, mas sim utilizado inúmeras, que comunicam dentro de um sistema. Existe ainda o problema dos requisitos estarem constantemente a serem mudados ao longo do desenvolvimento do projeto. Anneke Kleppe e outros [60] colocam quatro grandes problemas nos tradicionais sistemas de desenvolvimento de *software*: a portabilidade, a produtividade, a manutenção/documentação e a interoperabilidade que serão descritos de seguida.

O processo de desenvolvimento de *software* como o conhecemos hoje é muitas vezes impulsionado por um *design* de baixo nível e codificação. Estas abordagens erradas por falta de adoção de adequados processos de desenvolvimento de *software* contribuem para impactos negativos nas empresas, nomeadamente, a falta de qualidade do próprio *software*, custos inicialmente definidos largamente ultrapassados ou ainda desvios dos prazos definidos. A Figura 14 apresenta um esquema das principais fases e tarefas de um sistema típico de desenvolvimento de *software*.

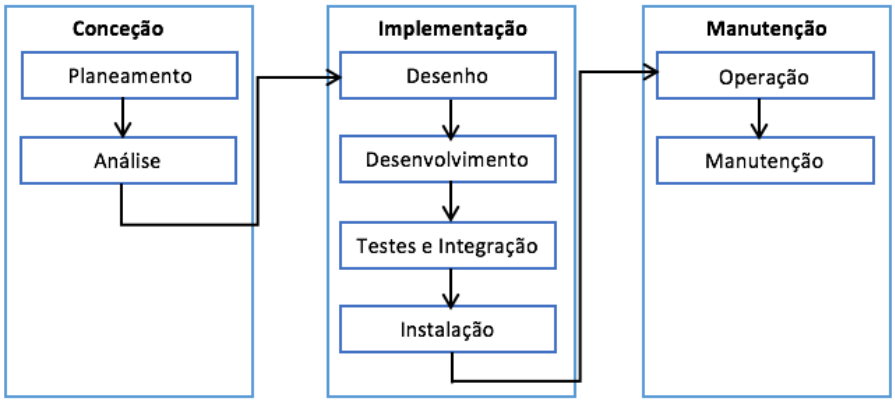

Figura 14 – Fases e tarefas do desenvolvimento de *software (Adaptado de* [61]*)*

As fases, numa visão de alto nível, são constituídas pela conceção, implementação e manutenção, mas é possível analisar ainda mais em pormenor cada uma destas fases subdividindo as mesmas em seis subfases representadas na Figura 14 [60] .

Independentemente do tipo de desenvolvimento adotado seja este incremental ou iterativo e até mesmo em cascata, são produzidos inúmeros documentos durante as três primeiras fases. Os **documentos** e os **diagramas** correspondentes criados nas fases rapidamente perdem o seu valor depois da fase de desenvolvimento de código começar. A ligação entre a **documentação**, diagramas e o código vai-se

perdendo à medida que a fase de desenvolvimento vai evoluindo. Isto leva a que os diagramas, em vez de serem o reflexo do código, sejam uma imagem não relacionada. Isto acontece porque normalmente o código vai mudando inúmeras vezes ao longo do tempo e, na fase de desenvolvimento, o tempo para fazer atualizações na **documentação** nem sempre é uma possibilidade. Para além disso, o valor adicionado para alterar um **diagrama** ou documento é questionável. Qualquer alteração vai ser começada no código de qualquer forma, logo é discutível o tempo que se perde a construir documentação com alto nível de especificação.

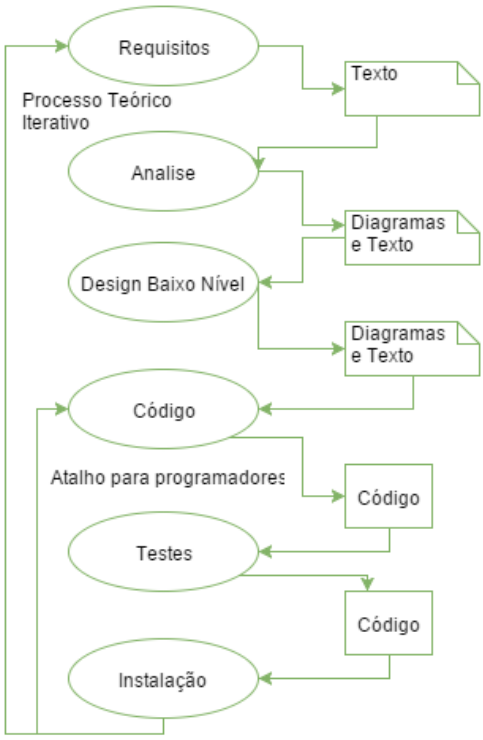

Figura 15 – Tradicional processo de desenvolvimento de *software* (Adaptado de [60])

Nos meados do ano 2000 surgiu a ideia do Extreme Programming (XP) [62], que rapidamente se tornou popular, pela principal razão do código ser a maior força na fase de desenvolvimento. Este tipo de método de desenvolvimento é focado apenas no desenvolvimento e testes sobre o mesmo. Mas esta abordagem resolveu apenas um problema, pois, à medida que as equipas vão desenvolvendo, o *software* vai atingindo um nível de complexidade maior. Caso a equipa tenha que ser desmantelada e seja necessário incorporar novos membros na equipa, será muito difícil entrar no ciclo de desenvolvimento do mesmo e acompanhar o ritmo dos restantes elementos. É necessário deixar marcos traduzidos normalmente por **texto** ou **diagramas** de alto nível para que seja possível a esses novos membros guiarem-se através do fluxo de

código. Este tempo é necessário que seja utilizado na primeira fase do desenvolvimento do *software* ou mais tarde, na fase de manutenção, será necessário perder esse tempo a tentar entender o que o código faz.

A **portabilidade** é outros dos problemas identificados no desenvolvimento do *software* tradicional pois a indústria na qual se insere tem características bastante diferentes de todas as outras. A cada ano que passa e cada vez mais rápidas novas tecnologias vão sendo propostas e ficam mais populares. Este fato tem repercussões, levando as empresas a seguir estas novas tecnologias ou por serem exigidas por clientes, ou para resolver problemas reais evidentes, ou então obrigados pelas ferramentas que deixam de dar suporte a tecnologias antigas. As novas tecnologias certamente oferecem benefícios para as empresas e muitas delas não podem deixar passar estes benefícios. Contudo, isto obriga as pessoas da mesma empresa a terem que se adaptarem a novas tecnologias de uma forma muito rápida. Como consequência disso os investimentos feitos nas tecnologias anteriores perdem valor ou até se torna mesmo inútil. Estas situações tornam-se ainda mais complicadas porque muitas vezes as ferramentas apenas dão suporte nas últimas duas ou três versões finais das tecnologias e, por consequência, ou o *software* é **atualizado** ou então é mudado para outra tecnologia.

Outro grande problema identificado é o da **interoperabilidade**, que é necessário cada vez mais existir no desenvolvimento do *software,* pois a ideia de *software* isolado deixou de ser uma realidade. Nos dias que correm as aplicações necessitam de comunicar com outras aplicações, esta realidade é possível ser analisada nas aplicações de Internet, onde estas necessitam de comunicar com múltiplos serviços para poder disponibilizar a informação necessária ao utilizador final. Nos últimos anos aprendemos a não construir sistemas monolíticos. Em vez de tentar construir componentes que fazem o mesmo, tentamos delegar esse trabalho para outros que já o fazem por nós. Estes componentes são construídos utilizando o melhor da tecnologia para o corrente trabalho, sendo agora necessário criar a forma de interagir entre eles, abrindo assim espaço para **interoperabilidade**.

Conforme o que já foi analisado acima o processo de **documentação** nem sempre é o mais preferencial no processo de desenvolvimento. O desenvolvimento de documentação tipicamente é o processo que os programadores menos gostam de desenvolver. Isto deve-se pelo facto de muitas vezes desenvolverem a documentação, sem entenderem o valor pelo qual esta é requisitada. Estes problemas levam a que a documentação nem sempre tenha a melhor qualidade ou então que se encontre desatualizada. A solução para este problema passa muitas vezes por gerar documentação a partir do código e assegurar que esta

se encontre sempre atualizada. A **documentação** e o código não são entidades separadas e, como tal, a documentação de baixo nível pode ser gerada por ferramentas, contudo, a **documentação** de alto nível continua sempre a precisar de ser mantida manualmente.

Como é possível analisar na Figura 16, o processo de desenvolvimento por MDA não difere muito do modelo tradicional de desenvolvimento de *software*, sendo que as diferenças são baseadas nos artefactos que são gerados através dos modelos ao longo do processo e que são a base do MDA.

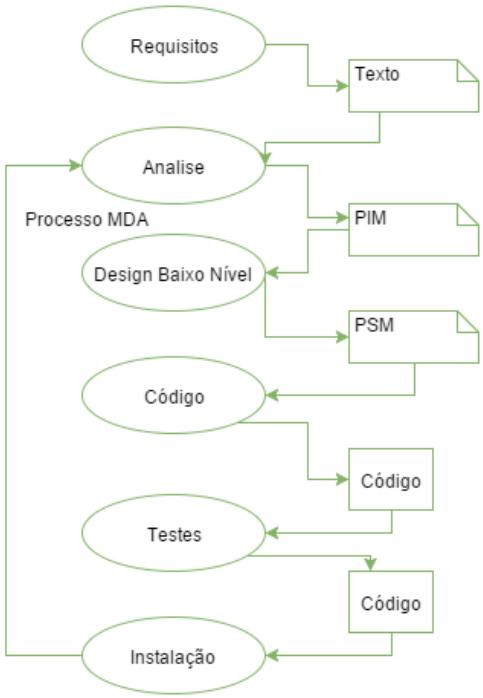

Figura 16 – Processo de desenvolvimento de *software* por MDA (Adaptado de [60])

O nível de abstração conseguido pelos PIM e pelos PSM permitem ao programador criar sistemas mais complexos com menos esforço. Através do MDA que, por processos automatizados por *templates* permitem transformar modelos em código ou modelos em modelos sem que seja necessária grande intervenção manual, não deixando, contudo, de existir.

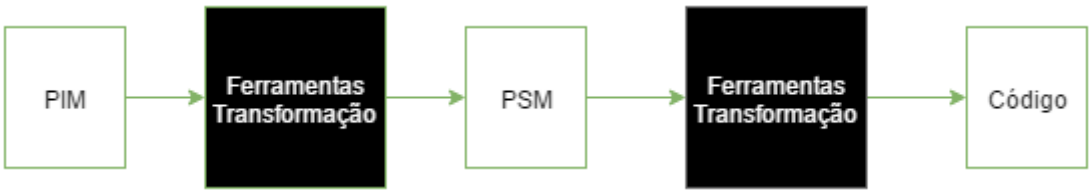

Figura 17 – Transformações no MDA (Adaptado de [60])

Como é possível analisar na Figura 17 as transformações ocorridas no MDA são sempre executadas por ferramentas, estas que por si não trazem nada de novo, pois existem muitas capazes de transformar um PSM em código. Contudo, o benefício do MDA está no tempo que pode ser poupado na análise da documentação do sistema, que por sua vez é traduzido em componentes do mesmo e que, através do MDA, pode ser facilmente automatizado. Apesar das ferramentas que fazem as transformações de PSM para código não estarem 100% automatizadas, precisando o programador de dar sempre o seu *input* nos PSM, este processo permite ainda assim ter *feedback* imediato, pois o resultado será um sistema gerado no momento.

O MDA leva a um aumento de produtividade em duas maneiras. Primeiro porque os programadores perdem menos tempo a desenvolver código de estruturação da informação, visto que este é definido previamente por programadores com alto nível de conhecimento nos PSM. Em segundo lugar o MDA permite que os programadores possam estar mais focados no código dos PIM, estando assim mais preocupados em resolver problemas de negócio. Estas duas medidas trazem benefícios, acabando o sistema por ir mais ao encontro das necessidades do utilizador final e em menos tempo. Contudo, é preciso realçar que este só é alcançável com um esforço prévio no desenvolvimento de um sistema capaz de automatizar estes passos, o que implica um esforço maior do que num desenvolvimento tradicional. Este esforço adicional no início do projeto é depois compensado com a fácil manutenção do sistema e com simples modificações dos ficheiros PIM.

Um dos problemas identificados no típico desenvolvimento de *software* é a **portabilidade** que num sistema de MDA é resolvido pelo facto do sistema assentar a maior parte do esforço no desenvolvimento dos PIM, que por sua vez são transformados automaticamente em PSM para diferentes plataformas, trazendo desta forma **portabilidade** para o sistema. Este cenário aplica-se até para as novas tecnologias, pois estas requerem que sejam reutilizados os PIM pensados anteriormente e sejam realizadas algumas alterações ao nível dos PSM.

Outros dos problemas identificados é o facto de, atualmente, as aplicações não poderem funcionar de forma isolada, existindo cada vez mais a necessidade destas se relacionarem. Essa **interoperabilidade** está subjacente no processo do MDA através das relações que existem entre os PIM, conhecidas também como pontes (Figura 18).

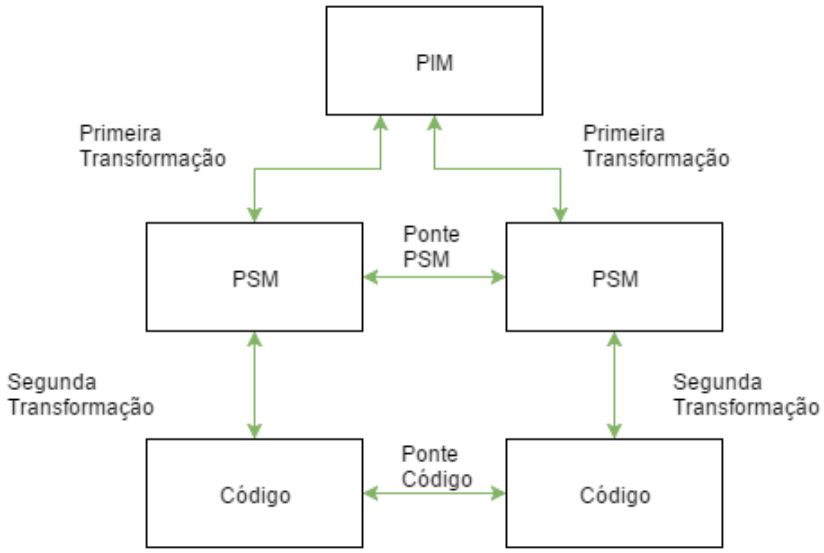

Figura 18 – Exemplos de pontes entre PSM e código (Adaptado de [60])

Essas pontes são criadas, permitindo os PSM comunicarem através delas de forma a poderem trocar informação. Analisando os PSM é possível identificar qual a correspondência de cada componente com os componentes descritos no PIM, sendo esta coerência verificada depois do processo de transformação de vários PSMs, identificando assim facilmente a respetiva correspondência entre os dois e com ela criar a ponte de comunicação entre os PSMs. Estas ferramentas ao gerar os PSM e as pontes entre eles permitem recuperar o investimento feito na criação dos PIM.

Outro problema identificado no ciclo de desenvolvimento tradicional de *software* refere a falta de acompanhamento no desenvolvimento da **documentação** do sistema. Este fator no MDA é atenuado, pois os programadores focam-se no desenvolvimento dos PIM que por sua vez já representam uma abstração de alto nível do código produzido. Podem então usar os PIM para preencher a funcionalidade de **documentação** do próprio sistema de *software*, sendo que estes vão colmatar a falha identificada anteriormente de falta de atualização, pois ao fazer alterações no sistema vai obrigar a fazer alterações no PIM que por sua vez vai voltar a ser transformado em PSM e por fim em código.

2.2.5 Vertentes do MDA

Depois do MDA ter sido publicado pela entidade OMG, outras implementações foram propostas baseadas no MDA, como foi o caso do Model Driven Development (MDD) , o Model Driven Engineering (MDE) e

ainda o Model Based Engineering (MBE) [63]. A Figura 19 representa a ligação entre as diversas implementações referidas.

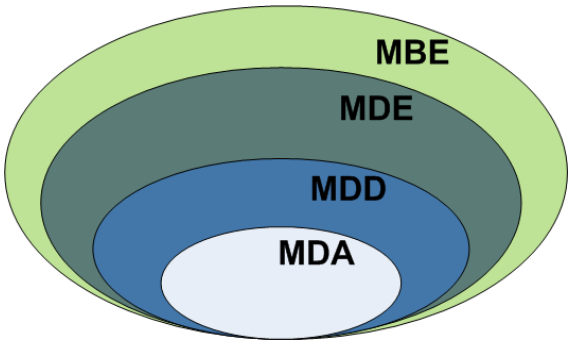

Figura 19 – Derivações do MDA (Adaptado de [60])

MDD é a implementação de modelos gerados parcialmente e automaticamente, sendo que esta é uma visão particular sobre o seu subconjunto MDA, usando os seus padrões e regras no desenvolvimento de uma aplicação.

O MDE é uma visão de mais alto nível baseado no subconjunto MDD que vai mais longe que o simples desenvolvimento de atividades de código puro e engloba outras tarefas baseadas em modelos de um processo de engenharia de *software* completo.

Por fim, o MBE que pretende ser uma versão mais leve do MDE. O MDE é um procedimento onde os processos no qual os modelos de *software* são um passo importante no desenvolvimento do sistema, mas não são necessariamente a chave. Neste caso, os modelos não guiam ("*driven*") o desenvolvimento do processo.

2.2.6 Compromisso do MDA

O MDA facilita a criação de modelos interpretáveis pelo computador com o objetivo de manter uma flexibilidade nos seguintes aspetos [58]:

- **Tecnologia obsoleta -** Novas implementações de infraestruturas podem ser facilmente integradas e suportadas pelo *design* e arquitetura existentes;
- **Portabilidade -** Novas funcionalidades podem ser mais rapidamente migradas para a novos ambientes e plataformas de forma a colmatar as necessidades do negócio;

- **Produtividade e tempo de entrada no mercado -** Ao automatizar muitas das tarefas repetitivas e entediantes, de forma a dar mais tempo livre para os programadores e arquitetos focarem a sua atenção para as tarefas lógicas do sistema;

- **Qualidade -** O MDA proporciona uma separação dos conceitos, consistência e qualidade, e confiabilidade dos artefactos produzidos, que todos juntos contribuem para a qualidade do sistema como um todo;

- **Integração -** A produção e integração de mecanismos de interligação de sistemas externos é bastante facilitada;

- **Manutenção -** A disponibilidade do projeto num formato legível por uma máquina, proporciona aos analistas, programadores e "*testers*" o acesso direto às especificações do sistema, simplificando a manutenção de tarefas;

- **Simulação e teste -** Os modelos podem ser diretamente confrontados e validados contra os requisitos apresentados, assim como testados nas várias infraestruturas selecionadas como destino de *output;*

- **Retorno de investimento -** O negócio consegue extrair o valor do investimento não usado em ferramentas.

2.2.7 MDA para sistemas E-Commerce

Esta secção explora o desenvolvimento de um processo de criação de modelos para EC de forma que se possa obter a reutilização e a interoperabilidade e no qual se propõe obter padrões para processos de EC [64]. Isto é, conseguido com a ajuda dos princípios propostos pelo MDE. O MDA permite minimizar o tempo e esforço necessários para criar soluções de comércio eletrónico.

Existem várias plataformas que oferecem soluções de EC, assim como empresas que desenvolvem os seus próprios *sites* de acordo com suas necessidades específicas. Ao selecionar uma solução de EC, é necessário ter em conta algumas característica que atualmente são essenciais, como é o caso das ferramentas para geração de relatórios, sistemas de recomendação e partilha nas redes sociais, repositórios de ficheiros multimédia, ferramentas de *marketing*, compras, *backoffice*, possibilidade de multi-língua, vários tipos de

pagamento e com vários tipos de moedas disponíveis, segurança de *e-mail,* alertas, entre outros. Cada solução que aborda estes processos desenvolve padrões mínimos que limitam o reuso e a interoperabilidade.

Durante a última década, os sistemas de EC ganharam uma rápida evolução da simples forma de uma página estática na Internet, onde eram fornecidas as informações e produtos promovidos, para sistemas altamente complexos e aplicações dinâmicas que suportam os processos de negócios e transações entre empresas.

Por consequência, o desenvolvimento de EC tornou as aplicações mais complexas. Alguns dos desafios nos processos de negócio e da tecnologia passam pela **exigência de um elevado grau de integração e interoperabilidade** que os sistemas de EC estão radicalmente a apoiar sobe forma de reproduzir uma competitividade entre as empresas. Isto requer a integração de processos de negócios entre empresas. Para conseguir esta integração são exigidos componentes distribuídos que utilizam tecnologias diferentes que precisam manter a sua interoperabilidade.

Outro dos desafios, que as empresas se têm deparado são os **ciclos de desenvolvimento cada vez mais curtos, devido à rápida evolução das tecnologias e à enorme concorrência.** Os sistemas EC normalmente têm um tempo muito curto para serem desenvolvidos e, como tal, tem que ser possível reduzir consideravelmente o ciclo de desenvolvimento para responder às novas exigências do mercada. A **qualidade do** sistema de EC deve ter as funcionalidades necessárias, o sistema também deve atender aos requisitos de qualidade, deve ser confiável e eficiente. Como já foi referido anteriormente a **tecnologia muda** regularmente e como tal são a causa das mudanças nos sistemas de EC. Uma clara separação de lógica de negócios e tecnologia fará os sistemas de EC mais resistentes à mudança. Outro grande desafio que cada vez mais as empresas encontram, é a **falta de pessoal qualificado exigida pela** complexidade dos sistemas de EC, gera grande procura por programadores com capacidades necessárias para os produzir. Essas exigências são difíceis de colmatar devido à escassez de arquitetos de sistemas e programadores experientes na área.

Além disso, a análise de modelos de EC é um processo complexo que envolve a seleção e implementação do modelo mais adequado de modo a facilitar a transformação do negócio e assegurar um desenvolvimento estável o suficiente para sobreviver na competição global. O modelo adequado é a primeira questão importante a considerar e tem um enorme potencial de transformação de negócios, mas não o suficiente se não se centrar na sua melhoria.

Quando os dados são processados e comunicados automaticamente, o tempo de processamento é reduzido. As interações de negócios automatizadas, utilizando as TIC, podem contribuir para menos interações nos negócios de forma manual. Através de interações automáticas é possível evitar potenciais erros, na utilização de *e-mail*, telefonemas ou faxes. Infelizmente, não é fácil automatizar processos de negócios e suas interações, porque os sistemas de informação não são interoperáveis devido às diferenças entre as empresas em cada um dos EC desenvolvidos.

Se os sistemas de informação não são interoperáveis, a intervenção humana é necessária para preparar os dados a serem processados. As diferenças na infraestrutura de tecnologia entre empresas são inevitáveis, levando a custos devido à falta e à dificuldade de criar interoperabilidade. Felizmente, as regras podem fornecer uma maneira de reduzir esses custos, colocando em ordem de complexidade e incerteza, reduzindo a quantidade de diferença entre sistemas. A padronização de documentos e processos de negócios promove assim a interoperabilidade, harmonizando os significados dos termos e modalidades de operações. As empresas podem otimizar seus sistemas de informação, desde que use o mesmo tipo de documentos de negócios, processos de negócios e otimizar suas *interfaces* de mensagens.

Os investimentos no futuro devem ser direcionados para financiar novos padrões de EC. Deve ser criada uma proposta e ser analisada por um conjunto de empresas de forma a tentar cumprir as expectativas de cada uma e manter sustentabilidade no contexto.

O conhecimento acumulado no campo da engenharia de conceção e usabilidade não é suficientemente organizado e estruturado em termos da sua representação para facilitar a construção de aplicações de comércio eletrónico. Este é um dos fatores que dificultam a utilização adequada de ontologias na análise e domínio para desenvolver *software* de EC. A partir da identificação das variáveis críticas, tais como metas de negócios, características e gostos dos utilizadores, é possível inferir que a formalização do conhecimento através de ontologias pode ser incorporada com sucesso no desenvolvimento de tais soluções. São necessárias normas, a fim de desenvolver modelos de negócios bem-sucedidos e, assim, avaliar a estratégia para assegurar a maximização do impacto.

Para isso é necessário uma proposta de solução que permita a geração de uma ontologia da qual seja depois possível proceder a uma modelação dos processos de EC padrão, utilizando o MDE para se conseguir a transformação dos modelos propostos de modo a permitir ir dos mais gerais (Requisitos) para o particular (implantação da solução) através da conversão destes modelos.

Oscar Martínez e outros [64] proposeram um *metamodelo* de componentes envolvidos em subprocessos de integração de um sistema de EC. Na Figura 20 é possível ver esses mesmos componentes representados.

Figura 20 – Proposta de metamodelo para sistemas EC (Adaptado de [64])

Esse mesmo *metamodelo* está representado por doze componentes:

- **Implementação de Site EC -** Resultado da Tecnologia estratégico alinhado com a estratégia de negócios para implantar o site garantindo a capacidade de gestão de conteúdo para EC;
- **Interação com o Cliente com Site EC -** Com a implementação de uma plataforma de utilização apelativa que pode ser conseguida através da interação do cliente;
- **Seleção e Ordem de Compra Produto / Serviço -** O cliente navega o catálogo de produtos ou serviços, seleciono os que me interesso e gera a ordem de compra.
- **O pagamento *online* -** O Site de EC contém um gestor de pagamento eletrónico que facilita o processo de pagamento *online*;

- **Geração de Ordem de Vendas e Registo do cliente -** A ordem é gerada e gerida nos repositórios e módulos de contabilidade e executado um armazenamento de dados de clientes;
- **Geração de Fatura** - Gera a contabilização da compra de apoio (fatura) com inclusão de impostos e descontos;
- **Contabilidade -** A integração de vendas eletrónicas com as aplicações de contabilidade devem ser fornecidas, a fim de atualizar as informações necessárias;
- **Geração de Ordem de Entrega -** A atividade de logística começa com o relatório da venda para garantir a entrega efetiva ao cliente;
- **Gestão de Inventário -** A integração com o módulo de inventário facilita a gestão do mesmo e no processo de gravação unidades disponíveis e ainda no processo de re-ordem;
- **Catálogo atualizado de Produtos e Serviços -** Uma atualização adequada facilita a gestão comercial, contabilidade e inventário da empresa;
- **Canais de Vendas Alternativas -** A empresa pode ter canais alternativos, como lojas físicas, outros distribuidores, etc.;
- **Reabastecimento -** Com base na informação no sistema o pedido de reabastecimento pode ser feito ao cliente automaticamente.

2.2.8 Ferramentas MDA

A arquitetura MDA é bem conhecida e bastante madura, como tal existem inúmeras empresas, organizações, pessoas individuais que contribuíram significativamente com ferramentas de desenvolvimento por modelos orientados pelos artefactos definidos pelo MDA.

Apresentam-se assim de seguida algumas soluções *open source* e outros comerciais de forma a demonstrar algum investimento de tempo e monetários envolvidos nesta temática.

Aplicações Open Source

Existem diversas ferramentas de geração por MDA *open source* espalhadas pela Internet*,* umas de expansão por comunidades outras mais fechadas por empresas que disponibilizam o seu código. Assim, algumas dessas possíveis ferramentas disponíveis são:

- **XDoclet -** XDoclet [65] é uma *framework* gratuita utilizada para geração de código orientado a objetos na linguagem de programação Java, mais propriamente Enterprise Java Beans (EJB)[2]. Tipicamente esta *framework* utiliza modelos de forma a gerar *JavaDoc,* utilizando para isso a geração de XML a partir do código fornecido e depois gerando o código fonte final.
  Este tipo de ferramenta traz múltiplos benefícios dado que o programador não tem necessidade de se preocupar com os ficheiros de *metadata,* que ao desenvolver o código são também atualizados. O programador apenas necessita de desenvolver um ficheiro de cada vez para cada componente, dado que os outros são gerados através da *framework,* de forma a manter o EJB completo. O foco passa a ser redirecionado inteiramente para o desenvolvimento da lógica de negócio, já que o XDoclet gera cerca de 85% do código restante automaticamente.

  Apesar do XDoclet ter sido inicialmente desenvolvido para geração de EJBs, este evoluiu para um propósito mais generalista, permitindo acrescentar módulos de forma a gerar novos artefactos;

- **AndroMDA -** AndroMDA [66] é uma *framework* de MDA *opensource* que leva qualquer número de modelos combinado com qualquer número de *plugins Andromeda* e produz qualquer número de componentes personalizados. Assim, pode-se gerar componentes para qualquer linguagem de programação, como por exemplo Java, .Net, HTML, PHP, entre outras.

  Esta *framework* é usada principalmente por programadores que trabalham com tecnologias J2EE, permitindo criar projetos J2EE a partir do zero, no qual o código é gerado a partir de um modelo UML. O AndroMDA tem como objetivo atenuar um conjunto de problemas, tais como:

  - Elimina a necessidade de escrever código redundante;
  - Os modelos de projeto serão refletidos no código;
  - Os projetos são documentados/expressos por um diagrama de uma forma independente de plataforma padrão tornando muito mais fácil/mais rápido para se adaptar cada vez que muda rapidamente de tecnologias;

[2] Componente da Plataforma JEE que consiste num *container* para vários ficheiros java que são posteriormente num servidor Web garantindo para este, segurança, um ciclo de vida no servidor Web, transações, entre outras características.

Aplicações Comerciais

A par das ferramentas *open source* existem também as ferramentas comerciais, fornecidas por empresas que investiram tempo e recursos para desenvolver *frameworks*/API's baseadas nos conceitos do MDA de forma a poder ser possível gerar código de *software* ou documentação, com base em metadata definida pelo utilizador. De seguida serão apresentados alguns desses produtos disponíveis em mercado:

- **Model Component Compiler -** A InferData desenvolveu uma ferramenta que reduz drasticamente o esforço de desenvolvimento exigido para implementar soluções de *software* o Model Component Compiler [67]. A ferramenta baseia-se nos esforços da OMG na definição do MDA, convertendo modelos declarativos em soluções completas, onde nenhum código explícito está escrito, de modo a ligar a lógica de negócio à *framework* das empresas.

  Atualmente, o Model Component Compiler apenas está desenvolvido para suportar J2EE, mas estão a ser feitos esforços para apoiar também a f*ramework* .NET.

- **Enterprise Architect 12 –** A sparx systems desenvolveu uma ferramenta que funciona tendo como base modelos. Usando padrões bem definidos, boas práticas de desenvolvimento de *software* e metodologias MDA, de forma a permitir o desenvolvimento de arquiteturas tanto para pequenas aplicações como grandes. Esta ferramenta proporciona um ambiente de desenvolvimento visual, como um IDE onde o utilizador pode de forma fácil definir os modelos e transforma-los nas mais diversas linguagens de programação.

2.3 Ferramentas e tecnologias

Neste capítulo pretende-se descrever as ferramentas e tecnologias utilizadas para a criação e elaboração do MGC. Assim sendo serão abordados: o Freemarker [68], como ferramenta auxiliar na geração de código (através do uso de *templates*); a linguagem Java [69] para o desenvolvimento do MGC e para a criação do *webservice* na *framework* Spring; a linguagem C# [70], usada para a criação do *webservice* WCF; o Maven [71], usado para a compilação dos módulos do MGC e para a compilação dos projetos desenvolvidos na linguagem Java (gerados pelo MGC); as *frameworks* Spring [72] e WCF [73] que são utilizadas para o desenvolvimento de *webservices*; o formato de texto XML e o validador XSD [74] que são usados para criar e validar os ficheiros CIM (respetivamente), utilizados no MGC.

2.3.1 Freemarker

O Freemarker [68] é numa ferramenta de manipulação e interpretação de *templates*, desenvolvida para ser usada na linguagem de programação Java. Os *templates* são desenvolvidos num formato próprio conhecido por FreeMarker Template Language (FTL), que tem uma linguagem de programação própria, especializada para lidar com *templates*. É utilizada apenas para definir como a informação vai ser apresentada, depois de uma outra linguagem de programação tratar os dados e os redirecionar para ela. No *template,* é focada a forma de como apresentar a informação, enquanto fora dele o foco redireciona-se para a informação a apresentar (Figura 21).

Esta abordagem é normalmente associada ao Model View Controller (MVC), tipicamente utilizada para dividir a lógica da apresentação. Assim, é possível mudar a forma como a informação é apresentada sem ter que recompilar o código. O Freemarker é uma linguagem poderosa que permite tratamento de condições, iterações, operações aritméticas, funções e outros [68].

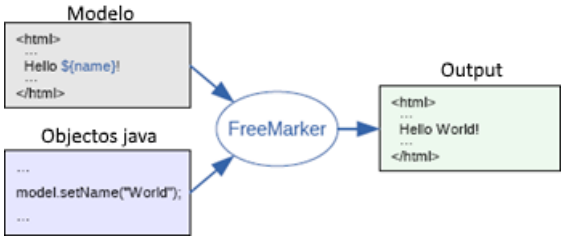

Figura 21 – Geração de *output* através de *templates*

2.3.2 Java

A linguagem de programação Java [69] é a base para praticamente todo o tipo de aplicação em rede e é o padrão global para desenvolvimento e fornecimento de aplicações incorporadas e móveis, jogos, conteúdos baseados na Internet e *software* da empresa. Com mais de 9 milhões de programadores em todo o mundo, o Java permite desenvolver de forma eficiente, implementar e utilizar aplicações e serviços.

O Java foi testado, refinado, ampliado e experimentado por uma comunidade dedicada de programadores, arquitetos de sistemas entusiastas. O Java é projetado para permitir o desenvolvimento de aplicações de alto desempenho, portáteis, para a mais ampla gama de plataformas. Ao disponibilizar aplicações em ambientes heterogéneos, as empresas podem oferecer mais serviços e aumentar a produtividade do utilizador final, nas mais diversas perspetivas como a

comunicação e colaboração, ainda ajudando a reduzir drasticamente o custo de propriedade de ambas as aplicações empresariais e de consumo. O Java tornou-se inestimável para programadores, permitindo-lhes:

- Desenvolver *software* numa plataforma e executá-lo em praticamente qualquer outra plataforma;
- Criar programas que podem ser executados dentro de um navegador de páginas de Internet e aceder a serviços nelas disponíveis;
- Desenvolver aplicações do lado do servidor para fóruns *online*, lojas, etiquetas, processamento de formulários HTML, e muito mais. Combinar aplicações ou serviços que usam a linguagem Java para criar aplicações ou serviços altamente personalizados;
- Escrever aplicações robustas e eficazes para telemóveis, processadores remotos, microcontroladores, módulos sem fio, sensores, *gateways* e praticamente todos os dispositivos eletrónicos.

2.3.3 Linguagem C#

C# [70] é uma linguagem elegante e segura orientada a objetos que permite aos programadores criar uma variedade de aplicações seguras e robustas que são executados na *framework* .NET. O C# pode ser usado para criar aplicações cliente para Windows, serviços em páginas de Internet em XML, componentes distribuídos, aplicações cliente-servidor, aplicações de base de dados, e muito mais. O Visual C# fornece um editor avançado de código, o design de interface de utilizador conveniente, o modo de *debug* integrado, e muitas outras ferramentas para tornar mais fácil de desenvolver aplicações baseadas na linguagem C# e *framework* .NET.

A sintaxe do C# é altamente expressiva, mas é simples e fácil de aprender. Por blocos entre chavetas do C# será instantaneamente reconhecível para qualquer pessoa familiarizada com as linguagens de programação C, C ++ ou Java. Os programadores que sabem programar qualquer uma dessas linguagens de programação são tipicamente capazes de começar a trabalhar de forma produtiva em C# dentro de um tempo muito curto. A sintaxe do C# simplifica muitas das complexidades do C++ e fornece recursos poderosos, como enumerações, expressões lambda e acesso direto à memória, que não são encontrados em Java. C# suporta métodos e tipos genéricos, que proporcionam maior segurança de tipo e

desempenho, e iteradores, que permitem implementadores de classes de coleção para definir comportamentos de iteração personalizados que são simples de usar por código do cliente.

Como uma linguagem orientada a objetos, o C# suporta os conceitos de encapsulamento, herança e polimorfismo. Todas as variáveis e métodos, incluindo o método principal, ponto de entrada da aplicação, são encapsulados dentro das definições de classe. Uma classe pode herdar diretamente de uma classe base, mas pode implementar qualquer número de interfaces. Métodos que substituem os métodos virtuais numa classe base exigem a palavra-chave *override* como uma maneira de evitar a redefinição acidental. Em C#, uma estrutura é como uma classe de peso leve; é um tipo alocado-*stack* que podem implementar interfaces mas não suporta herança.

Além desses princípios básicos de orientação a objetos, o C# facilita o desenvolvimento de componentes de *software* através de várias construções de linguagem inovadoras, incluindo o seguinte:

- Assinaturas de método encapsulado chamados delegados, que permitem que as notificações de eventos de tipo seguro;
- Propriedades, que permitem o acesso a variáveis privadas.
- Atributos, que fornecem *metadados* declarativos sobre tipos em tempo de execução;
- Comentários de documentação XML *inline*;
- Language-Integrated Query (LINQ), que fornece recursos de consulta internos através de uma variedade de fontes de dados.

2.3.4 Maven

Maven [71] é uma ferramenta que pode ser usada para a construção automatizada e gestão de qualquer projeto baseado em Java. Através de um repositório central o Maven permite gerir dependências entre projetos internos e externos. Com o intuito de suportar no trabalho do dia-a-dia dos programadores Java, torná-lo mais fácil e, geralmente, ajudar com a compreensão de qualquer projeto baseado na linguagem Java.

O principal objetivo do Maven é permitir que um programador possa compreender o estado completo de um desenvolvimento no menor esforço e período de tempo. Para atingir este objetivo há várias áreas de preocupação que o Maven tenta lidar:

- Tornar o processo de compilação fácil;
- Fornecer um sistema de construção uniforme;
- Fornecendo informações sobre o projeto de qualidade;
- Fornecendo orientações para melhor desenvolvimento de práticas;
- Permitindo a migração transparente para novas funcionalidades.

2.3.5 Spring Famework

O Spring Framework [72] é uma plataforma Java que fornece suporte de infraestrutura abrangente para desenvolvimento de aplicações Java. Esta *framework* permite construir aplicações de "Plain Old Java Objects"[3] (POJOs) e aplicar serviços da empresa de forma não invasiva a POJOs. Esse recurso aplica-se ao modelo de programação Java SE e total e parcial Java EE.

Apresentam-se de seguida alguns exemplos de como um programador pode beneficiar na construção de aplicações através da plataforma de Spring:

- Construir um método na linguagem Java e executá-lo numa transação de base de dados sem ter que lidar com APIs de transação;
- Fazer um método Java local, um procedimento remoto sem ter que lidar com APIs remotas;
- Fazer um método Java local, uma operação de gestão sem ter que lidar com JMX APIs;
- Fazer um método Java local, um manipulador de mensagem sem ter que lidar com JMS APIs.

2.3.6 WCF Framework

O Windows Communication Foundation (WCF) [73] é uma estrutura para a construção de aplicações orientadas a serviços. Usando WCF, é possível enviar dados como mensagens assíncronas a partir de um *endpoint* do serviço para outro. Um *endpoint* do serviço pode ser parte de um serviço continuamente

[3] POJO é tipicamente um objeto na linguagem de programação Java que não detém de nenhuma implementação de interfaces, anotações e sem dependências de outros objetos de forma a ser compatível uma *framework*.

disponível hospedado pelo IIS, ou pode ser um serviço hospedado numa aplicação. Um ponto de extremidade pode ser um cliente de um serviço que solicita dados de outro *endpoint* de outro serviço. As mensagens podem ser tão simples como um único caractere ou palavra enviados como XML, ou tão complexo como um fluxo de dados binários. Alguns cenários de exemplos incluem:

- Um serviço seguro para processar transações de negócios;
- Um serviço que fornece dados atuais aos outros, como um relatório de tráfego ou outro serviço de monitoramento;
- Um serviço de *chat* que permite a duas pessoas comunicarem ou trocarem dados em tempo real;
- Uma aplicação de painel de pesquisas de um ou mais serviços de dados e apresenta-lo em uma apresentação lógica;
- Expondo um fluxo de trabalho implementado usando o Windows Workflow Foundation como um serviço WCF.

O WCF torna o desenvolvimento de aplicações mais fácil do que nunca, sendo projetado para oferecer uma abordagem de gestão para criar serviços em páginas na Internet e clientes de serviços *para aceder as páginas* [73].

2.3.7 XML e XSD

O Extensible Markup Language (XML) [74] é um formato de texto simples, muito flexível derivado de SGML (ISO 8879). Originalmente concebido para enfrentar os desafios da publicação eletrónica em grande escala, o XML também está a desempenhar um papel cada vez mais importante na troca de uma ampla variedade de dados nas páginas na Internet .

A finalidade de um esquema XSD é definir e descrever uma classe de documentos XML, usando componentes de esquema para restringir e documentar o significado, o uso e as relações de suas partes constituintes: Tipos de dados, elementos, atributos e seus valores. Os esquemas também podem fornecer para a especificação de informações de documentos adicionais, tais como a normalização e o incumprimento de valores de atributos e elementos. Os esquemas têm instalações para auto

documentação. Assim, as estruturas do XSD podem ser usadas para definir e descrever vocabulários para definição de ficheiros XML.

Qualquer aplicação que consome XML bem formado pode usar as regras aqui definidas para expressar, estruturas bem definidas com valores sintáticos a serem aplicáveis às suas instâncias do documento. O formalismo XSD permite um nível útil de verificação de restrição a ser descrito e implementado para um amplo espectro de aplicações XML. No entanto, o idioma definido por esta especificação não tenta fornecer todas as facilidades que possam ser necessárias para as aplicações. Algumas aplicações podem exigir capacidades de restrição não definidas no XSD, e assim vai precisar para realizar suas próprias validações adicionais.

2.3.8 AngularJS

AngularJS [75] é tipicamente por definição uma framework, apesar de muitas vezes pelo facto de ser muito leve ser considerada como uma biblioteca. É desenvolvida cem porcento em *JavaScript,* sendo cem porcento utilizada do lado do cliente e compatível tanto com navegadores de computadores como navegadores de dispositivos moveis.

Em alto nível o *AngularJS* parece-se com mais um sistema de modelos. Mas existe uma característica que o diferencia, pelo facto de permitir a ligação bidirecional de dados, sendo que o modelo é compilado no navegador e ao mesmo tempo produz o resultado para visualização dos dados. Isto significa que o programador não precisa de escrever código e estar constantemente a sincronizar a camada de visualização com a de modelo, como acontece normalmente com outros sistemas de modelos.

2.4 Sumário

Neste capítulo, foi possível estudar e analisar a evolução dos negócios de retalho desde o início dos tempo onde os negócios eram simples trocas de um produto por outro até os dias de hoje onde existe pagamentos representados por moeda física ou virtual. Foi realizado também o enquadramento dos sistemas de retalho computorizados, no seu aparecimento em 1951 até os dias de hoje, frisando a sua evolução ao longo dos anos.

Foi explicado o que consiste um sistema de ecommerce, assim como os seus respetivos benefícios para o negócio do retalho, assim com descritos os vários tipos de negócios *online* que podem existir. Por fim apresentadas as várias camadas de *software* que estão presentes no desenvolvimento de um sistema de ecommerce.

Foi apresentada a arquitetura Model Driven Architeture (MDA), onde foi descrita sua visão no processo de desenvolvimento de *software*, a forma como define conceitos e definições que permitem uma abstração de alto nível aplicável nos mais diversos conceitos de negócio e linguagens de programação. Foi também comparado o tradicional modelo de desenvolvimento de *software* com o modelo de desenvolvimento por MDA, assim como algumas ferramentas que já existem no mercado tanto *open-source* como comerciais para fazer este tipo de desenvolvimento. Por fim, são apresentados conceitos e tecnologias que serão utilizados para o desenvolvimento do sistema por MDA proposto no âmbito desta dissertação.

3 Mecanismo de Geração de Código

Depois de efetuado o estudo do estado da arte será abordada a análise do sistema a ser desenvolvido e o processo de implementação da solução proposta. Nesta fase são identificados os requisitos funcionais e não funcionais e a proposta da arquitetura do sistema proposto que irá suportar o sistema. Na implementação serão abordados os detalhes do desenvolvimento da solução proposta.

3.1 Análise

Nesta secção, são identificados os requisitos necessários para a correta utilização por parte dos programadores e/ou gestores de projetos do Motor de Geração de Código (MGC), requisitos esses que vão guiar o processo de desenvolvimento e possibilitar que esta funcione corretamente e que gere o *output* desejado pelo utilizador.

Assim, será apresentada ao utilizador uma síntese dos requisitos necessários para o desenvolvimento do MGC, com o objetivo de ajudar o leitor a entender a estrutura lógica da solução, adaptação, correta utilização e implementações futuras.

3.1.1 Levantamento de Requisitos

O levantamento de requisitos abrange o processo de recolha de toda a informação necessária para a definição das funcionalidades que o *software* deve ter, do ambiente interativo desejado e do âmbito pretendido para a gestão de aplicações.

Requisitos Funcionais

Após a realização do enquadramento das tecnologias, do estado da arte e a familiarização com a temática da criação de aplicações e respetivos conceitos, procedeu-se à identificação das funcionalidades que deveriam ser desenvolvidas para a construção do MGC, as quais se encontram listadas de seguida:

- **Ler CIM criados pelo utilizador -** A plataforma deve ser capaz de suportar a leitura de ficheiros CIM definidos pelo utilizador. Estes ficheiros CIM devem seguir uma estrutura definida pelo MGC. Os seguintes elementos devem estar presentes:
    - ***Entities*** - As entidades são a parte representativa abstrata na qual o utilizador pode definir as partes intervenientes no sistema e representar as suas características;
    - ***Relationship*** - Representa a relação entre duas entidades;
    - ***Transformation*** - Representa uma transformação que vai ocorrer perante as entidades definidas previamente;
- **Validador de CIM -** A plataforma deve ter a capacidade de validar se o CIM introduzido pelo utilizador se encontra corretamente estruturado e para isso será necessário desenvolver um **XML Schema** que possa fazer essa validação juntamente com a leitura do PIM.
- **Adição de Módulos -** A plataforma deve ser capaz de gerar *output's* através das transformações definidas do MDA, relativamente à passagem do modelo PIM para PSM. Estes módulos devem poder ser adicionados pelos utilizadores em qualquer fase do ciclo de vida do *software*.
- **Utilização de Módulos -** A área de aplicação alvo de atenção nesta dissertação está relacionada com o retalho *online*, sendo necessário o desenvolvimento de um conjunto de módulos para que seja possível criar este tipo de negócios online. Como tal é espectável que o seguinte conjunto de módulos sejam desenvolvidos:

- **Gerar código SQL para a criação da base de dados -** O MGC deve, através das entidades definidas nos PIM, proceder à criação de uma base de dados relacional com as respetivas relações, de forma a poder suportar as entidades definidas. Neste módulo, é espectável que os PIM definidos no sistema sejam transformados em tabelas SQL, com os respetivos atributos e tipos de dados definidos. Relações e tabelas intermédias devem também ser geradas neste processo.

- **Gerar código Java de suporte ao modelo de negócio -** O MGC deve também gerar código Java que serve de suporte para a modelação das entidades especificadas nas entidades definidas nos PIM Estas também devem ser relacionadas consoante os dados definidos no PIM.

- **Gerar código Java e C# que sirva de suporte para a visualização de dados relacionados com o negócio -** O MGC, tendo em conta os dados definidos no PIM, deve gerar artefactos tanto em Java como em C# para que seja possível visualizar dados relacionados com as entidades previamente. Estes artefactos serviram para representar as entidades do sistema definidas no CIM, e por sua vez vão ser utilizados nas transações futuras do sistema.

Requisitos não funcionais

Como qualquer desenvolvimento de *software* existem requisitos que, apesar de não corresponder às funcionalidades do sistema, são sempre tidos em conta quando se está a desenvolver. Ligados à qualidade do *software existem sempre* diversos aspetos que devem ser tidos em conta no seu processo de elaboração. Deles poderá depender o sucesso do projeto no presente e com continuidade no futuro. Referem-se alguns dos aspetos não funcionais identificados:

- **Segurança -** A plataforma será desenvolvida em Java que, por sua vez, já é uma das linguagens de programação mais seguras. Aplicações em Java correm numa Java Virtual Machine (JVM) que mantém os seus processos protegidos num tipo de *Sandbox* [4] da própria JVM permitindo assim o isolamento de aplicações maliciosas externas. Esta também proporciona um *garbage colector* que

[4] A Sandbox é uma ferramenta eficaz e simples que isola a execução de programas e seus processos, tornando possível testar as suas operações num ambiente seguro e controlado.

faz a gestão da memória dinamicamente, permitindo uma melhor gestão dos recursos e evitando assim problemas de falta de memória das aplicações. Cuidados na comunicação das aplicações também foram tidos em conta pois não existe passagem de texto em claro em nenhum dos componentes, já que toda informação é encapsulada como objectos, não sendo possível assim ser informação intersectada as claras por utilizadores maliciosos. Como não existe qualquer contacto do MGC com a Internet esta permanece de portas fechadas para o mundo exterior, reduzindo assim inúmeros perigos que o acesso ao exterior, como possíveis ataques de hackers.

- **Usabilidade -** A plataforma a ser desenvolvida deverá possuir uma interface através de consola com fácil interação de comandos, contudo apesar de não ser a mais apelativa, esta será desenvolvida de forma a permitir extensões futuras na forma de comunicação e interação. A aplicação encontra-se desenvolvida numa linguagem multiplataforma, permitindo assim utilizadores de qualquer plataforma usufruir dela.

- **Ambiente de desenvolvimento -** A plataforma será desenvolvida sobre a última versão de Java SE até o momento disponível, ou seja, a versão 8 do Java. Os projetos serão criados através do Maven 3.3.3 que irá fazer também a gestão de dependências necessárias tanto para o MGC como para os módulos que serão criados para interagir com ele. No desenvolvimento dos módulos, o módulo que permitirá a criação de serviços para páginas na Internet, em Java, será desenvolvido através da *framework* Spring Boot. O módulo que permitirá o desenvolvimento de serviços em .NET será o *Windows Communication Foundation*(WCF). Módulos como o HTML, o Bootstrap e o AngularJS também serão adicionados para criar e gerir as páginas que darão a forma ao EC.

- **Hardware e Software -** A plataforma deverá encontrar-se alojada numa máquina com o sistema operativo Mac OS X e NetBeans, sendo necessário estar instalado o Java e o Maven.

- **Escalável -** O sistema deve ser escalável ao ponto de aceitar novos módulos independentemente da plataforma utilizada e/ou da tecnologia e do tipo de output que é gerado para os PSM. Deve ainda ser possível construir camadas de iteração com o MGC para que o utilizador possa interagir de formas diferentes com o MGC.

- **Padrão de desenvolvimento -** O código fonte da página na Internet, segue o padrão MVC [76] . A *framework* AngularJS utiliza como base este modelo de arquitetura de *software* composto por três camadas que permite separar a camada de acesso a dados, camada de negócio e camada de

apresentação. A representa a relação entre essas camadas.

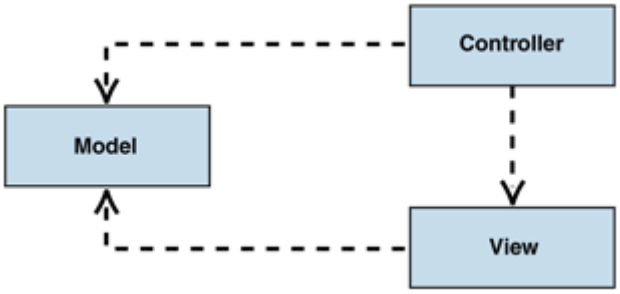

Figura 22 – Estrutura do padrão MVC

Este padrão é importante, pois promove a reutilização de código, facilita a sua manutenção e permite manter o código organizado/limpo. O *Model* é independente da interface apresentada ao utilizador e é responsável pela representação dos dados, oferecendo meios de acesso (leitura e escrita). Por outro lado, o *Controller* processa e responde a eventos, geralmente ações do utilizador, e interage diretamente com o *Model* para satisfazer essas ações. A validação e filtragem de dados introduzidos pelo utilizador são realizadas no *Controller*. Esta camada não é responsável pela obtenção dos dados (responsabilidade do *Model*) nem pela sua exibição (responsabilidade da *View*), serve para controlar as outras duas camadas como um todo. Por fim, a *View* é responsável pela exibição de dados, sendo com esta camada que o utilizador interage. O código do MGC será desenvolvido como já foi referido previamente segundo o modelo de arquitetura do MDA, que engloba a transformação sucessiva de três artefactos, o CIM para um PIM e este para um PSM. A arquitetura que irá ser praticada nos *WebServices* Spring é a arquitetura RESTFul e arquitetura a Spring Data JPA. Relativamente aos *WebServices* desenvolvidos em C# a serem publicados, estes seguem a arquitetura definida pelo WCF.

Com base no estudo do Estado da Arte e nas funcionalidades identificadas no levantamento de requisitos que devem ser desenvolvidas para a construção da plataforma, é possível concluir que nenhuma das ferramentas estudadas satisfaz o problema na totalidade, pois nenhuma permite a criação e manutenção de reduzido custo e tempo, considerando a evolução de mercado, de modo a ser capaz de dar resposta em tempo útil e no orçamento desejado.

3.1.2 Sistema Proposto

Na presente secção é descrita a arquitetura do sistema proposto para o desenvolvimento da plataforma, com base no levantamento de requisitos e no estudo de conceitos relacionados com o tema em estudo.

De seguida é apresentada a arquitetura do sistema proposto, sendo abordadas em detalhe algumas das suas caraterísticas. A arquitetura encontra-se dividida em três componentes (Figura 23) e módulos que irão ser criados para interagir com o MGC. O MGC encontra-se a funcionar como ponto central, este será o responsável por realizar as transformações típicas de uma arquitetura MDA, leitura e validação do CIM, estruturação dos PIM gerados e controlo de fluxo das suas sucessivas transformações e por fim proceder a transformação e gravação dos ficheiros de Plataform Specific Model(PSM). Os módulos serão desenvolvidos para responder às necessidades identificadas no levantamento de requisitos de forma a estruturar os dados que o MGC irá usar na transformação dos Plataforma Independe Model (PIM) para PSM.

Sendo o objetivo principal desta dissertação a criação de um MGC que permita gerar artefactos de *software*, mais propriamente que permitam ajudar na criação, atualização e manutenção de lojas *online*, torna-se importante falar da estrutura comum. Existem vários tipos de lojas *online* e com diferentes dimensões e especificações.

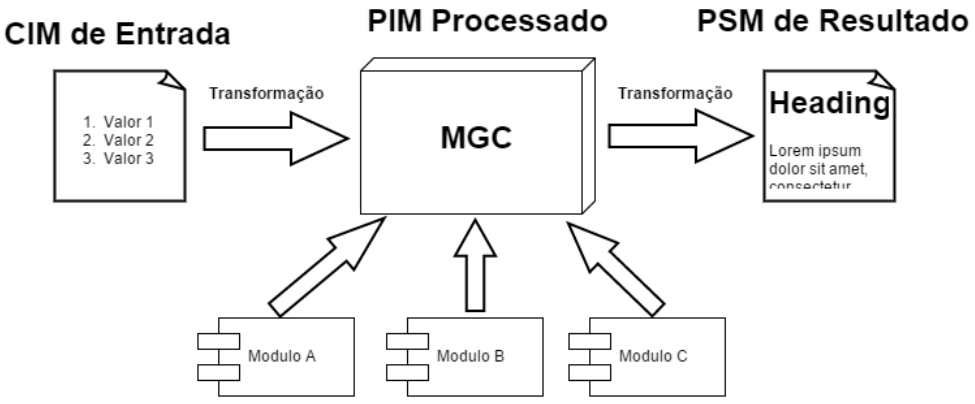

Figura 23 – Diagrama de blocos do sistema

Na Figura 24 é possível visualizar um exemplo de uma estrutura de uma loja *online* básica. Para efeitos desta dissertação os módulos que vão ser criados pelo MGC serão mais focados nos componentes dos *Produtos* e *Utilizadores* para provar que o conceito pode ser possível e crescer para os outros

componentes, pois apesar de ser um sistema simples seria necessário criar vários módulos para criar um sistema desta dimensão de forma dinâmica.

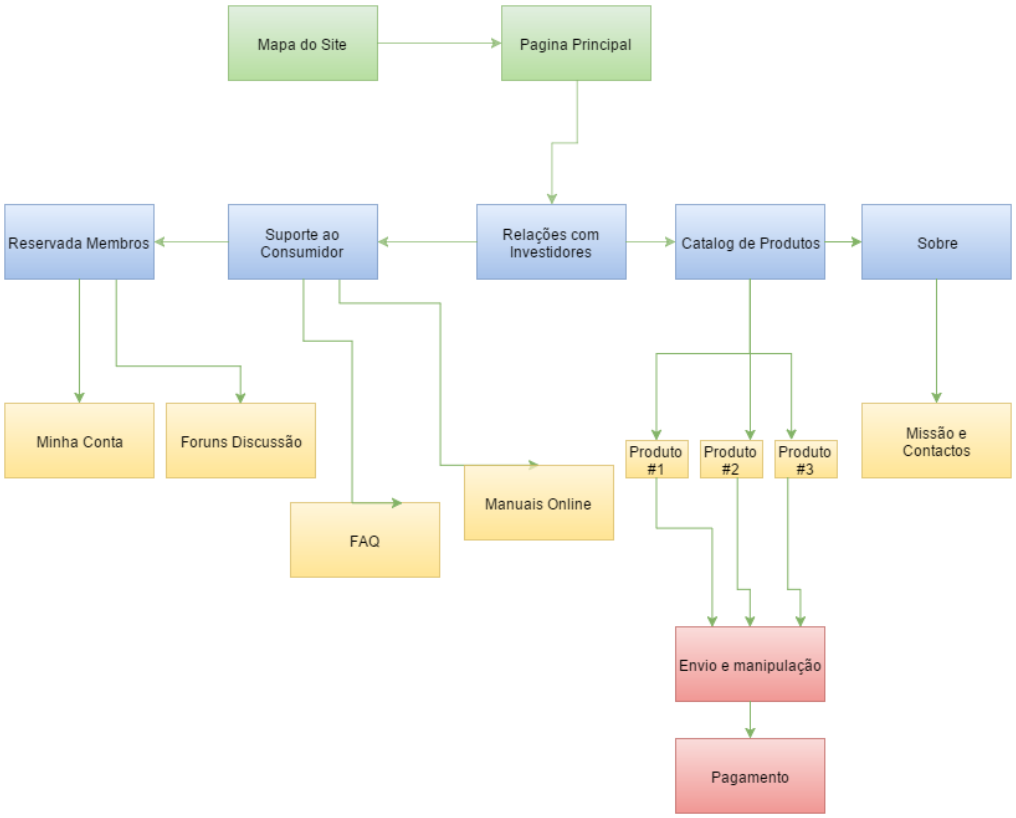

Figura 24 – Site map comum de um *site* para retalho

A análise efetuada no âmbito do presente estudo foi dividida em duas fases. A primeira, o levantamento de requisitos, serviu para identificar as funcionalidades que a plataforma tem que conter, de forma a satisfazer os critérios exigidos.

A segunda fase consistiu no desenho de um sistema proposto, com base nos requisitos funcionais e não funcionais identificados. No estudo das arquiteturas foi possível identificar uma estrutura para o MGC que será criado assim como um modelo para um simples E-Commerce. Estes modelos de arquitetura revelaram-se muito importantes para o desenvolvimento da plataforma, pois permitiu a definição dos objetivos a atingir na fase seguinte.

3.2 Implementação

Terminada a fase descrita na seção 3.1, procedeu-se à implementação de todos os componentes identificados na seção Sistema Proposto

Ao longo desta secção é abordada a fase de desenvolvimento do MGC, assim como os respetivos módulos que o acompanham nesta fase, sendo apresentadas e descritas as principais funcionalidades e indicada a forma como foram implementadas.

Com base no que já tinha sido definido previamente, na fase de análise, no diagrama de blocos e após estruturar qual seria o desenvolvimento necessário para a criação de um MGC que fosse capaz de gerar artefactos de valor acrescentado para a criação de um EC. Foi desenhado um diagrama de *deployment* de forma a representar os principais componentes a serem desenvolvidos, tal como é possível visualizar na Figura 25.

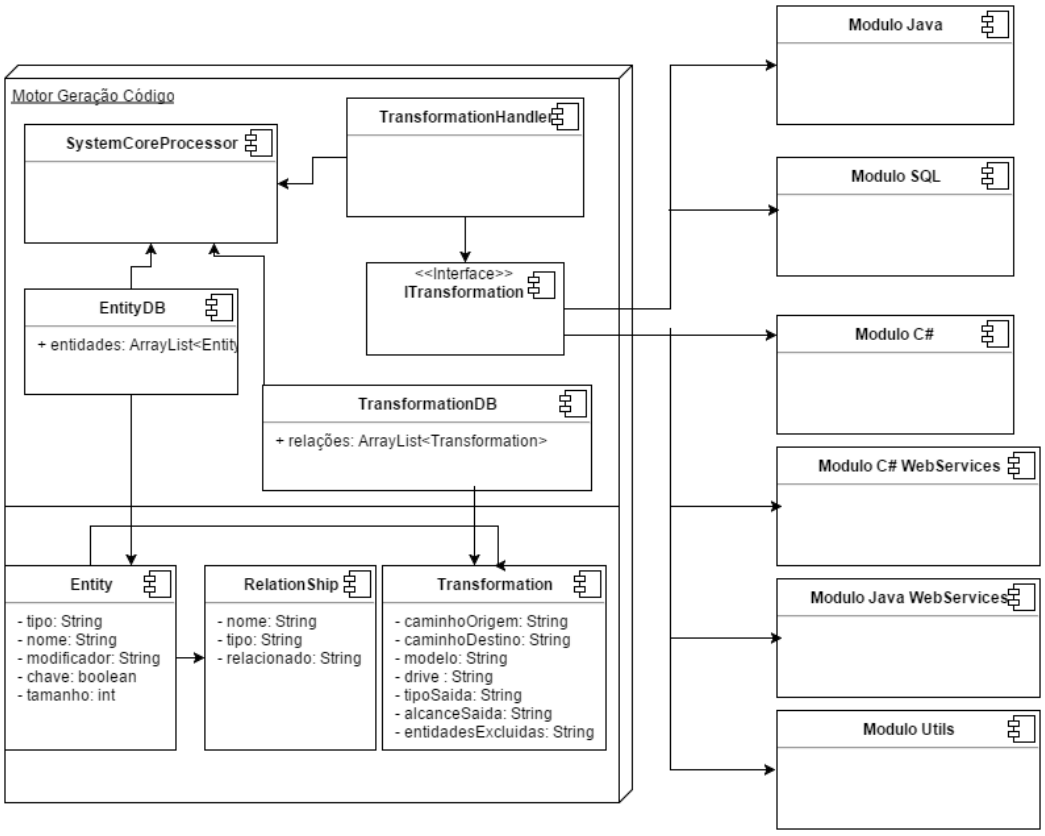

Figura 25 – Diagrama de *deployment* do sistema

Entity é algo que representa um conceito, uma classe, um objeto, entre outras representações. Funciona como o ponto de partida para representar algo em que o utilizador tenha como parte de interesse para um sistema que de alguma forma precisa ter sido em conta para a sua existência faça sentido. *RelationShip* representa as relações entre *Entity,* caso seja necessário e faça sentido que estas se relacionem no sistema. *Transformation* representa um comportamento que o utilizador pretende definir para uma ou mais *Entity*.

As classes *EntityDB* e *RelationShipDB* serão responsáveis por armazenar toda a informação relativamente às *Entity* e *RelationShip*. *TransformationHandler* será o componente responsável por, através da interface definida pelo MGC, invocar todos os módulos que estarão à disposição do MGC. Por fim, o último componente *SystemCoreProcessor será o* responsável por gerir todo o fluxo da aplicação.

3.2.1 Arranque e configuração MGC

O MGC é desenvolvido em Java assim como todos os módulos usados para interagir com o MGC. Este necessita em tempo de execução de conhecer previamente todos os módulos e dependências que estão disponíveis para poder utilizar as bibliotecas externas e os módulos criados de forma a poder gerar o *output* desejado. Para que isto possa ser possível a JVM necessita, ao "lançar" a MGC, ter todas essas dependências ligadas, caso contrário o MGC iria emitir erros por desconhecimento de informação que só existe nessas dependências.

Este processo pode ser realizado de várias formas umas com as suas vantagens e desvantagens. Uma das formas poderia ser acrescentando as dependências dos módulos no projeto do MGC e gerar de novo o seu ficheiro binário, mas iria obrigar o utilizador que está a desenvolver os módulos a ter que conhecer e a ter acesso ao código do MGC, o que pode não ser o desejado. Além disso, torna o conteúdo mais estático no processo de escalabilidade. Outra forma seria fazer *update* do ficheiro **pom.xml**[5] que é consultado pelo Maven no processo de compilação do MGC. Este passo seria mais dinâmico e coerente com o processo do MGC, mas ao mesmo tempo criaria uma dependência para o utilizador, pois obrigá-lo-ia a compilar o projeto sempre em Maven e este processo podia não ir de encontro com o que os programadores estariam à procura caso fosse preciso desenvolver novos módulos para o MGC. Assim, tendo em conta

[5] O ficheiro pom.xml é um ficheiro utilizado pelos projetos *Maven* para definir as propriedades que vão tido em conta no processo de compilação. Aqui definem-se dependências, versões java de compilação, propriedades, entre outras.

estas alternativas foi utilizada para o arranque e configuração de um ficheiro de *bash script*. Este foi designado como **run.sh** e está programado para correr no mesmo diretório que o ficheiro binário do MGC e se encarregar de fazer todo o processo de carregamento de bibliotecas e módulos e, após esse processo estar concluído, arrancar o MGC. O Código Fonte 1 apresenta o conteúdo do ficheiro **run.sh**.

```
clear

echo "Starting Code Generator Engine!..."

echo "Loading Libs..."
LIBS="libs/*"

echo "Loading Modules..."
MODULES="modules/*"

ENGINE="CodeGenerationEngine-1.0-SNAPSHOT.jar"

echo "Starting engine..."
echo "Using configuration :" + java -classpath $ENGINE:$LIBS:$MODULES
code.codegenerationengine.App;

java -classpath $ENGINE:$MODULES:$LIBS code.codegenerationengine.App $1
```

Código Fonte 1 – Arranque do MGC

Este ficheiro interage com o utilizador enviando algumas mensagens sobre o processo de carregamento e arranque. É simples, mas que pode ser alterado para ir de encontro às necessidades do utilizador que interage com o MGC, sendo atualmente constituído por três principais varáveis: **Libs**, **Modules** e **Engine**.

LIBS é a variável que contém o caminho onde todas a bibliotecas externas devem ser inseridas sob a forma de ficheiros binários ***.jar** das quais são indispensáveis para o funcionamento do MGC e dos módulos. Por outro lado**, MODULES** é a variável que contem o caminho para todos os módulos que o utilizador pretenda que estejam disponíveis a serem usados pelo MGC. Não sendo obrigatório que todos sejam utilizados. Tanto na variável **LIBS** como na **MODULES,** representadas acima, estão a referenciar todos os ficheiros contidos na pasta através do **wild-card(*)** inserido à frente do caminho da pasta, por uma questão de carregamentos mais dinâmicos e redução de manutenção do ficheiro. Contudo, a declaração implícita dos módulos pode ser feita incluindo o caminho para cada ficheiro binário e separando os mesmos por um ":" em sistemas UNIX, ou com ";" em sistemas Windows. Por fim, a variável **ENGINE** fica a apontar para o caminho e o nome do binário do MGC.

Após estas definições de variáveis necessárias para o arranque o script corre a aplicação tendo em conta a informação nela contida. De seguida, após confirmar as configurações no ficheiro **run.sh** e correr o ficheiro irá aparecer uma mensagem de erro ao utilizador. Esta mensagem irá conter o seguinte texto "*You need to specify some parameters. Use -help for more information*". Isto deve-se ao facto do MGC ter sido desenvolvido de forma a que no futuro possam existir outras formas de iteração, que não apenas por consola, de forma a proporcionar uma melhor utilização para o utilizador final. Nesta dissertação apenas o modo de consola está disponível sendo necessário correr o MGC com o modo **run.sh -console**.

Para finalizar o processo de arranque e configuração é necessário o utilizador introduzir o caminho para o ficheiro **CIM** que irá representar todo o sistema a ser gerado.

3.2.2 Configuração do CIM do MGC

Como já foi referenciado anteriormente, o CIM é o primeiro artefacto de uma arquitetura MDA. Este pretende representar como o próprio nome sugere um modelo de computador independente. É representado por um ficheiro em formato **xml** de forma a seguir uma estrutura que possibilita uma fácil reutilização, assim como uma fácil compreensão. Os dados, ao serem identificados sob forma de *tags,* podem facilmente ser processados em qualquer tipo de sistema ou arquitetura de computação. Esta portabilidade associada aos ficheiros **xml** que a fez tornar numa das mais populares tecnologias para trocar dados, foi a base de sustentação para representar o CIM do MGC.

Apesar do MDA ser uma ferramenta abstrata que pretende ser o mais flexível possível, é necessário em algum ponto definir regras de formatação, para que seja possível manter uma coerência no desenvolvimento do CIM e para que seja possível o MGC reconhecer a estrutura que está definida no nele.

Para isso foi criado um XML Schema (fichero **.xsd**) de forma a poder validar a estrutura do ficheiro CIM. É possível verificar se a estrutura do PIM foi bem desenvolvida, assim como orientar o utilizador com mensagens para a correta formatação.

O formato do CIM definido no Código Fonte 2 foi desenvolvido de forma a poder representar todas as possíveis estruturas de desenvolvimento de *software*.

```
<system name="">
    <entities>
        <entity name="">
            <attribute type="" name="" modifier="" length=""></attribute>
            <attribute type="" name="" modifier=""></attribute>
            <relationship type="" with=""></relationship>
        </entity>
    </entities>
    <transformations>
        <transformation>
            <destinationPath></destinationPath>
            <template></template>
            <drive></drive>
            <outputType></outputType>
            <outputRange></outputRange>
        </transformation>
    </transformations>
</system>
```

Código Fonte 2 - Exemplo CIM

Ao começar o desenvolvimento de um CIM o primeiro passo é definir o nome para o sistema que estamos a pretender construir. Este passo traduz-se por introduzir o nome dentro do atributo *name* no elemento sistema, como no exemplo seguinte:

```
<system name="Nome do Sistema">
```

Depois é necessário definir quais as entidades necessárias para que o sistema possa existir e fazer sentido no contexto para o qual está a ser usado. É a volta delas que todo o sistema vai ser gerado em conformidade com as configurações atribuídas a este ponto no CIM.

Estas representam elementos complexos pois podem existir inúmeras entidades no sistema e cada uma delas sendo também um elemento complexo pois contém outros elementos que representam atributos dessa mesma entidade assim como possíveis relações que podem ou não ter com outras entidades.

Assim, para validação dessas entidades o XML Schema define que, para que um sistema possa existir, é necessário existir pelo menos uma entidade, porque sem elas não faz sentido sequer existir um sistema e, ao mesmo tempo, o número que podem estar envolvidas não pode ser calculável. Sendo assim, a regra é definida do seguinte modo:

```
<xs:element name="entity" minOccurs="1" maxOccurs="unbounded">
```

Relativamente aos atributos e relações que existem nas entidades estes podem não existir, ou então, ainda não estarem definidos no momento inicial em que o CIM está a ser desenvolvido. Estes elementos são opcionais e podem ser representados como o exemplo no Código Fonte 3.

```
<xs:element name="attribute" minOccurs="0" maxOccurs="unbounded">
<xs:element name="relationship" minOccurs="0" maxOccurs="unbounded">
```

Código Fonte 3 - Código com os elementos de um elemento **entity**

Cada um destes elementos é constituído por um conjunto de atributos que pretendem especificar ao máximo as características dos elementos. O atributo é constituído pelos elementos apresentados no Código Fonte 4.

```
<xs:attribute name="type" type="xs:string"/>
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="modifier" type="xs:string"/>
<xs:attribute name="iskey" type="xs:boolean"/>
<xs:attribute name="length" type="xs:integer"/>
```

Código Fonte 4 - Código com os atributos do elemento **attribute**

Este com a maioria dos constituintes de um sistema requer um **nome** pelo qual vai ser identificado, depois atributos com o **type, modifier, length e isKey** são usados para identificar qual o tipo de dados que este atributo pretende representar assim como a sua visibilidade, tamanho e se representa um campo chave para entidade.

Relativamente às relações, estas não necessitam de ter um nome pois representam apenas a relação entre a entidade ao qual esta relação se encontra associada e a entidade representada no atributo **with**. Relativamente ao campo **type**, este representa o tipo de relação que esta entidade vai manter com as outras entidades que vão de encontro às que são de definidas no Modelo Entidade-Relação (Modelo ER). As relações podem ser de 1-1, 1-n ou n-n. Uma relação pode ser do tipo **1-1**, que **r**epresenta a cardinalidade entre duas entidades, sendo que as mesmas apenas têm um relacionamento entre si**.** Um relação de **1-n** representa a cardinalidade em que uma entidade vai estar referenciada na outra repetidas vezes, usando para isso o campo que contem o atributo **isKey** definido previamente. Por último, uma relação do tipo **n-n** representa a cardinalidade em que o relacionamento entre duas entidades que é

repetido inúmeras vezes e que por sua vez se relacionam. Para esta finalidade é necessário criar uma entidade auxiliar que irá conter os dois atributos **isKey** das entidades.

Esta fase de especificação das entidades pode ser desenvolvida por programadores, mas não só. Tipicamente num cenário ideal de desenvolvimento de *software* as equipas reúnem-se para discutir o que é necessário existir no sistema, tendo em conta os requisitos reunidos através de reuniões com clientes. Neste processo de diálogo e troca de ideias pode descrever-se as entidades necessárias para o sistema enquadradas diretamente no CIM. Desta forma, é possível registar o conteúdo da reunião, assim como contribuir desde início para um sistema desenvolvido por MDA e acelerar logo o processo de desenvolvimento de *software*.

Após configurar todas as Entidades e suas características, o próximo passo será definir as transformações que ocorrem no sistema. Essas transformações representam utilizações dos módulos carregados no arranque que nesta fase o utilizar irá definir a sua forma de utilização.

```
<xs:element name="transformation" minOccurs="1" maxOccurs="unbounded">
```

Da mesma forma que está definida nas entidades é necessário definir pelo menos uma transformação pois, apesar de o sistema já conter as entidades definidas, é sempre necessário definir quais serão as ações a realizar com elas. Essa definição expressa-se por definir transformações que vão ocorrer no MGC ao utilizar os módulos juntamente com as Entidades e os *templates* que são disponibilizados.

```
<xs:element name="destinationPath" minOccurs="1" maxOccurs="1"
type="xs:string"></xs:element>
<xs:element name="template" minOccurs="1" maxOccurs="1"
type="xs:string"></xs:element>
<xs:element name="drive" minOccurs="1" maxOccurs="1"
type="xs:string"></xs:element>
<xs:element name="outputType" minOccurs="1" maxOccurs="1"
type="xs:string"></xs:element>
<xs:element name="outputRange" minOccurs="1" maxOccurs="1"
type="xs:string"></xs:element>
<xs:element name="excludedEntities" minOccurs="0" maxOccurs="1"
type="xs:string"></xs:element>
```

Código Fonte 5 - Código elementos de uma ***transformation***

Para isso o elemento complexo de Transformação conta com um conjunto de outros elementos necessários para serem configurados para que a transformação possa efetivamente ocorrer. Apresenta-se de seguida no Código Fonte 5 .

Em cada uma das transformações é sempre necessário definir qual o destino para onde o utilizador pretende que o *output* desta transformação seja redirecionado. O *template* representa o guia de comportamento que o MGC vai utilizar para gerar o *output* processado transformando assim o PIM num PSM. O *drive* é o ponto de entrada que o MGC irá utilizar para chamar a implementação definida em cada um dos módulos. O tipo de *output, que* está diretamente relacionado com o *output* gerado, será o tipo que os ficheiros vão assumir quando são gerados. É também possível excluir Entidades de serem processadas, isto porque pode não fazer sentido algumas entidades serem reproduzidas em alguns módulos. Por fim, existe o elemento que define a forma como vai ser gerado o *output* dos ficheiros, existindo para isso as respetivas configurações disponíveis na Tabela 4. A tabela pretende mostrar as relações entre as entidades e os ficheiros de output, ou seja, uma entidade pode ser mapeada para um vários ficheiros e várias entidades podem ser mapeados para um ou vários ficheiros.

Tabela 4 – Configurações para *output* de ficheiros

Entidades	**Ficheiros**	**Descrição**
1	n	Uma entidade mapeada para vários ficheiros.
1	1	Uma entidade mapeada para um ficheiro.
n	1	Todas as entidades mapeadas para um ficheiro.
n	n	Todas as entidades mapeadas para vários ficheiros.

3.2.3 Transformação CIM para PIM

Após a validação do CIM, o sistema MGC procede à transformação do CIM para um PIM. Este tipicamente deve ser independente da plataforma destino, sendo que, umas das formas mais comuns é utilizar uma tecnologia que corra sobre uma máquina virtual no sistema. O Java foi a opção mais madura para fazer esta implementação. A Figura 26 pretende representar as bases de dados e as respetivas entidades e relações.

As entidades e relações definidas são mapeadas para objetos em Java que contém toda a informação contida no PIM e após esse carregamento, são introduzidos numa base de dados de Entidades e de Transformações respetivamente. Estas bases de dados são depois consultadas nos mais diversos pontos da aplicação para manipulação de informação.

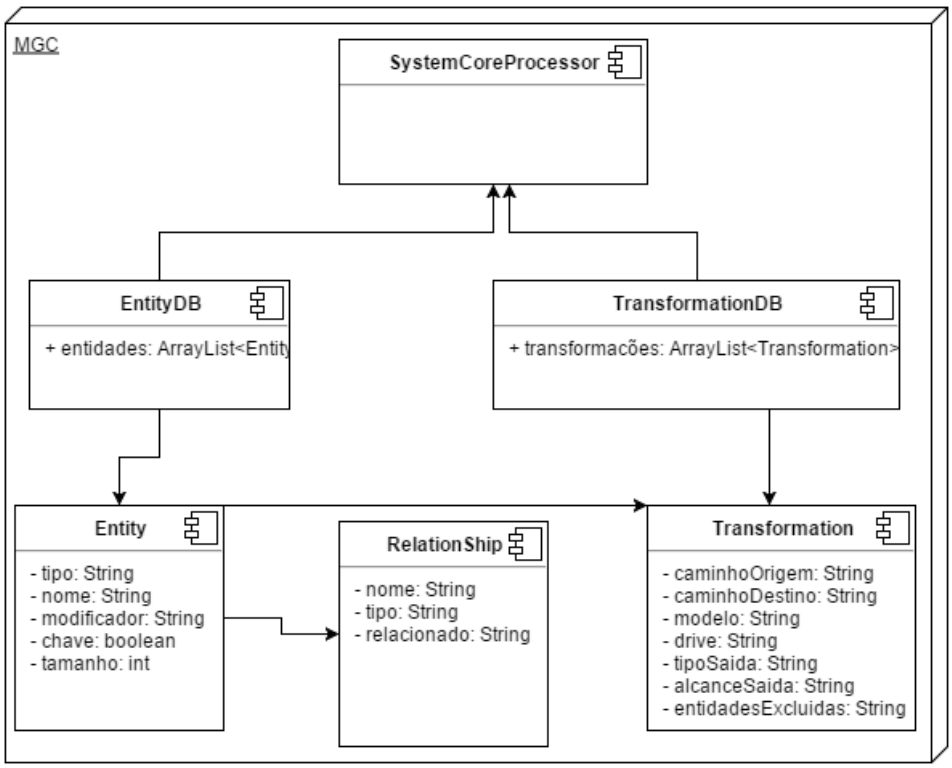

Figura 26 – *Containers* de entidades e relações

3.2.4 Transformação PIM para PSM

Após o MGC proceder à transformação dos CIM para PIM e, caso não exista nenhum erro, o sistema consulta todas as transformações disponíveis na base de dados de transformações e envia para o *transformer handler,* que é responsável por analisar os tipos de transformações que estão pendentes e selecionar qual o tipo de abordagem a seguir perante as entidades carregadas.

Depois de escolher qual a orientação a seguir perante as configurações definidas nas transformações, o *transformer handler* usa a interface definida pelo MGC exposta para os módulos, para invocar os respetivos *drivers*.

A interface exposta com o nome *ITransformation* define os métodos que os módulos necessitam de implementar para que se possa seguir um comportamento bem definido pelo MGC. A interface a ser seguida está presente no Código Fonte 6

.

```
public interface ITransformation {
    public Map<String, Object> transform(Entity entity);
    public Map<String, Object> transform(List<Entity> entities);
    public String getTransformationName();
}
```

Código Fonte 6 - Interface do MGC

Os métodos **transform** recebem as entidades do MGC, e com elas o utilizador pode nos módulos tratar e processar a informação das entidades retornando a informação tratada para ser confrontada pelo MGC com o *template* definido para a respetiva transformação. Por outro lado, método **getTransformationName** permite ao utilizador definir qual o nome do ficheiro do *output,* consoante as regras definidas para os ficheiros gerados. Na Figura 27, é possível observar a representação dos módulos desenvolvidos e a respetiva implementação da interface exposta pelo MGC.

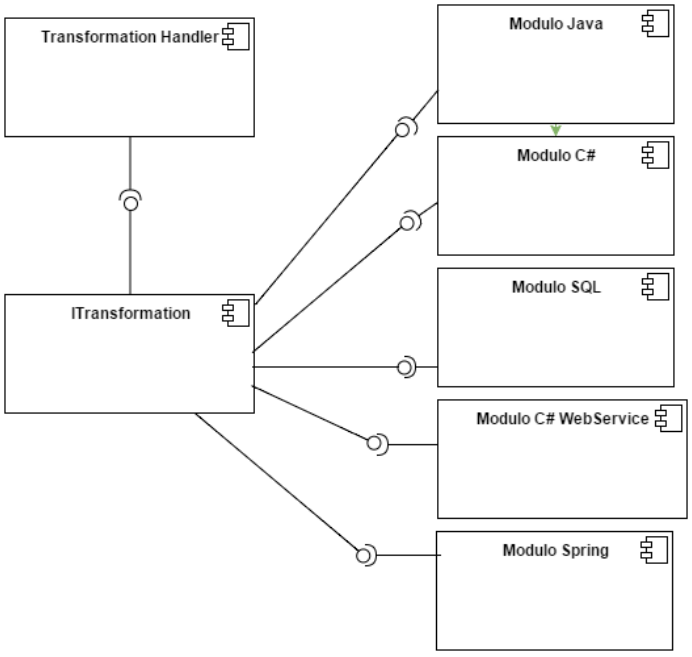

Figura 27 – Módulos e Interface do MGC

3.2.5 Componentes e Módulos da Aplicação

Esta secção tem como finalidade descrever os principais componentes e módulos da aplicação. Neste caso, serão descritos os componentes principais às transformações usadas pela arquitetura MDA no MGC desenvolvido. Os módulos que irão ser desenvolvidos para interagir com o MGC e parte constituinte desta dissertação, também serão descritos nesta secção.

- **Entity -** A *Entity* é um componente abstrata ao contexto que pretende representar algo de lógico para o sistema de *output* para o qual vão existir transformações e relações com outras *Entity*. Por outro lado, é um componente do MGC de geração de código que num sistema MDA representa um PIM. Este traduz de uma forma abstrata para o Sistema um modelo com base no CIM definido no ficheiro de configuração que será utilizado na geração de um PSM;
- **RelationShip -** Uma RelationShip é um componente que faz parte de uma *Entity* e que tem como objetivos representar uma relação de uma *Entity* com outras *Entities* do sistema. No presente MGC são permitidos três tipos de relações entre entidades muito semelhantes aos típicos relacionamentos dos modelos de dados tão bem conhecidos. Uma relação de 1-1, em que uma *Entity* está relacionada com uma outra *Entity* que é definida no campo *relatedTo*. Uma relação de 1-N, em que uma *Entity* está relacionada com varias *Entities*. Por fim, a relação do tipo N-N, onde várias *Entities* estão relacionadas com outras *Entities* de um determinado tipo.
- **Transformation -** *Transformation* é a declaração de um processo de tratamento das *Entities* para o qual vão ser executadas uma série de eventos consoante os módulos atribuídos a cada uma das transformações. A transformação pode ou não ser aplicada a todas as *Entities* pois no cenário para o qual esta vai ser aplicada pode não fazer sentido no contexto. Definem ainda o *template* para o qual vai ser processado o *output* gerado pelos respetivos módulos e o tipo de *output* dos ficheiros destino;

- **EntityDB -** Este componente é responsável por armazenar todas as entidades do sistema e mantê-las em memória para serem consumidas/consultadas nas mais diversas zonas do MGC;
- **SystemCoreProcessor -** É o componente de *backbone* do sistema responsável por executar as tarefas principais do MGC quando este é executado pelo utilizador. Faz a validação do ficheiro PIM produzido pelo utilizador, a respetiva conversão deste para linguagem conhecida pelo MGC e, por fim, invoca o *TransformationHadler* que vai proceder às transformações dessas entidades;

- **TransformationHadler -** O *TransformationHadler*, como já referido em cima, é responsável por tratar do processo de transformação de entidades carregadas pelo sistema. Este, através dos tipos de *output* das transformações, vai gerar os ficheiros correspondentes, com o conteúdo gerado através da chamada dos métodos de transformação declarados na *ITransformation***.** Esta interface está desenvolvida dentro do MGC e é consumida pelos módulos acrescentados no sistema de forma a poder implementar ações sobre as *Entities*;
- **ITransformation -** A *ITransformation* é a interface que se encontra neste momento disponível para ser implementada pelos módulos que disponibiliza alguns métodos, para que na sua implementação seja possível o utilizador definir qual o comportamento que as entidades devem ter perante aquele módulo. Estes métodos tipicamente retornam valores processados que posteriormente são confrontados com os respetivos *templates* da transformação de forma a gerar o respetivo *output*.

Módulos

Os módulos desenvolvidos pelos utilizadores para interagir com o mecanismo de geração de código representam uma parte importante num sistema de desenvolvimento de *software* por MDA. Estes representam os PSM onde o utilizador tem à disposição todos os PIM gerados através das entidades definidas como CIM no ficheiro XML de entrada no gerador de código.

Cada módulo representa um PSM a ser processado pelo MGC, onde o utilizador pode definir o comportamento destino, assim com o tipo de linguagem de programação que este vai gerar como *output* do Sistema. Estes módulos processam de forma lógica o comportamento que é pretendido apresentar para cada uma das entidades do Sistema na linguagem pretendida, sendo que a esta pode ser representativa de um tipo de programação ou meramente um tipo de formato de documentação de texto onde os dados são representados.

Cada um dos módulos, após definida a estrutura de como as entidades vão estar organizadas para serem processadas, utilizam *templates* definidos em Freemarker que vão ser usados como confrontação com a estrutura definida pelo utilizador, de forma a gerar o *output* com o tipo de estrutura que define o tipo escolhido pelo utilizador para o respetivo módulo.

No âmbito desta dissertação foram desenvolvidos vários módulos de forma a poder gerar o máximo de artefactos necessários para o desenvolvimento de um EC. Estes módulos, criados para auxiliar a criação de código para a geração de um EC, são constituídos por módulos de geração de código da aplicação, assim como, de geração de comandos que são executados de forma a gerar as aplicações e de forma a reduzir o tempo para a criação e depuração de projetos.

Todos os módulos obedecem a uma interface bem definida pelo MGC para que seja possível gerar artefactos com base nas entidades especificadas no PIM, assim como seguir uma ordem lógica de transformações com as respetivas entidades.

O MGC recorre também à definição das transformações definidas pelo utilizador para saber qual a estrutura de ficheiros, *template*s, exceções de entidades, tipos de *output*, origem e destino para onde os ficheiros com os resultados dos módulos vão ser gerados.

No âmbito desta dissertação foi utlizado como área de aplicação o EC, linguagens como o Java e o C# foram usadas para gerar diferentes tipos de artefactos de forma a poder gerar partes de um sistema desse EC. Como tal, existe inicialmente um módulo Java e outro em .NET responsável por gerar as classes que representam as entidades necessárias para o sistema de EC definidas pelo utilizador no PIM. Estas podem ser usadas para representar partes constituintes do negócio durante o desenvolvimento do produto.

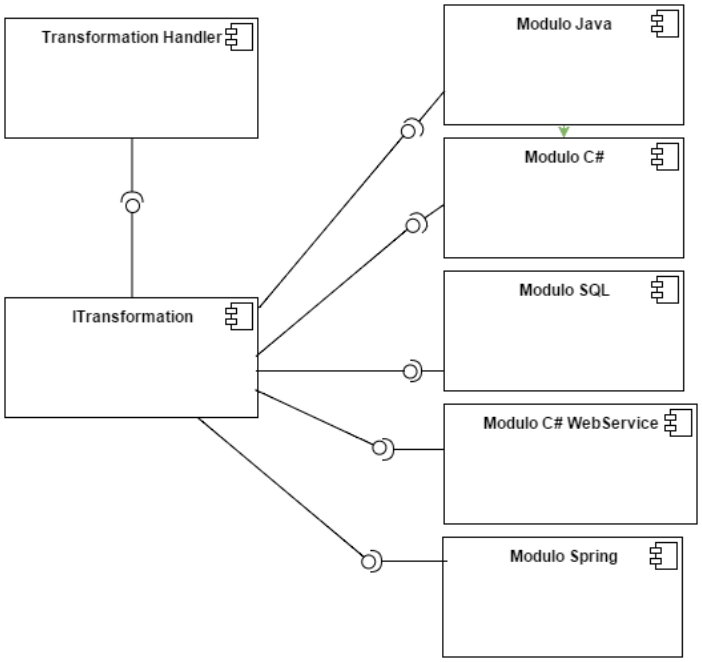

Figura 28 – Módulos e Interface do MGC

Esta informação gerada é usada como base para os módulos de Spring e WebServices em .NET que são responsáveis por gerar *webservices* de forma a poder ser visualizada a informação relativamente aos componentes previamente gerados. Para acelerar o desenvolvimento dos *webservices* e para seguir as mais comuns e modernas tendências do mercado foram usadas *frameworks* como o .NET e o Spring Boot para geração desses *webservices*.

Estes módulos usam as classes de negócio gerados pelos primeiros módulos para construir os artefactos dos *webservices* e comunicam com uma base de dados MySQL, gerada pelo módulo SQL também disponível para o mapeamento das entidades especificadas no sistema.

Como já referido anteriormente estes módulos seguem uma interface bem definida pelo MGC que permite tratar todas as transformações de forma homogénea, independente do tipo de transformação que vai ser gerada, inerente também ao tipo de linguagem de programação e tipo de ficheiro. Estas iterações entre módulos e a interface podem ser representados como na Figura 28.

3.3 Sumário

Nesta secção, foi realizada uma análise onde foi possível identificar quais os requisitos funcionais e não funcionais que seriam necessários ser considerados no desenvolvimento do sistema por Model Driven Architecture (MDA). Feita a analise foi proposto um sistema que fosse de encontro ao especificado na análise e que fosse de permitisse a criação de um sistema de EC.

Vista a análise do sistema a ser desenvolvido passou-se para o desenvolvido do Motor de geração de Código (MGC) que juntamente com os seus módulos desenvolvidos em Java fosse possível a geração de sistemas EC.

Tendo em conta os requisitos identificados na fase de análise foram especificados quais os componentes necessários de uma forma geral, que seriam necessários para o desenvolvimento do sistema, assim como as relações e iterações entre os mesmos. Nesta fase de implementação foi apresentado o desenvolvimento do MGC considerando os requisitos e o sistema proposto inicialmente. É explicado o funcionamento do mesmo, assim como os módulos criados, para interagir com o MGC e ir de encontro às

necessidades especificadas. É ainda explicado o script usado para executar o MGC e sua forma de iteração com o utilizador.

Dentro do desenvolvimento do MGC é explicado quais os componentes chave para o seu funcionamento, a forma como estes se relacionam, como obedecem as especificações inferidas pelo MDA e de que forma é feita a ligação entre o MGC e os seus módulos, de forma a respeitarem regras comuns entre eles.

4 Validação do Mecanismo de Geração de Código

Após o processo de análise e implementação do MGC, considerou-se importante desenvolver alguns testes de desempenho e de análise dos artefactos produzidos através da plataforma desenvolvida e obter algumas conclusões sobre o aumento de produtividade. Assim, no presente capítulo será utilizado o esquema representado na Figura 29, como especificação para o sistema de Electronic Commerce (EC) a ser gerado. Serão representados os respetivos *inputs* para o MGC assim como os artefactos propostos.

4.1 Análise proposta do Protótipo

O sistema de para a criação de um protótipo foi identificado na Figura 29 como um modelo simples para o desenvolvimento de um sistema EC. Para o desenvolvimento de um protótipo que fosse de encontro a esse modelo e, tendo em conta tanto o MGC desenvolvido com os módulos a ele associado, foram assinaladas quais os componentes do sistema que podiam ser geradas pelo MGC.

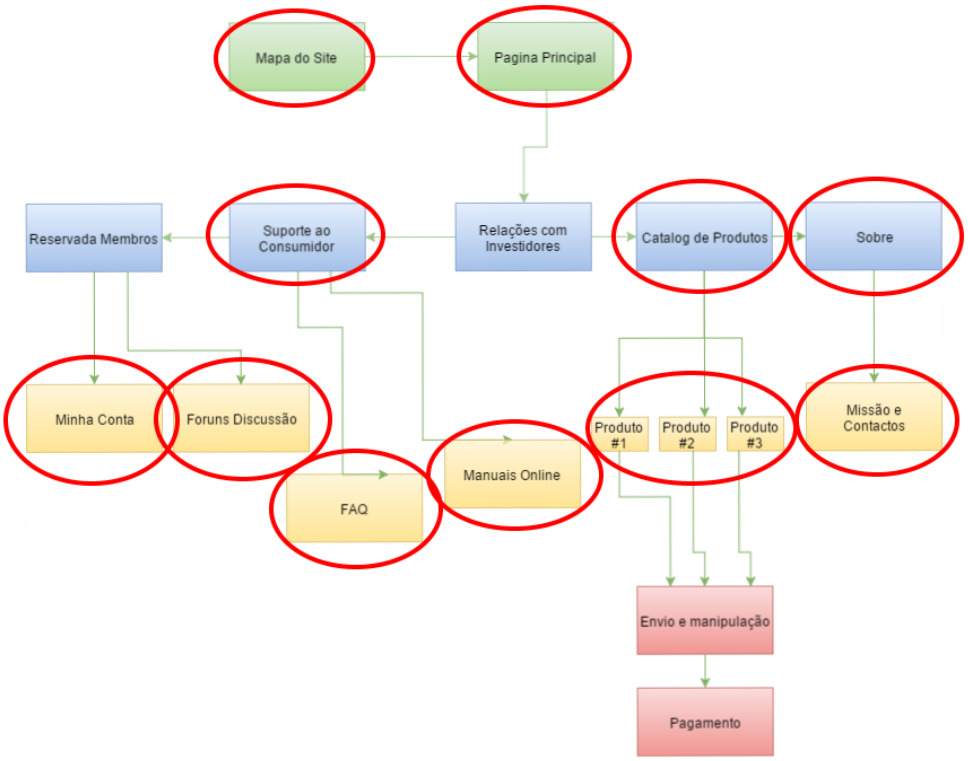

Figura 29 – Arquitetura do protótipo

Na Figura 29 estão representadas as várias entidades envolvidas no sistema a ser gerado. Para gerar o sistema é necessário identificar cada uma das entidades de forma isolada, assim como as suas características, sendo que com elas é possível criar o *input* necessário para o MGC gerar o sistema. Estão assinalas por uma círculo vermelho todas as entidades que podem ser geradas pelos módulos desenvolvidos utilizados pelo MGC.

Tabela 5 – Entidades e atributos protótipo

Nome tabela	**Campos**
Catalogo	Id_catalogo, Id_categoria, Nome, Validade
Categoria	Id_catalogo, Id_categoria, Nome, Validade
Produto	Id_catalogo, Id_produto, Preço, Quantidade, Validade, Sobre, Missão, Contactos, Página principal
Tópico	Id_forum, Id_topico, Nome, Fechado
Utilizador	Id_utilizador, Nome, Correio Eletrónico, Palavra-chave, Morada
FAQ	Id_faq, Questão, Resposta
Fórum	Nome, Id_forum
Manuais Online	Id_manual, TextoManual

Analisadas as entidades assinaladas foi possível identificar algumas características expectáveis de ser encontradas em cada uma delas. Na Tabela 5 é possível identificar de forma sintetizada todas as entidades e os respetivos atributos.

Identificados os componentes e entidades constituintes do sistema é possível passar a próxima fase do desenvolvimento do sistema, ou seja, a da conceção do protótipo.

4.2 Conceção do Protótipo

Nesta fase serão especificados quais os passos necessários para estruturar as entidades e componentes num ficheiro CIM, que como, foi visto é o ponto de entrada para o MGC analisar quais as entidades que estarão presentes no sistema.

Um ficheiro CIM é constituído por entidades e as respetivas transformações. Como tal, foi desenvolvido um CIM (Anexo 1), que contém as entidades e as transformações necessárias para a criação do protótipo.

As transformações que foram especificadas no CIM irão originar vários *outputs* de ficheiros, tais como *WebServices* Spring, *WebServices* em WCF, objetos base tanto em C# como em Java, código SQL para criação das tabelas em base de dados, código em Javascript para suportar os controladores para a *framework* AngularJS e páginas HTML.

De seguida é necessário usar o terminal do sistema operativo para utilizar o MGC e gerar o *ouput* para as entidades e transformações definidas. Para isso basta usar o comando "./run.sh –console" e utilizar o ficheiro CIM com o nome exemplo.xml. O processo de leitura, transformação e geração pode ser visualizado na Figura 30.

Ainda em relação à Figura 30 é importante salientar que o MGC demorou apenas **1 segundo** a processar as transformações**.** O tempo de geração é sem dúvida bastante baixo para a quantidade dos artefactos gerados. Claro que ao tempo de processamento do MGC de código é necessário adicionar o tempo de configuração do CIM. Este acaba por ser diluído na fase de análise do projeto, visto que pode usado como documentação do projeto, requerendo para isso que o projeto seja projetado desde o início com o desenvolvimento por MDA, para poder tirar esse benefício.

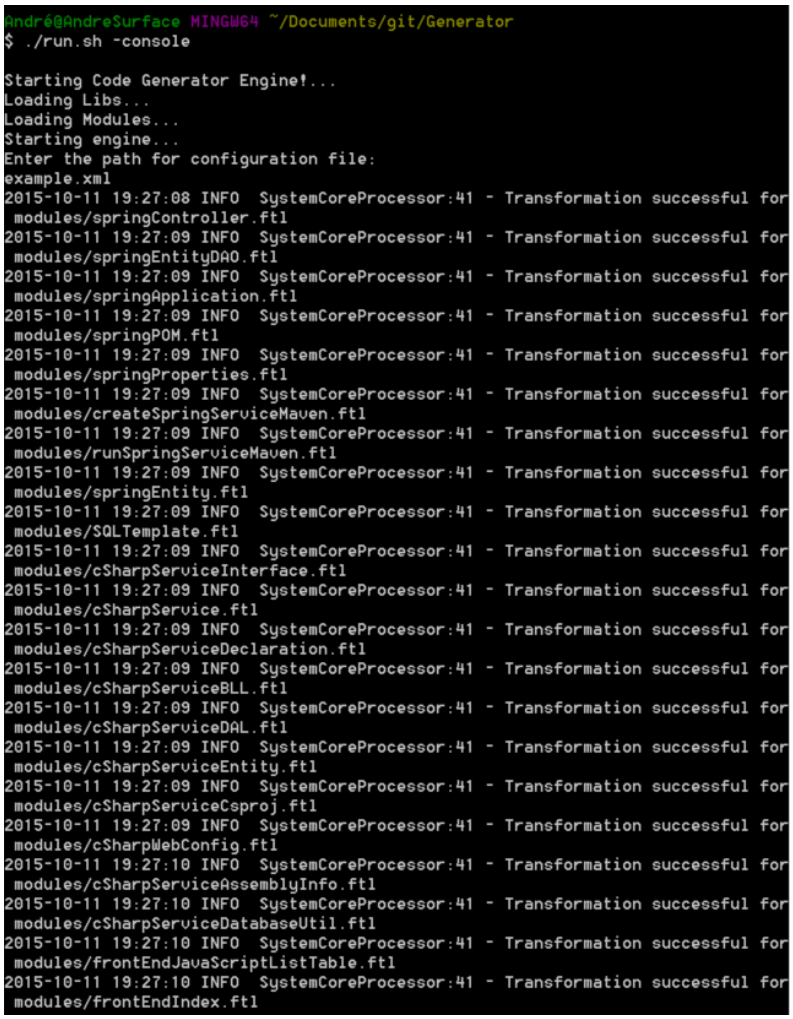

Figura 30 – Execução do MGC

4.3 Ficheiros do Protótipo gerados

Estando concluída a fase de geração dos *outputs,* é necessário usar os ficheiros no desenvolvimento do sistema EC. Em alguns casos pode passar por incluir apenas os ficheiros nas pastas do projeto. Noutros casos, como o dos *WebServices*, os ficheiros são gerados por completo sendo apenas necessário executar os serviços.

4.3.1 Spring WebService

No caso do *WebService* desenvolvido na *framework* Spring, os ficheiros necessários para a geração estão agrupados consoante o representado na Figura 31.

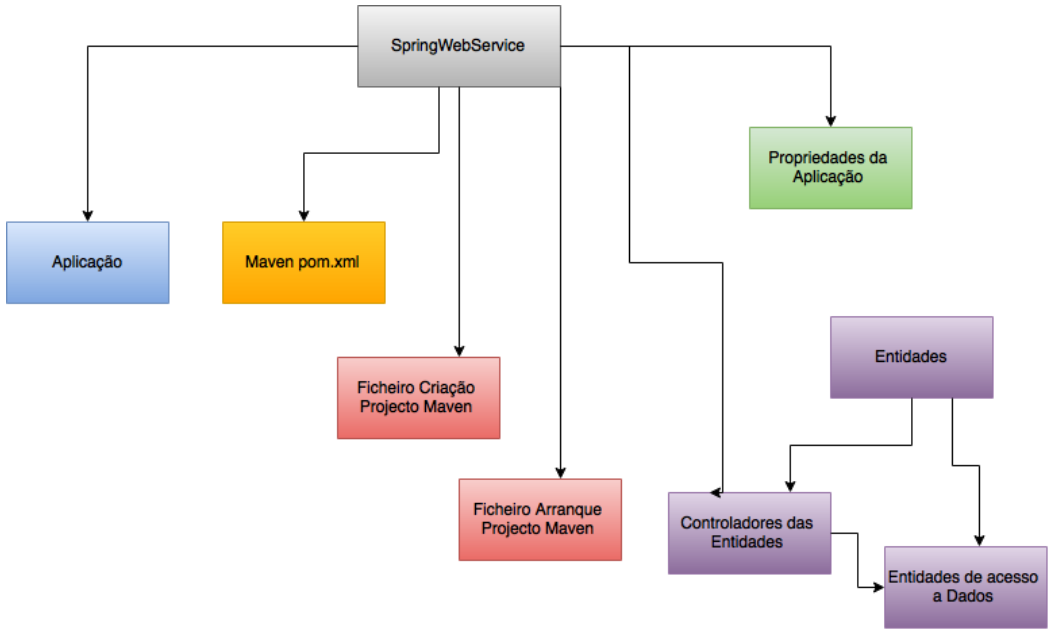

Figura 31 – Esquema de ficheiros – *webservice* Spring

O ficheiro de Aplicação é o ficheiro que serve de ponto de entrada na aplicação Spring. O ficheiro *Pom* juntamente com o ficheiro de *scripts* representados a vermelho são usados para criar, configurar e executar a aplicação. Os ficheiros representados a roxo fazem referência aos recursos necessários para mapear as entidades para objetos e as persistir em base de dados, através de mecanismos definidos pelo *hibernate*. Por fim, o ficheiro de propriedades representado pela cor verde contém as configurações de acesso à base de dados, assim como a forma como os objetos são mapeados para as tabelas pelo *hibernate.*

4.3.2 WCF WebService

No caso do *WebService* desenvolvido através da *framework* WCF (em C#), os ficheiros necessários para a geração estão agrupados consoante o representado na Figura 32.

O ficheiro de interface de serviço indica onde estão especificados os caminhos para os serviços, os parâmetros que aceitam ao serem executados e, por fim, o tipo de retorno. Os ficheiros representados a

roxo fazem referência aos recursos necessários para mapear as entidades para objetos e persisti-las em base de dados. No caso dos serviços WCF os dados não são mapeados para a base de dados diretamente dos objetos, mas usando uma classe utilitária definida a verde para persistir os dados.

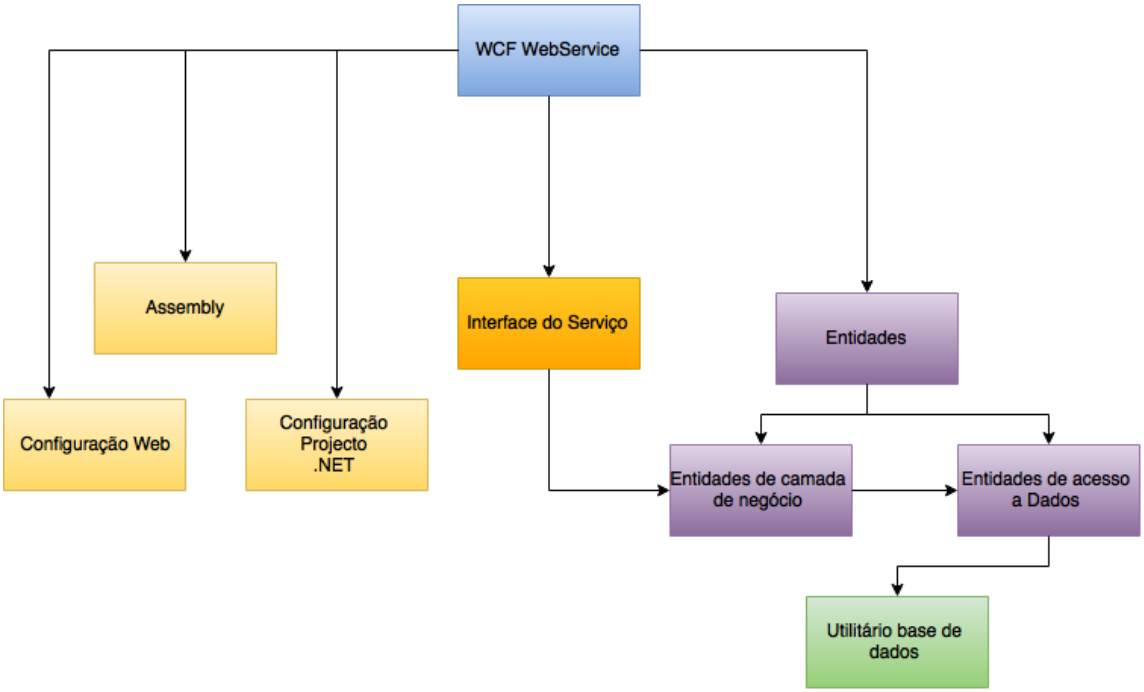

Figura 32 – Esquema de ficheiros – *webservice* WCF

4.3.3 FrontEnd

Os módulos de *FrontEnd* são responsáveis por gerar os artefactos que representam a camada à qual o utilizador terá acesso, pois todos ficheiros são carregados do lado do cliente ao tentar aceder ao EC. A página inicial representada a vermelho na Figura 33, será a página principal que utilizador terá acesso ao aceder ao EC. Por fim, os controladores representados pela cor roxa representam o ficheiro que contém todos os controladores do AngularJS, usados para carregar a informação disponibilizada pelos *WebServices.*

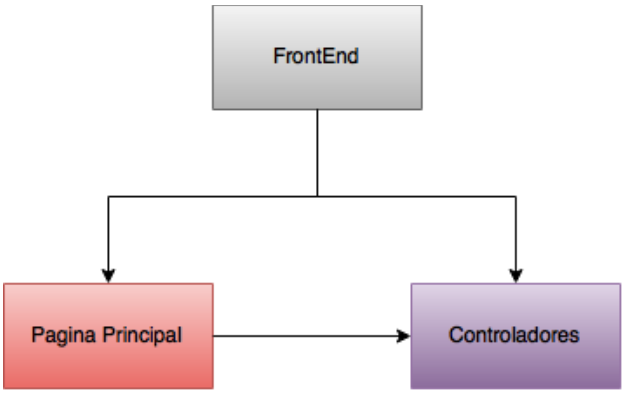

Figura 33 – Esquema de ficheiros do *frontend*

4.4 Resultado Obtido

Após gerar os ficheiros dos vários módulos do MGC para o protótipo descrito no anexo 1, torna-se importante referir algumas das vantagens e resultados obtidos através de todo o processo de transformações e todos os ficheiros descritos anteriormente.

Ao arrancar o serviço Spring, criado para este protótipo, é possível ver no painel de administração das bases de dados, a base de dados gerada automaticamente. A base de dados é gerada automaticamente seguindo as especificações definidas no ficheiro de propriedades disponível no serviço Spring, juntamente com as entidades especificadas no ficheiro CIM (Anexo 1) previamente criada pelo utilizador para o arranque do MGC. O resultado das entidades mapeadas para base de dados é visível na Figura 34.

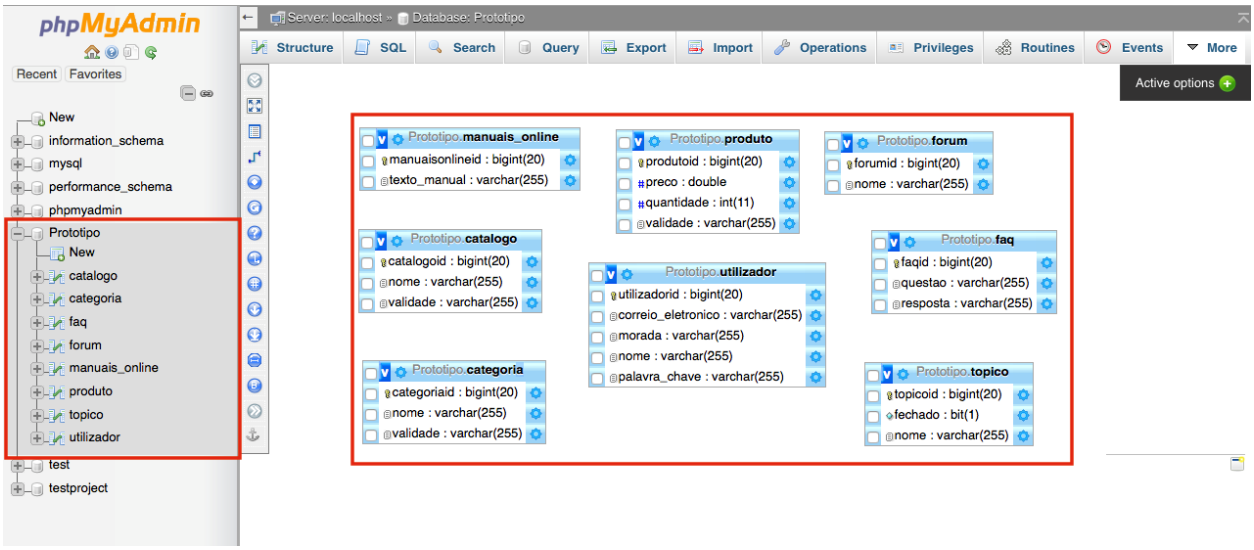

Figura 34 – Painel de administração – base de dados

De seguida apresentam-se alguns testes feitos para demonstrar o funcionamento dos *webservices* criados. Para os testes serão criados produtos, categorias para classificar esses produtos e um catálogo para agregar todos os produtos criados.

Os pedidos aos serviços para a criação dos produtos, categorias e o catálogo podem ser visualizados na tabela disponível no Anexo 2, onde é possível analisar todos os pedidos realizados para proceder a criação em base de dados do registo, assim como as chamadas feitas aos serviços responsáveis por listar todos os elementos das tabelas.

Através dos dados analisados no Anexo 2 é possível verificar que os serviços gerados automaticamente cumprem o objetivo pelo qual foram criados, contudo, os dados fornecidos por eles necessitam de ser mapeados de forma a poderem serem usados num sistema EC. Esse mapeamento passa pela criação de páginas na Internet no qual seja possível o utilizador interagir com o sistema de EC.

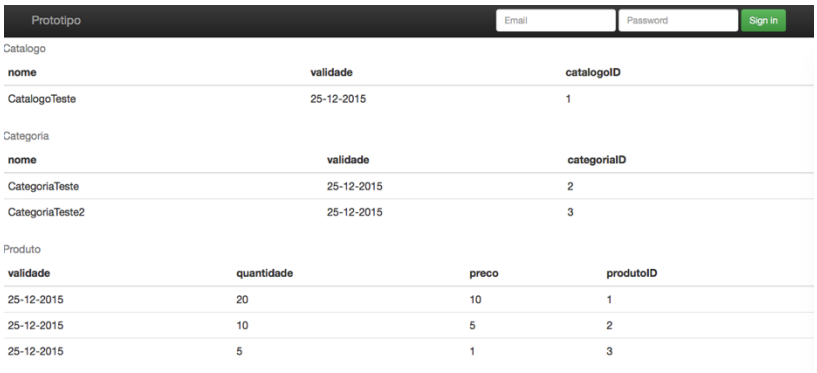

Figura 35 – Página principal gerada (computador)

O resultado desses ficheiros é possível visualizar tanto num navegador de páginas de Internet num computador (Figura 35), como num navegador de um telemóvel (Figura 36). Os dados previamente adicionados serão apresentados neste momento na página principal, futuramente a informação poderá ser usada para a criação do sistema EC final.

Figura 36 - Página principal gerada (mobile)

4.5 Atualização e Manutenção de *software*

Como já foi discutido nos capítulos acima, a manutenção e atualização de *software* representa o maior esforço, custo no desenvolvimento de *software*. No desenvolvimento de sistemas por MDA, padrões e boas regras de programação podem ser sempre mantidas assim como a coerência de assinaturas de métodos, nomes de variáveis, ou seja, todo o fluxo das aplicações é sempre mantido de forma homogenia.

Estas particularidades do ciclo de desenvolvimento por MDA, tornam-na na sua maior vantagem quando é necessário fazer manutenção e atualização de *software*. Quando a coerência é mantida ao longo de todas as aplicações fica fácil para o programador identificar, como e onde proceder as alterações. Esta coerência permite ainda ao programador ter com um elevado nível de certeza que as suas alterações não vão apenas funcionar no local onde está alterar mais sim em todas as aplicações que usarem o mesmo mecanismo de MDA.

Utilizado o protótipo especificado no subcapítulo 4.4, será projetado o caso prático de ser necessário proceder há sua alteração de forma a alterar os serviços para que tenham que ser invocados utilizando um utilizador e uma palavra-chave para autenticação em *Lightwight Direcctory Access Protocol* (LDAP).

Para realizar esta alteração seriam apenas necessários duas alterações, o primeiro passo seria acrescentar o utilizador e a palavra-chave nos métodos dos serviços, o segundo passo será acrescentar o método que efetivamente iria ser responsável pela autenticação no servidor LDAP.

Na classe de utilitários dos webservices Spring, ao qual é produzida através de uma transformação pelo MGC já definida previamente no módulo de Spring, bastaria acrescentar os métodos que serão chamados para realizar a ligação e validação no servidor LDAP.

Um excerto desse método pode ser visualizado no Código Fonte 7 o restante código pode ser consultado no Anexo 3.

```
public tryLoginLDAP(String user,String password) throws Exception {

      String url = "ldaps://localhost:389/${systemName}"; // ldap url
              String domainName = params.get("${systemName}"); // system domain

         if (domainName==null || "".equals(domainName)) {
      …

      private static String toDC(String domainName) {
              StringBuilder buf = new StringBuilder();
              for (String token : domainName.split("\\.")) {
      …
```

Código Fonte 7 - Excerto de código para ligação LDAP

```
      /**
         * /create  --> Create a new ${className?uncap_first} and save it in
      the database.
         *
         *<#list attributes as attribute>
          <#if attribute.isKey == false>* @param ${attribute.name} ${className}
      ${attribute.name}</#if>
          </#list>
         * @return A string describing if the ${className?uncap_first} is
      succesfully created or not.
         */
        @RequestMapping("${className?uncap_first}/create")
        @ResponseBody
        public String create(<#list attributes as attribute><#if
      attribute.isKey == false>${attribute.type} ${attribute.name}<#if
      attribute_has_next><#if attributes[attribute_index +1].isKey ==
      false>,</#if></#if></#if></#list>,String user,String password){
          tryLoginLDAP(user,password);
          ${className} ${className?uncap_first} = null;
          try {
            ${className?uncap_first} = new ${className}(<#list attributes as
      attribute><#if attribute.isKey == false>${attribute.name}<#if
      attribute_has_next><#if attributes[attribute_index +1].isKey ==
      false>,</#if></#if></#if></#list>);
            ${className?uncap_first}Dao.save(${className?uncap_first});
          }
          catch (Exception ex) {
            return "Error creating the ${className?uncap_first}: " +
      ex.toString();
          }
          return "${className} succesfully created! (id = " +
      ${className?uncap_first}.get<#list attributes as attribute><#if
      attribute.isKey == true>${attribute.name}</#if></#list>() + ")";
        }
```

Código Fonte 8 - Excerto de código do template do serviço

Visto o Código Fonte 7 para a utilização da autenticação em servidor LDAP, fica apenas a faltar a alteração do modelo já existente nos serviços Spring, responsável por disponibilizar os endereços de acesso do serviço.

As alterações necessárias no modelo passam como já referido em cima por acrescentar o utilizador e a palavra-chave como parâmetros necessários a serem enviados pelo programador ao executar a chamada ao serviço. Essas alterações podem ser visualizadas no Código Fonte 8. O código apresentado faz apenas referencia as alterações no método de adicionar relacionado para cada uma das entidades do sistema, as alterações dos restantes métodos pode ser consultada no Anexo 4.

Neste ponto é possível notar os benefícios do desenvolvimento do sistema por MDA, com umas simples alterações foi possível acrescentar um mecanismo de validação por LDAP nos serviços. As vantagens não estão só na facilidade de adicionar novos comportamentos no sistema, mas na coerência que estas alterações têm sempre perante todo o sistema. A adição em apenas um sitio irá permitir que os mecanismos de autenticação sejam verificados em todas as chamadas dos serviços criados no projeto protótipo e ainda em casos que existam outros projetos previamente desenvolvidos, atualizar os mesmos e manter coerência entre projetos.

A melhoria manutenção do código pode ser facilmente comprovada pelo exemplo apresentado neste subcapítulo, contudo nesta dissertação e como era espectável, não foi possível quantificar em termos temporais a redução que o MGC pode trazer nesta fase do ciclo de desenvolvimento de *software*. Mais testes e questionários com utilizadores a experimentarem o MGC seria necessário para monitorizar e analisar dados que fossem possível gerar métricas temporais.

4.6 Sumário

Este capitulo tem como funcionalidade explicar como o Motor de Geração de Código (MGC) contribui efetivamente no desenvolvimento de *software*, mais propriamente no desenvolvimento de sistemas de Electronic Commerce (EC). Para isso através de um esquema de componentes de um sistema EC foi possível selecionar quais os que seriam possíveis de serem gerados através do MGC.

Após escolhidos quais os componentes do sistema que podiam ser gerados pelo MGC, foi realizada demonstração dos passos necessários para o desenvolvimento e manutenção desses mesmos componentes.

Foi especificado o Computer Independent Model (CIM) com as especificações das entidades do sistema identificadas e com as transformações necessárias para desenvolver o sistema de EC de acordo com componentes identificados.

Após ser criado o ficheiro CIM foi utilizado o MGC para proceder a geração do código do sistema. Esse código iria permitir a criação de um sistema EC tanto através da linguagem programação Java com da linguagem de programação C#. Foram especificados os ficheiros de cada uma das plataformas e como estes se encontram agrupados.

Foram executados os respetivos componentes gerados de forma a ver o resultado obtido pelo MGC. Dessa execução foi possível ver a respetiva base de dados criada automaticamente de acordo com as entidades especificadas no ficheiro CIM. Foi possível analisar os serviços disponibilizados pelos serviços gerados que possibilitam a gestão das mesmas entidades definidas no CIM. Por fim foi possível visualizar num navegado o resultado sobre a forma de uma página de internet onde os dados podiam ser consultados.

Por fim após mostrar o sistema gerado foi introduzido um novo caso de implementação para evidenciar o maior benefício do MGC e respetivamente do MDA, ou seja, a redução de tempo e trabalho na manutenção de *software*. Para isso foi realizada a alteração em todos os serviços disponibilizados pelos webservices de forma a necessitassem de autenticação *Lightwight Direcctory Access Protocol* (LDAP) para validar a respetiva autorização na execução dos pedidos aos webservices.

5 Conclusão e Trabalho Futuro

Através do estudo realizado no âmbito desta dissertação foi possível analisar a evolução sistemas de Electronic Commerce (EC). Nas últimas décadas estes sistemas têm vindo a aumentar e a proporcionar cada vez mais funcionalidades mais completas e complexas. Estas alterações acontecem porque no mundo das TI mudanças acontecem todos os dias, contribuindo para um desenvolvimento continuo nas mais diversas áreas das TI e nos sistemas EC.

Após analisar as características tanto do desenvolvimento típico de *software* como do desenvolvimento de *software* orientado por modelos proposto pelo MDA, foi possível identificar várias diferenças. Estas traduziram-se na forma como o MDA estrutura o desenvolvimento de *software*, permitindo tirar vantagens nas fases de criação e manutenção/atualização.

O desenvolvimento de um sistema por MDA proporciona que este seja aberto a novas atualizações de forma facilitada. Tendo em conta o crescimento do mercado dos sistemas EC e das tecnologias no geral, é uma característica importante pois o custo de os desenvolver é elevado e a fácil atualização é uma valorização do investimento feito neste tipo de sistemas. A portabilidade e qualidade dos Plataform Independent Models (PIM) permite que os sistemas desenvolvidos por MDA possam ser facilmente migrados para novos ambientes e plataformas que, tendo em conta a evolução das tecnologias, pode ser cada vez mais uma realidade. A interoperabilidade é o ponto-chave dos sistemas MDA e, no caso de sistemas EC também

constitui uma vantagem, visto que cada vez mais comunicam com outros sistemas externos de forma a enriquecer o conteúdo para o utilizador final.

Tendo em conta as características definidas pelo MDA na reutilização de código, através da respetiva padronização de código e de processos repetitivos no desenvolvimento de *software,* verificou-se a grande vantagem da sua aplicabilidade para sistemas de EC. Os sistemas de EC encontram-se em expansão, portanto, sistemas como MDA podem ser uma mais-valia na criação do que podem ser os sistemas de EC do futuro.

Foi criado um Motor de Geração de Código (MGC) orientado pelas metodologias inferidas pelo MDA para criar um sistema expansível e robusto que permitisse a geração de artefactos de *software* e, por sua vez, auxiliar na criação de novos sistemas de EC e atualização dos antigos.

O MGC permite a adição e atualização por módulos de *software,* por forma a contemplar futuras implementações necessárias e para poder alcançar a abrangência, tanto na quantidade, como na qualidade de artefactos que podem ser gerados. Foram desenvolvidos módulos juntamento com o MGC para serem utilizados por uma equipa de programadores no desenvolvimento de um sistema de EC. Assim, através destes módulos e do MGC é possível começar o desenvolvimento do sistema com um *kickstart,* sendo que as suas vantagens vão-se notar no tempo total de despendido pela equipa, visto que será menor que fazer o desenvolvimento todo de raiz. Outras vantagens estão presentes de forma intrínseca no MDA, visto que a grande parte do desenvolvimento fica do lado de código gerado, este pode seguir sem boas práticas no desenvolvimento de *software*, assim como manter sempre uma coerência na forma como o desenvolvimento é realizado e os nomes atribuídos nos componentes de software. Isto pode não parecer uma grande vantagem, mas quando este tipo de sistema aplicado no desenvolvimento de uma grande aplicação onde a equipa é grande, os recursos humanos presentes nas equipas vão deixando a empresa, sendo necessária a contratação de novos que entram em fases de desenvolvimento avançado, onde este código padronizado servira também para que estes entendam qual a forma correta de fazer o desenvolvimento de *software*, continuando assim o código a manter a sua qualidade e organização.

Os módulos desenvolvidos no âmbito da dissertação foram meramente para provar o potencial do MDA no desenvolvimento de sistemas *ecommerce*. Contudo, para desenvolver sistemas mais complexos e como trabalho futuro, serão sempre necessários serem desenvolvidos novos

módulos de forma a poder desenvolver as restantes funcionalidades espectáveis para um sistema de EC. Por exemplo diferentes sistemas de pagamento, gestão de contas de utilizador e suas respetivas preferências e atividades, entre muitas outras.

Como trabalho futuro ficam em aberto várias possibilidades, uma delas é o desenvolvimento mecanismo de interação com o utilizador de forma de aplicação *desktop* ou *online* para que a parte de análise e estruturação do CIM do MGC possa ser desenvolvido de uma forma mais apelativa para o utilizador. A nível de funcionalidades, fator pouco explorado nesta dissertação criar módulos para geração de código de *frontend* de forma a poder criar páginas de internet mais informativas para o utilizador, assim como a possibilidade de criar o painel de administração para os utilizadores típico em sistemas de *ecommerce* mais avançados. Na parte de *backend* fica a possibilidade de criação de módulos como geração de lógica de negócio, a geração do processo de carrinho de compras, o modulo para vários sistemas de pagamento, entre muitos outros.

Por fim é importante constatar que os objetivos propostos para esta dissertação foram cumpridos, onde foi possível analisar os sistemas de retalho na sua definição e evolução ao longo dos séculos. Foram analisados os vários tipos de *ecommerce* existentes, assim como a evolução ao longo dos anos a nível de impacto no mercado. Foi analisado o ciclo de desenvolvimento de *software* por MDA, a sua visão, as suas vantagens relativamente ao típico ciclo de desenvolvimento de software e de que forma este pode contribuir para os sistemas EC. Foi analisado e proposto um sistema que fosse capaz de trazer benefícios para o desenvolvimento de sistemas EC e que fosse de encontro as metodologias do MDA. Após o desenvolvimento do sistema especificado foram realizados testes no mesmo, onde foi possível comprovar os benefícios do sistema criado para o auxílio na criação de sistemas EC.

Concluindo, penso que a realização da presente dissertação promoveu a aplicação de conhecimentos adquiridos no âmbito das várias unidades curriculares frequentadas, conhecimentos numa área de investimento pessoal devido à sua importância no futuro do desenvolvimento de *software*. Houve ainda o desenvolvimento de competências no domínio da pesquisa e escrita, que resultaram num significativo enriquecimento de competências profissionais.

Referências

[1] Statista, “Number of Internet Users Worldwide,” 2015. [Online]. Available: http://www.statista.com/statistics/273018/number-of-internet-users-worldwide/. [Acedido em Setembro 2015].

[2] Statista, “Number of digital buyers worldwide from 2011 to 2016 (in millions),” 2015. [Online]. Available: http://www.statista.com/statistics/251666/number-of-digital-buyers-worldwide/. [Acedido em Setembro 2015].

[3] A. Silva e C. Videira, UML Metologias e Ferramentas CASE, 2ª Edição ed., Centro Atlântico, Lda, 2005.

[4] Object Management Group, “Object Management Group,” 2004. [Online]. Available: http://www.omg.org/. [Acedido em Março 2015].

[5] Object Management Group, “MDA - The Architecture of Choice for a Changing World,” 2004. [Online]. Available: http://www.omg.org/mda/. [Acedido em Março 2015].

[6] D. Roth, “History of Retail in 100 0bjects,” 2013. [Online]. Available: http://www.retail100objects.com/History-of-retail-in-100-objects.pdf. [Acedido em Setembro 2015].

[7] LEO, “Leo Computers Society,” [Online]. Available: http://www.leo-computers.org.uk/. [Acedido em Outubro 2015].

[8] Walmart, “Walmart.com: Save Money. Live Better,” 1962. [Online]. Available: http://www.walmart.com/. [Acedido em Setembro 2015].

[9] Kmart, “Kmart - Deals on Furniture, Toys, Clothes, Tools, Tablets ...,” 1977. [Online]. Available: http://www.kmart.com/. [Acedido em Setembro 2015].

[10] Target, “Target: Expect More. Pay Less.,” 1902. [Online]. Available: http://www.target.com/. [Acedido em Setembro 2015].

[11] Mastercard, “History of the Card Payments System,” [Online]. Available: http://www.mastercard.com/us/company/en/docs/history%20of%20payments.pdf. [Acedido em Outubro 2015].

[12] eBay, “eBay: Electronics, Cars, Fashion, Collectibles, Coupons and ...,” 1995. [Online]. Available: http://www.ebay.com/. [Acedido em Setembro 2015].

[13] Amazon, “Amazon.com: Books,” 1994. [Online]. Available: http://www.amazon.com. [Acedido em Setembro 2015].

[14] Ecommerce-Land, “History of ECommerce,” 2004. [Online]. Available: http://www.ecommerce-land.com/history_ecommerce.html.

[15] eMarketer, “eMarketer: Digital Marketing Research & Insights,” 1996. [Online]. Available: http://www.emarketer.com/. [Acedido em Setembro 2015].

[16] eMarketer, “Retail Sales Worldwide Will Top $22 Trillion This Year,” 2014. [Online]. Available: http://www.emarketer.com/Article/Retail-Sales-Worldwide-Will-Top-22-Trillion-This-Year/1011765. [Acedido em Setembro 2015].

[17] ACEPI, “Principais Indicadores do Estudo em relação a Portugal,” 2012. [Online]. Available: http://www.computerworld.com.pt/media/2012/07/ACEPI-Estudo-Media-Scope-10_07_12.pdf. [Acedido em Setembro 2015].

[18] European ECommerce, “European B2C E-commerce Report 2015,” 2015. [Online]. Available: http://nrw.nl/wp-content/uploads/2015/07/European-b2c-e-commerce-report-2015.pdf. [Acedido em Setembro 2015].

[19] E. Turban, D. King, J. K. Lee, T.-P. Liang e D. Turban, “Electronic Commerce: Definitions and Concepts,” em *Electronic Commerce: A Managerial and Social Networks Perspective*, 2015, pp. 7, 15,16.

[20] H. J. Wen, H.-G. Chen e H.-G. Hwang, “E-commerce Web site design: strategies and models,” em *Information Management & Computer Security*, Emerald Group Publishing Limited, 2001, pp. 5-12.

[21] I. Lee, “Introduction to E-Commerce in the Global Economy,” em *Electronic Commerce Management for Business Activities and Global Enterprises*, IGI Global, 2012, pp. 15-19.

[22] HP, “HP® Official Site | Laptop Computers, Desktops, Printers,” 1939. [Online]. Available: http://www8.hp.com/pt/pt/home.html. [Acedido em Setembro 2015].

[23] John Deere, “John Deere Início,” 1837. [Online]. Available: https://www.deere.pt/. [Acedido em 2015 Setembro].

[24] Home Depot, “Home Depot,” 1978. [Online]. Available: www.homedepot.com/. [Acedido em Setembro 2015].

[25] Dell, “Dell Official Site - The Power To Do More | Dell,” 1984. [Online]. Available: www.dell.com/. [Acedido em Setembro 2015].

[26] Best Buy, “Best Buy: Expert Service. Unbeatable Price.,” 1966. [Online]. Available: www.bestbuy.com/. [Acedido em Setembro 2015].

[27] Fotolia, “Fotolia - Venda e compra de imagens, vetores e vídeos ...,” 2005. [Online]. Available: https://pt.fotolia.com/. [Acedido em Setembro 2015].

[28] Survey Scout, “Survey Scout >> Where We Pay for Your Opinion!,” [Online]. Available: www.surveyscout.com/. [Acedido em Setembro 2015].

[29] Craigslist, “craigslist: Lisboa empregos, apartamentos, anúncios ...,” [Online]. Available: craigslist.pt. [Acedido em Setembro 2015].

[30] PayPal, “PayPal Portugal,” [Online]. Available: https://www.paypal.com/pt/home. [Acedido em Setembro 2015].

[31] J.-P. Rodrigues, “Logistics and Freight Distribution,” em *The Geografy of Translport Systems*, 2013.

[32] E. Turban, D. King, J. K. Lee, T.-P. Liang e D. Turban, “Retailing in Electronic Commerce: Products and Services,” em *Electronic Commerce A Managerial and Social Networks Perspective*, 2015, pp. 110-113, 325.

[33] QVC, “Shopping Online at Home is Easy with QVC — Official Site,” 1986. [Online]. Available: www.qvc.com/. [Acedido em Setembro 2015].

[34] Lands' End, “Lands' End | Sweaters, Coats, Jeans, Dress Shirts & More,” 1963. [Online]. Available: http://www.landsend.com/. [Acedido em Setembro 2015].

[35] Staples, “Staples,” [Online]. Available: www.staples.pt/. [Acedido em Setembro 2015].

[36] BedandBreakfast, “BedandBreakfast.com: Top B&Bs, Inns, & Romantic Hotels,” [Online]. Available: www.bedandbreakfast.com/. [Acedido em Setembro 2015].

[37] Yahoo , “Yahoo Small Business: Web Hosting, Domains, Ecommerce ...,” [Online]. Available: https://smallbusiness.yahoo.com/. [Acedido em Setembro 2015].

[38] BingShop, “Bing Shop Paintball, Airsoft Park, and Skatboarding ...,” [Online]. Available: www.bingshop.com. [Acedido em Setembro 2015].

[39] Woot.com, “Woot: Daily Deals for Electronics, Computers, Home, Tools ...,” [Online]. Available: www.woot.com/. [Acedido em Setembro 2015].

[40] E. Turban, D. King, J. K. Lee, T.-P. Liang e D. Turban, “Implementing EC Systems: From Justification to Successful Performance,” em *Electronic Commerce A Managerial and Social Networks Perspective*, 2015, pp. 667-669.

[41] I. Lee, “Systems Development, Technologies, and Issues in E-Commerce,” em *Electronic Commerce Management for Business Activities and Global Enterprises*, IGI Global, 2012, pp. 407-412.

[42] Microsoft, “Home : The Official Microsoft IIS Site,” [Online]. Available: https://www.iis.net/. [Acedido em Setembro 2015].

[43] The Apache Software Foundation, “Welcome! - The Apache HTTP Server Project,” [Online]. Available: http://httpd.apache.org/. [Acedido em Setembro 2015].

[44] Red Hat, Inc, “JBoss Application Server - JBoss Community,” [Online]. Available: http://wildfly.org/. [Acedido em Setembro 2015].

[45] IBM, “WebSphere - IBM,” [Online]. Available: http://www.ibm.com/software/websphere. [Acedido em Setembro 2015].

[46] Microsoft, “Windows Server 2012 R2 overview | Microsoft,” [Online]. Available: http://www.microsoft.com/en-us/server-cloud/products/windows-server-2012-r2/. [Acedido em Setembro 2015].

[47] Oracle, “Application Server - Oracle WebLogic Server | Oracle,” [Online]. Available: http://www.oracle.com/us/products/middleware/cloud-app-foundation/weblogic/overview/index.html. [Acedido em Setembro 2015].

[48] IBM, “IBM - WebSphere Application Server,” [Online]. Available: http://www-03.ibm.com/software/products/pt/appserv-was. [Acedido em Setembro 2015].

[49] Microsoft, “SQL Server 2014 | Microsoft,” [Online]. Available: http://www.microsoft.com/pt-pt/server-cloud/products/sql-server/. [Acedido em Setembro 2015].

[50] MySQL, “MySQL :: The world's most popular open source database,” [Online]. Available: https://www.mysql.com/. [Acedido em Setembro 2015].

[51] Oracle, “Database 12c | Oracle,” [Online]. Available: https://www.oracle.com/database/index.html. [Acedido em Setembro 2015].

[52] PostgreSQL Global Development Group, “PostgreSQL: The world's most advanced open source database,” [Online]. Available: http://www.postgresql.org/. [Acedido em Setembro 2015].

[53] E. Turban, D. King, J. K. Lee, T.-P. Liang e D. Turban, “Implementing EC Systems: From Justifi cation to Successful Performance,” em *Electronic Commerce A Managerial and Social Networks Perspective*, 2015, pp. 670-672.

[54] IBM, “IBM - Portugal,” [Online]. Available: http://www.ibm.com/pt/pt/. [Acedido em Setembro 2015].

[55] Oracle, “Oracle Portugal | Integrated Cloud Applications and Platform Services,” [Online]. Available: https://www.oracle.com/pt/index.html. [Acedido em Setembro 2015].

[56] MCI, “Home | MCI Global | Building Community,” [Online]. Available: http://www.mci-group.com/. [Acedido em Setembro 2015].

[57] Cephas Consulting Corp, em *The Fast Guide to Model Driven Architecture: The Basics of Model Driven Architecture*, 2006, pp. 3-5.

[58] OMG, “Introduction to MDA,” em *MDA Guide Version 1.0.1*, 2003, pp. 1-10.

[59] A. Kleppe, W. Bast e J. Warmer, “The MDA Framework,” em *MDA Explained: The Model Driven Architecture™: Practice and Promise*, Addison-Wesley Professional, 2003.

[60] A. Kleppe, W. Bast e J. Warmer, “The MDA Development Process,” em *MDA Explained: The Model Driven Architecture™: Practice and Promise*, Addison-Wesley Professional, 2003.

[61] A. Silva e C. Videira, “Enquadramento e Conceitos Gerais,” em *UML Metodologias e Ferramentas CASE*, CentroAtlântico, 2005, pp. 35,36.

[62] D. Wells, “Extreme Programming: A gentle introduction,” [Online]. Available: http://www.extremeprogramming.org/. [Acedido em Steembro 2015].

[63] J. Cabot, “Relationship between MDA,MDD and MDE,” 2009. [Online]. Available: http://modeling-languages.com/relationship-between-mdamdd-and-mde/. [Acedido em Setembro 2015].

[64] O. Martínez, J. Lovelle, V. Díaz e B. Gar, “Model-Driven Engineering for Electronic Commerce,” em *Progressions and Innovations in Model-Driven Software Engineering*, IGI Global, 2013.

[65] XDoclet , “XDoclet: Attribute-Oriented Programming,” [Online]. Available: http://xdoclet.sourceforge.net/xdoclet/index.html. [Acedido em Setembro 2015].

[66] Maven, “What is AndroMDA?,” 2014. [Online]. Available: http://www.andromda.org/whatisit.html. [Acedido em Setembro 2015].

[67] InferData, “Model Component Compiler,” 2005. [Online]. Available: http://www.inferdata.com/products/modelcomponentcompiler.html.

[68] Freemaker Project, “FreeMarker Java Template Engine,” [Online]. Available: http://freemarker.org. [Acedido em Setembro 2015].

[69] Oracle, “Learn About Java Technology,” [Online]. Available: https://www.java.com/en/about/. [Acedido em Setembro 2015].

[70] Microsoft, “Introduction to the C# Language and the .NET Framework,” 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx. [Acedido em Setembro 2015].

[71] The Apache Software Foundation, “Apache Maven Project,” 2002. [Online]. Available: https://maven.apache.org/what-is-maven.html.

[72] Spring, “Introduction to the Spring Framework,” [Online]. Available: http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html. [Acedido em Setembro 2015].

[73] Microsoft, “What Is Windows Communication Foundation,” [Online]. Available: https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx. [Acedido em Setembro 2015].

[74] W3C, “Extensible Markup Language (XML),” [Online]. Available: http://www.w3.org/XML/. [Acedido em Setembro 2015].

[75] Google, “AngularJS,” [Online]. Available: https://docs.angularjs.org/misc/faq. [Acedido em Outubro 2015].

[76] Microsoft, “Model-View-Controller,” [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649643.aspx. [Acedido em Agosto 2015].

[77] IBM, “Model Transformation with the IBM Model Transformation,” 2015. [Online]. Available: http://www.ibm.com/developerworks/rational/library/05/503_sebas/index.html. [Acedido em Setembro 2015].

6 Anexos

6.1 Anexo 1

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
To change this license header, choose License Headers in Project
Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->

<!--
Data types avaliable : string,int,double,boolean,other entities

-->
<system name="Prototipo">
    <entities>
        <entity name="Catalogo">
            <attribute type="String" name="nome" modifier="private"
length="30"></attribute>
            <attribute type="String" name="validade"
modifier="private"></attribute>
            <relationship type="1-n" with="Categoria"></relationship>
        </entity>
        <entity name="Categoria">
            <attribute type="String" name="nome"
modifier="private"></attribute>
            <attribute type="String" name="validade"
modifier="private"></attribute>
        </entity>
        <entity name="Produto">
            <attribute type="String" name="validade"
modifier="private"></attribute>
            <attribute type="int" name="quantidade"
modifier="private"></attribute>
            <attribute type="double" name="preco"
modifier="private"></attribute>
        </entity>
        <entity name="Utilizador">
            <attribute type="String" name="correioEletronico"
modifier="private"></attribute>
            <attribute type="String" name="nome"
modifier="private"></attribute>
            <attribute type="String" name="palavraChave"
modifier="private"></attribute>
            <attribute type="String" name="morada"
modifier="private"></attribute>
        </entity>
         <entity name="Forum">
            <attribute type="String" name="nome"
modifier="private"></attribute>
        </entity>
        <entity name="Topico">
            <attribute type="String" name="nome"
modifier="private"></attribute>
            <attribute type="boolean" name="fechado"
modifier="private"></attribute>
        </entity>
        <entity name="FAQ">
```

```
            <attribute type="String" name="questao"
modifier="private"></attribute>
            <attribute type="String" name="resposta"
modifier="private"></attribute>
        </entity>
        <entity name="ManuaisOnline">
            <attribute type="String" name="textoManual"
modifier="private"></attribute>
        </entity>
    </entities>
     <transformations>
          <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/springController.ftl</template>

<drive>code.codegenerationengine.springwebservices.SpringControllerDriver</
drive>
            <outputType>.java</outputType>
            <outputRange>1-1</outputRange>
        </transformation>
         <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/springEntityDAO.ftl</template>

<drive>code.codegenerationengine.springwebservices.SpringEntityDAODriver</d
rive>
            <outputType>.java</outputType>
            <outputRange>1-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/springApplication.ftl</template>

<drive>code.codegenerationengine.springwebservices.ApplicationDriver</drive
>
            <outputType>.java</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/springPOM.ftl</template>

<drive>code.codegenerationengine.springwebservices.PomDriver</drive>
            <outputType>.xml</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/springProperties.ftl</template>

<drive>code.codegenerationengine.springwebservices.ApplicationPropertiesDri
ver</drive>
            <outputType>.properties</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/createSpringServiceMaven.ftl</template>
```

```
<drive>code.codegenerationengine.springwebservices.SpringCreateMavenProject
Driver</drive>
            <outputType>.sh</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/runSpringServiceMaven.ftl</template>

<drive>code.codegenerationengine.springwebservices.RunSpringApplicationDriv
er</drive>
            <outputType>.sh</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/springEntity.ftl</template>

<drive>code.codegenerationengine.springwebservices.SpringEntityDriver</driv
e>
            <outputType>.java</outputType>
            <outputRange>1-1</outputRange>
        </transformation>
     <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/SQLTemplate.ftl</template>
            <drive>code.codegenerationengine.sqltemplate.SQLDriver</drive>
            <outputType>.sql</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
         <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/cSharpServiceInterface.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceInterfaceDr
iver</drive>
            <outputType>.cs</outputType>
            <outputRange>1-1</outputRange>
         </transformation>
         <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/cSharpService.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceDriver</dri
ve>
            <outputType>.svc.cs</outputType>
            <outputRange>1-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/cSharpServiceDeclaration.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceDeclaration
Driver</drive>
            <outputType>.svc</outputType>
            <outputRange>1-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
```

```
        <template>modules/cSharpServiceBLL.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceBLLDriver</
drive>
        <outputType>.cs</outputType>
        <outputRange>1-1</outputRange>
    </transformation>
    <transformation>
        <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
        <template>modules/cSharpServiceDAL.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceDALDriver</
drive>
        <outputType>.cs</outputType>
        <outputRange>1-1</outputRange>
    </transformation>
    <transformation>
        <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
        <template>modules/cSharpServiceEntity.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpWebServiceEntity</
drive>
        <outputType>.cs</outputType>
        <outputRange>1-1</outputRange>
    </transformation>
    <transformation>
        <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
        <template>modules/cSharpServiceCsproj.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceCsproj</dri
ve>
        <outputType>.csproj</outputType>
        <outputRange>n-1</outputRange>
    </transformation>
     <transformation>
        <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
        <template>modules/cSharpWebConfig.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceWebConfig</
drive>
        <outputType>.config</outputType>
        <outputRange>n-1</outputRange>
    </transformation>
    <transformation>
        <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
        <template>modules/cSharpServiceAssemblyInfo.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceAssemblyInf
o</drive>
        <outputType>.cs</outputType>
        <outputRange>n-1</outputRange>
    </transformation>
     <transformation>
        <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
        <template>modules/cSharpServiceDatabaseUtil.ftl</template>

<drive>code.codegenerationengine.csharpwebservices.CSharpServiceDataBaseUti
ls</drive>
        <outputType>.cs</outputType>
```

```
            <outputRange>n-1</outputRange>
        </transformation>
         <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/frontEndJavaScriptListTable.ftl</template>

<drive>code.codegenerationengine.frontend.JavaScriptListTableDriver</drive>
            <outputType>.js</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
        <transformation>
            <destinationPath>/Users/andreribeiro/Desktop/</destinationPath>
            <template>modules/frontEndIndex.ftl</template>
            <drive>code.codegenerationengine.frontend.IndexDriver</drive>
            <outputType>.html</outputType>
            <outputRange>n-1</outputRange>
        </transformation>
    </transformations>
</system>
```

6.2 Anexo 2

Tabela 6 – Código pedidos e respostas webservice Spring

N	Pedido	Resposta
1	localhost:8080/catalogo/create?nome=CatalogoTeste&validade=25-12-2015	{"ID":1}
2	localhost:8080/catalogo/all	[{"nome":"CatalogoTeste","validade":"25-12-2015","catalogoID":1}]
3	localhost:8080/categoria/create?nome = CategoriaTeste&validade=25-12-2015	{"ID":1}
4	localhost:8080/ categoria /create?nome=CategoriaTeste1&validade=25-12-2015	{"ID":2}
5	localhost:8080/categoria/all	[{"nome":"CategoriaTeste","validade":"25-12-2015","categoriaID":1},{"nome":"CategoriaTeste2","validade":"25-12-2015","categoriaID":2}]
6	localhost:8080/produto/create?preco=10&quantidade=20&validade=25-12-2015	{"ID":1}
7	localhost:8080/produto/create?preco=5&quantidade=10&validade=25-12-2015	{"ID":2}
8	localhost:8080/produto/create?preco=1&quantidade=5&validade=25-12-2015	{"ID":3}
9	http://localhost:8080/produto/all	[{"validade":"25-12-2015","quantidade":20,"preco":10.0,"produtoID":1},{"validade":"25-12-2015","quantidade":10,"preco":5.0,"produtoID":2},{"validade":"25-12-2015","quantidade":5,"preco":1.0,"produtoID":3}]

6.3 Anexo 3

```
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;

public class LoginLDAP {

    public tryLoginLDAP(String user,String password) throws Exception {

        String url = "ldaps://localhost:389/${systemName}"; // ldap url
        String domainName = params.get("${systemName}"); // system domain

        if (domainName==null || "".equals(domainName)) {
            int delim = principalName.indexOf('@');
            domainName = principalName.substring(delim+1);
        }

        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
        props.put(Context.PROVIDER_URL, url);
        props.put(Context.SECURITY_PRINCIPAL, user);
        props.put(Context.SECURITY_CREDENTIALS, password); // secretpwd
        if (url.toUpperCase().startsWith("LDAPS://")) {
            props.put(Context.SECURITY_PROTOCOL, "ssl");
            props.put(Context.SECURITY_AUTHENTICATION, "simple");
            props.put("java.naming.ldap.factory.socket",
"test.DummySSLSocketFactory");
        }

        InitialDirContext context = new InitialDirContext(props);
        try {
            SearchControls ctrls = new SearchControls();
            ctrls.setSearchScope(SearchControls.SUBTREE_SCOPE);
            NamingEnumeration<SearchResult> results =
context.search(toDC(domainName),"(&
(userPrincipalName="+user+")(objectClass=user))", ctrls);
            if(!results.hasMore())
                throw new AuthenticationException("Principal name not
found");

            SearchResult result = results.next();
            System.out.println("distinguisedName: " +
result.getNameInNamespace() ); // CN=Firstname
Lastname,OU=Mycity,DC=mydomain,DC=com

            Attribute memberOf = result.getAttributes().get("memberOf");
            if(memberOf!=null) {
                for(int idx=0; idx<memberOf.size(); idx++) {
                    System.out.println("memberOf: " +
memberOf.get(idx).toString() ); // CN=Mygroup,CN=Users,DC=mydomain,DC=com
                }
            }
```

```
        } finally {
            try { context.close(); } catch(Exception ex) { }
        }
    }

    /**
     * Create "DC=sub,DC=mydomain,DC=com" string
     * @param domainName    sub.mydomain.com
     * @return
     */
    private static String toDC(String domainName) {
        StringBuilder buf = new StringBuilder();
        for (String token : domainName.split("\\.")) {
            if(token.length()==0) continue;
            if(buf.length()>0)  buf.append(",");
            buf.append("DC=").append(token);
        }
        return buf.toString();
    }
}
```

6.4 Anexo 4

```
package code.codegenerationengine.springwebservices;

import java.util.List;

import code.codegenerationengine.springwebservices.${className};
import code.codegenerationengine.springwebservices.${className}Dao;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * A class to test interactions with the MySQL database using the
${className}Dao class.
 *
 */
@Controller

public class ${className}Controller  {

 // ------------------------
  // PUBLIC METHODS
  // ------------------------
  /**
   * /create  --> Create a new ${className?uncap_first} and save it in the
database.
   *
   *<#list attributes as attribute>
    <#if attribute.isKey == false>* @param ${attribute.name} ${className}
${attribute.name}</#if>
    </#list>
   * @return A string describing if the ${className?uncap_first} is
succesfully created or not.
   */
  @RequestMapping("${className?uncap_first}/create")
  @ResponseBody
  public String create(<#list attributes as attribute><#if attribute.isKey
== false>${attribute.type} ${attribute.name}<#if attribute_has_next><#if
attributes[attribute_index +1].isKey ==
false>,</#if></#if></#if></#list>,String user, String password){
  tryLoginLDAP(user,password);
   ${className} ${className?uncap_first} = null;
    try {
      ${className?uncap_first} = new ${className}(<#list attributes as
attribute><#if attribute.isKey == false>${attribute.name}<#if
attribute_has_next><#if attributes[attribute_index +1].isKey ==
false>,</#if></#if></#if></#list>);
      ${className?uncap_first}Dao.save(${className?uncap_first});
    }
    catch (Exception ex) {
      return "Error creating the ${className?uncap_first}: " +
ex.toString();
    }
```

```
    return "${className} succesfully created! (id = " +
${className?uncap_first}.get<#list attributes as attribute><#if
attribute.isKey == true>${attribute.name}</#if></#list>() + ")";
  }

  /**
   * /delete  --> Delete the ${className?uncap_first} having the passed
key.
   *
   <#list attributes as attribute>
    <#if attribute.isKey == true>* @param ${attribute.name} ${className}
${attribute.name}</#if>
    </#list>
   * @return A string describing if the ${className?uncap_first} is
succesfully deleted or not.
   */
  @RequestMapping("${className?uncap_first}/delete")
  @ResponseBody
  public String delete(<#list attributes as attribute><#if attribute.isKey
== true> ${attribute.type} ${attribute.name}</#if></#list>,String user,
String password){
  tryLoginLDAP(user,password);
    try {
      ${className} ${className?uncap_first} = new ${className}(<#list
attributes as attribute><#if attribute.isKey ==
true>${attribute.name}</#if></#list>);
      ${className?uncap_first}Dao.delete(${className?uncap_first});
    }
    catch (Exception ex) {
      return "Error deleting the ${className?uncap_first}:" +
ex.toString();
    }
    return "${className} succesfully deleted!";
  }


   /**
   * /update  --> Update the email and the name for the
${className?uncap_first} in the database
   * having the passed id.
   *
   <#list attributes as attribute>
    * @param ${attribute.name} ${className} ${attribute.name}
    </#list>
   * @return A string describing if the ${className?uncap_first} is
succesfully updated or not.
   */
  @RequestMapping("${className?uncap_first}/update")
  @ResponseBody
  public String update${className}(<#list attributes as
attribute>${attribute.type} ${attribute.name}<#if
attribute_has_next>,</#if></#list>,String user, String password){
  tryLoginLDAP(user,password);
    try {
      ${className} ${className?uncap_first} =
${className?uncap_first}Dao.findBy<#list attributes as attribute><#if
attribute.isKey == true>${attribute.name}(${attribute.name})</#if></#list>;
      <#list attributes as attribute>
      ${className?uncap_first}.set${attribute.name}(${attribute.name});
```

```
      </#list>
      ${className?uncap_first}Dao.save(${className?uncap_first});
    }
    catch (Exception ex) {
      return "Error updating the ${className?uncap_first}: " +
ex.toString();
    }
    return "${className} succesfully updated!";
  }

   <#list attributes as attribute>
   <#if attribute.isKey == false>
    /* @param ${attribute.name} ${className} ${attribute.name}

   /**
   * GET /get-by-${attribute.name}  --> Return the id for the
${className?uncap_first} having the passed
   * ${attribute.name}.
   */
  @RequestMapping("${className?uncap_first}/get-by-${attribute.name}")
  @ResponseBody
  public String getBy${attribute.name}(${attribute.type}
${attribute.name},String user, String password){
  tryLoginLDAP(user,password);
    <#list attributes as attribute><#if attribute.isKey ==
true>${attribute.type} ${attribute.name}</#if></#list>;
    try {
      ${className} ${className?uncap_first} =
${className?uncap_first}Dao.findBy${attribute.name}(${attribute.name});
      <#list attributes as attribute><#if attribute.isKey ==
true>${attribute.name}</#if></#list> = ${className?uncap_first}.get<#list
attributes as attribute><#if attribute.isKey ==
true>${attribute.name}</#if></#list>();
    }
    catch (Exception ex) {
      return "${className} not found";
    }
    return "The ${className?uncap_first} id is: " + <#list attributes as
attribute><#if attribute.isKey == true>${attribute.name}</#if></#list>;
  }
 </#if>
 </#list>
   /**
   * GET /all  --> Return all elements from ${className}
   *
   */
  @RequestMapping("${className?uncap_first}/all")
  @ResponseBody
  public List<${className}> getAll${className}() {
    List<${className}> ${className?uncap_first}s = null;
      try {
       ${className?uncap_first}s = (List<${className}>)
${className?uncap_first}Dao.findAll();

    }
    catch (Exception ex) {
      return ${className?uncap_first}s;
    }
    return ${className?uncap_first}s;
  }
```

```
  // ------------------------
  // PRIVATE FIELDS
  // ------------------------

  @Autowired
  private ${className}Dao ${className?uncap_first}Dao;

}
```