PM

Desenvolvimento de uma Ferramenta de Geração de Código para um Sistema de Informação

PROJETO DE MESTRADO

André Ornelas Telo
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

setembro | 2021

Desenvolvimento de uma Ferramenta de Geração de Código para um Sistema de Informação

PROJETO DE MESTRADO

André Ornelas Telo

MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO

Eduardo Miguel Dias Marques

CO-ORIENTAÇÃO

Pedro Dionísio Valente

Desenvolvimento de uma Ferramenta de Geração de Código para um Sistema de Informação

André Ornelas Telo

Constituição do júri de provas públicas:

Karolina Baras, Professora Auxiliar, UMa, Presidente

Leonel Nóbrega, Professor Auxiliar, UMa, Vogal

Eduardo Marques, Professor Auxiliar, UMa, Vogal

dezembro 2021

Funchal – Portugal

Resumo

A análise de requisitos de software e design de software são tradicionalmente inconsistentes, nomeadamente na perspetiva de modelação de negócios, software e rastreabilidade de implementação [1]. Este problema poderá fazer com que a fase de desenvolvimento dos projetos seja um processo mais demorado que o esperado, e quando um dos modelos sofre uma alteração também poderá fazer com que potencialmente não esteja garantida a rastreabilidade, a partir de qualquer um dos modelos, até à implementação do código.

O problema maior das possíveis alterações dos requisitos e de não ser mantida a atualização dos modelos, é que por vezes a adaptação de um sistema, já implementado ou em vias de implementação, não é simples, podendo fazer com que estas alterações tenham um custo elevado, tal como um impacto negativo relativamente ao tempo de implementação, podendo inclusivamente inviabilizar o projeto [2].

O objetivo deste projeto passa pelo desenvolvimento de uma ferramenta que permita gerar o código que possa ser utilizado, por sua vez, no *Hydra Code Generator* para gerar o código para a plataforma Hydra, de forma a estabelecer uma relação direta entre a modelação do negócio e a implementação, e assim acelerar o processo de implementação do software, e tornar a fase de desenvolvimento de software mais rápida e automatizada. O código deverá ser gerado tendo em conta uma modelação de um processo de negócio. A modelação será assim baseada numa abordagem aplicada a uma ferramenta de geração de código para a *framework* Hydra [3].

A principal contribuição deste projeto visa acelerar e automatizar o processo de desenvolvimento de software utilizado na Universidade da Madeira, automatizando a implementação com a geração de uma *Domain Specific Languag*e (DSL) para a *framework* Hydra, através da implementação da relação entre o modelo de negócio, o *Plataform Independent Model* (PIM), e o modelo de implementação, o *Plataform Specific Model* (PSM), de forma a conseguir gerar a interface do utilizador, a lógica de negócio e a base de dados.

Posteriormente, foram realizados casos de estudo como forma de testar e verificar o funcionamento da ferramenta implementada assim como verificar as suas limitações.

Palavras-chave: Geração de Código, *Domain Specific Language*, *Model Driven Architecture*, *Computation Independent Model*, *Plataform Independent Model*, *Plataform Specific Model*, Wisdom, Goals Approach

Abstract

Requirements analysis in software and software design are traditionally inconsistent, namely from the perspective of business modeling, software and implementation traceability [1]. This problem can make the development phase of the projects a process that takes longer than expected, and when one of the models is changed, it can also make the traceability not guaranteed, from any of the models to code implementation.

The biggest problem with possible changes in system requirements and not keeping the the models updated is that sometimes the adaptation of a system, already implemented or in the process of being implemented, is not simple, which can cause these changes to have a high cost, as well as a negative impact in the implementation time, which may even make the project unfeasible [2].

The objective of this project involves the development of a tool that allows the generation of code that can be used, in the Hydra Code Generator to generate the code for the Hydra platform, in order to establish a direct relationship between business modeling and implementation, and speeding up the software implementation process, and making the software development phase faster and more automated. The code should be generated taking into account a business process modeling. The modeling will be based on an approach applied to a code generation tool for the Hydra framework [3].

The main contribution of this project aims to accelerate and automate the software development process used at the University of Madeira, automating the implementation with the generation of a Domain Specific Language (DSL) for the Hydra framework, through the implementation of the relationship between the business model, the Platform Independent Model (PIM), and the implementation model, the Platform Specific Model (PSM), in order to generate the user interface, the business logic and the database.

Later, case studies were carried out as a way of testing and checking the operation of the implemented tool and to verify its limitations.

keywords: Code Generation, Domain Specific Language, Model Driven Architecture, Computation Independent Model, Plataform Independent Model, Plataform Specific Model, Wisdom, Goals Approach

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1 Motivação

O estabelecimento de um processo de desenvolvimento bem definido que permita a implementação de um sistema de informação baseado numa arquitetura de software, também bem definida, e com sucessos recorrentes, ainda não foi alcançado [2], [4], [5]. A inconsistência entre a análise de requisitos de software e o desenho de software é um problema que costuma ocorrer durante a fase de implementação de um projeto, nomeadamente na perspetiva de modelação de negócios, de software e rastreabilidade de implementação [1].

Este problema poderá fazer com que a fase de desenvolvimento dos projetos seja um processo mais demorado que o esperado, e que quando um dos modelos sofre uma alteração, o outro não é necessariamente alterado. Também poderá fazer com que potencialmente não esteja garantida a rastreabilidade, a partir de qualquer um dos modelos, até à implementação do código.

Consequentemente, o facto de existirem muitas destas alterações e de não ser mantida a atualização dos modelos, é que por vezes a adaptação de um sistema, já implementado ou em vias de implementação, não é simples, podendo fazer com que estas alterações tenham um custo elevado, tal como um impacto negativo relativamente ao tempo de implementação, podendo inclusivamente inviabilizar o projeto [2].

Para tal, seria importante que existisse uma ferramenta capaz de facilitar o processo de implementação do software, garantindo uma relação consistente entre a modelação do negócio, a modelação do software, e implementação (ficheiros, objetos, dados) do software, de forma a facilitar tanto a implementação, como a reimplementação de requisitos.

1.2 Contexto

A Universidade da Madeira tem um departamento de desenvolvimento (GDAI - Gabinete de Desenvolvimento de Aplicações Informáticas) no qual o objetivo é realizar a implementação de vários processos de negócio ou projetos de forma a melhorar as plataformas a serem utilizadas pelos docentes e alunos.

A implementação destes projetos envolve um processo de desenvolvimento no qual primeiro é realizada a modelação do processo de negócio, utilizando o modelo *Goals Approach*[6] que por sua vez utiliza o método Wisdom[7].

Após a modelação do processo de negócio, durante a fase de implementação é utilizado o *Hydra Code Generator*, um gerador de código que recebe como input uma *Domain Specific Language* (DSL), que gera código numa estrutura Hydra de forma a tornar possível a realização ou implementação de novas funcionalidades em várias plataformas como o InfoAlunos (plataforma dos alunos), o Sidoc (plataforma dos docentes) e entre outras.

Existe a preocupação em melhorar o processo de desenvolvimento de software com o intuito de acelerar a fase de implementação dos processos de negócio que possam ser propostos, e também para facilitar a adaptação de um sistema já implementado ou a ser implementado, pois como já foi referido, as alterações num sistema muitas vezes não são fáceis de fazer, sendo necessário alterar e adaptar o código específico para a ser utilizado pelo *Hydra Code Generator* (ferramenta utilizada no Gabinete de aplicações Informáticas da Universidade da Madeira) cada vez que é alterado algum regulamento ou funcionamento do sistema.

1.3 Objetivos

Tendo em conta o problema apresentado, o objetivo deste projeto passa pelo desenvolvimento de uma ferramenta que permita gerar o código que o Gabinete de aplicações Informáticas da

Universidade da Madeira usa na *Hydra Code Generator* para gerar o código para a plataforma Hydra, por forma a estabelecer uma relação direta entre a modelação do negócio e a implementação, acelerar o processo de implementação do software, para tornar a fase de desenvolvimento de software mais rápida e automatizada. O código deverá ser gerado tendo em conta uma modelação de um processo de negócio. A modelação será assim baseada numa abordagem que unifica a engenharia empresarial à engenharia informática (*The Goals Approach* [6]) aplicada a uma ferramenta de geração de código para a *framework* Hydra [3].

Tendo conhecimento da modelação do processo de negócio com base no modelo Wisdom [7] e *Goals Approach*, e com conhecimento da *framework* Hydra, e tendo em conta que o código utilizado para a *framework* Hydra envolve um processo que é feito de forma manual e que poderá ocupar um tempo relevante por parte do desenvolvedor, o objetivo deste projeto será arranjar uma solução com o intuito de contribuir para a automatização da geração de código, que por sua vez poderá servir para gerar código para a *framework* Hydra ao nível da interface de utilizador, lógica de negócio e base de dados. O objetivo será fazer a geração com base na modelação de um processo de negócio, ou seja, fazer com que através de um modelo se consiga gerar código sem a necessidade de alterações por parte do programador (ou desenvolvedor).

O maior desafio deste projeto é a implementação de uma solução que permita gerar código através da modelação e que possa ser utilizado para o *Hydra Code Generator*, ou seja, numa abordagem *Model Driven Architecture* (MDA), fazer a transformação do modelo de negócio, o Platform Independent Model (PIM) [8]**,** para o modelo de implementação, o Platform Specific Model (PSM) [8] que leva à geração da DSL**,** através da criação de um módulo de integração entre estes modelos (PIM e PSM).

A ferramenta implementada para este projeto será utilizada para a implementação de um processo de negócio real da Universidade da Madeira.

1.4 Contribuição

Uma das contribuições deste projeto é o Estado de Arte que permite investigar o que existe acerca da geração automática de código, sendo possível retirar algumas conclusões daquilo que é encontrado sobre este tema e como este poderá ter cada vez mais importância ao longo do tempo.

A principal contribuição deste projeto visa melhorar o processo de desenvolvimento de software utilizado na Universidade da Madeira, automatizando a geração de uma DSL para a *framework* Hydra, através da implementação da relação entre o modelo de negócio, o PIM, e o modelo de implementação, o PSM, de forma a conseguir gerar a interface do utilizador, a lógica de negócio e a base de dados. Para tal, através de um modelo de negócio poderá ser possível gerar código sem a necessidade de haver alterações por parte do programador (ou desenvolvedor).

A automatização deste processo poderá contribuir para que, fazer as alterações na modelação não provoque um impacto muito grande no tempo da implementação do código, que também é um dos problemas referidos neste projeto.

1.5 Questão de Investigação

Um dos objetivos deste projeto será também, através de uma possível solução implementada para o problema deste projeto referido com mais detalhe na secção 3 “Descrição do Problema”, responder a uma questão de investigação:

Será possível realizar a geração de módulos que possam ser utilizados por uma framework apenas com modelação de um processo de negócio, sem existir a necessidade de verificação manual por parte do desenvolvedor?

Neste projeto também serão apresentados casos de estudo, que irão permitir responder à questão de investigação através de testes efetuados com uma possível solução implementada.

1.6 Estrutura da Tese

Este trabalho encontra-se estruturado em sete capítulos, sendo este o capítulo 1, "Introdução", no qual é descrito o que irá ser abordado durante todo o projeto dando uma visão dos seus objetivos.

No capítulo 2, "Estado de Arte", são introduzidos primeiro alguns conceitos relativamente a linguagem de modelação "*Unified Modeling Language*" (UML) e a "*Model Driven Architecture*" (MDA) que são necessários para entender o resto do capítulo e do documento. Ainda no capítulo 2 é descrito o método de investigação utilizado para a pesquisa dos trabalhos relacionados com tema e depois nos "Resultados obtidos" são descritos os resultados desta investigação, sendo cada um dos mesmos analisados e colocados em quatro categorias. Posteriormente é realizada uma comparação destes resultados utilizando uma tabela, e é apresentada uma conclusão daquilo que foi encontrado na pesquisa destes trabalhos relacionados.

No capítulo 3, "Descrição do Problema", é primeiro feita uma introdução ao problema incluído neste projeto, assim como a introdução das ferramentas utilizadas durante a realização do processo de desenvolvimento de software na Universidade da Madeira, como o "Goals Approach", "Wisdom" e a *Framework* Hydra. Ainda neste capítulo e realizada uma análise a este processo de desenvolvimento de software e depois é apresentada uma pequena conclusão que também dá de alguma forma uma ideia do que poderá ser melhorado para resolver o problema.

No capítulo 4, "Análise e Especificação", é apresentada a solução para o problema, referido no capítulo 3, sendo apresentado de que forma será realizada a modelação do processo de negócio e de que forma está implementada a ferramenta de geração de código.

No capítulo 5, "Implementação", primeiro é feita uma introdução à uma ferramenta de geração de código da *framework* Hydra (Hydra Code Generator) já existente e depois é descrito o método de implementação da ferramenta de geração de código realizada neste projeto. Também contém a descrição das fases de desenvolvimento da ferramenta implementada bem como a informação acerca das palavras-chave que é possível utilizar na modelação com a versão atual da ferramenta.

No capítulo 6, "Casos de Estudo" são apresentados os vários processos de negócio que foram utilizados como casos de estudo, assim como a modelação do processo de negócio para cada um deles. Também é descrito de que forma foram realizados os testes em cada um dos casos de estudo e as conclusões obtidas.

No capítulo 7, "Conclusão", é apresentada a conclusão que é possível retirar para todo este projeto realizado, assim como algumas propostas de trabalho futuro a desenvolver.

2. Estado de Arte

Nesta secção são inicialmente apresentados alguns conceitos relacionados com a geração automática de código e o *Model Driven Architeture* (MDA). Também são apresentados alguns conceitos acerca do *Unified Modeling Language* (UML), visto que muitos dos trabalhos relacionados deste tema apresentam diagramas UML para explicar o seu contributo, sendo então necessária a sua compreensão. É descrito o método de investigação utilizado, sendo apresentados os resultados obtidos, que são depois comparados através de uma tabela.

2.1 Geração Automática de Código

A geração automática de código tem ganho muita atenção por parte da engenharia de software pois pode oferecer muitos benefícios como estar menos sujeito a erros e a perder demasiado tempo com as suas correções, ajudar na reutilização do código e obter maior facilidade na manutenção do sistema [9]. Conseguir gerar código automaticamente a partir de um modelo faz com que a modelação de um sistema seja uma tarefa mais importante e eficaz, pois a geração poderá permitir reduzir os erros ocorridos durante a implementação, e como tal, reduzir também o tempo utilizado na implementação. A procura de ferramentas de geração automática de código tem aumentado, sendo utilizados diagramas UML ou XML, para obter a linguagem requerida de forma mais automática.

A ferramenta de geração automática de código utilizará uma *General Purpose Language* (GPL [10]), como *Plataform Independent Model* (PIM), para a modelação do processo de negócio, que será transformada em *Domain Specific Language* (DSL [11]), como *Plataform Specific Model* (PSM), para a especificação da implementação, DSL a partir da qual será gerado o código. A GPL é uma linguagem capaz de modelar ou implementar qualquer tipo de programa e pode ser por exemplo UML, C++, Java, entre outros. A DSL é uma linguagem que pode ter como objetivo criar um programa, ou ser utilizada para algo específico. Por exemplo, o *HyperText Markup Language* (HTML) é uma DSL porque apenas consegue mostrar dados ao utilizador e o SQL também é porque é apenas utilizado para realizar várias operações na base de dados. As DSL analisadas no presente documento, têm como objetivo a conceção parcial ou total de um programa.

2.2 Model Driven Architecture (MDA)

O *Object Management Group* (OMG) propôs uma abordagem chamada MDA [12]. Esta abordagem foi proposta para a utilização de modelos durante a fase de desenvolvimento de software, que são uma versão abstrata do sistema pretendido. O objetivo da utilização do MDA era aumentar a reutilização, reduzir a complexidade e os custos. O MDA tem têS modelos envolvidos que são o *Computation Independent Model* (CIM), *Plataform Independent Model* (PIM) e o *Plataform Specific Model* (PSM). Estes modelos vão sofrendo transformações até chegar a um modelo resultante que é o modelo menos abstrato do sistema pretendido [13].

O *Computation Independent Model* (CIM) é um modelo que representa o nível mais alto de abstração e descreve os requisitos do sistema e como ele funciona no seu ambiente, enquanto os detalhes da estrutura da aplicação e da realização estão ocultos ou mesmo indeterminados [14].

O *Plataform Independent Model* (PIM) é um modelo abstrato que contém informações suficientes para conduzir a um ou mais *Plataform Specific Models* (PSM). É um modelo que descreve os detalhes do sistema sem mostrar detalhes específicos para uma plataforma de execução ou uma tecnologia particular[14].

O *Plataform Specific Model* (PSM) é um modelo que descreve os detalhes e características que foram excluídas do PIM. Deve ser adaptado para especificar a implementação do sistema numa única plataforma tecnológica[14]. Os artefatos PSM possíveis podem incluir um código-fonte,

Data Definition Language (DDL), arquivos de configuração, XML e outras saídas específicas para a plataforma de destino.

Cada um destes três modelos representam níveis de detalhe diferentes segundo Zachman, em que o CIM é o mais abstrato apresentando uma perspetiva de controlo de negócio, o PIM apresenta os detalhes numa perspetiva de lógica de negócio e o PSM apresenta os detalhes na perspetiva de engenharia, sendo este último o modelo menos abstrato [15].

2.3 Unified Modeling Language

O *Unified Modeling Language* (UML) é uma linguagem padrão de modelação desenvolvida por Grady Booch, Ivar Jacobson e James Rumbaugh em 1996 [16], vocacionada para a modelação do desenho e implementação de sistemas baseados em software. É uma linguagem muito utilizada na área de engenharia de software, nomeadamente para facilitar a visualização da estrutura ou do comportamento do sistema a ser desenvolvido [16]–[18].

O UML possui diagramas que permitem visualizar o sistema a nível estrutural como a utilização dos diagramas de classes ou de componentes ou de objeto. Também permite visualizar a nível comportamental com a utilização dos diagramas de casos de utilização ou de atividade (que assim descrevem parcialmente o negócio), e também possui diagramas de iteração como o diagrama de estados ou de comunicação [19].

Os diagramas estruturais são utilizados para a documentação de uma arquitetura de software.

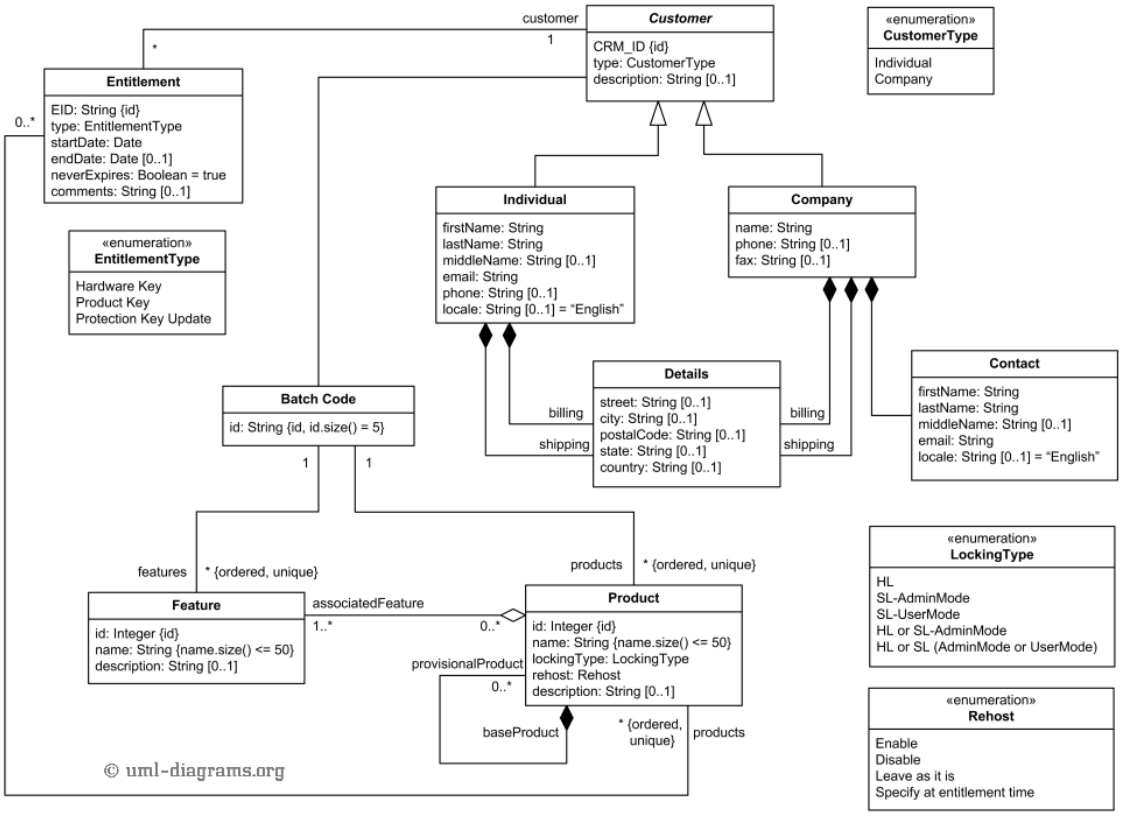

Figura 1. Diagrama de classes[20]

Como é possível verificar na Figura 1, com o diagrama de classes consegue-se saber melhor os objetos ou classes que vão ser criadas e os seus possíveis atributos, representando também as

relações entre objetos para a implementação do sistema, melhorando o conhecimento da arquitetura do sistema. Em comparação com o diagrama de componentes, este diagrama só mostra as classes do sistema e as suas relações.

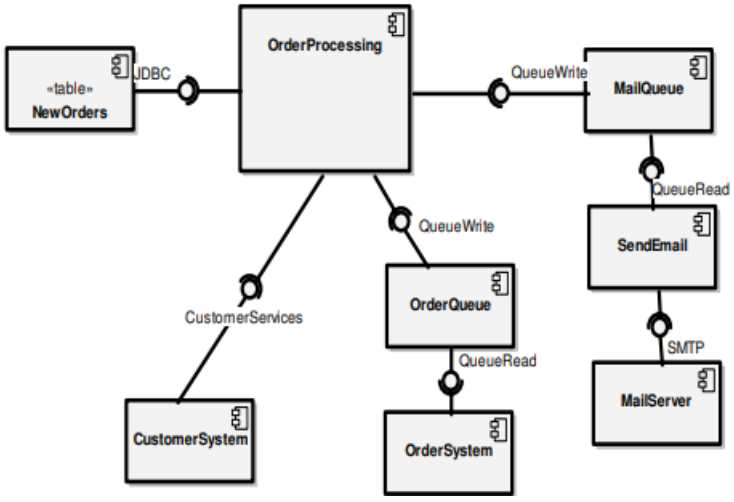

Figura 2. Diagrama de componentes[21]

Os diagramas de componentes, como o representado na Figura 2, descrevem a relação entre componentes com interfaces bem definidas. Cada um dos componentes como o "OrderSystem" podem ter várias classes incluídas.

Os diagramas comportamentais representam o funcionamento (comportamento) de um software, representando o que deverá acontecer durante a sua utilização.

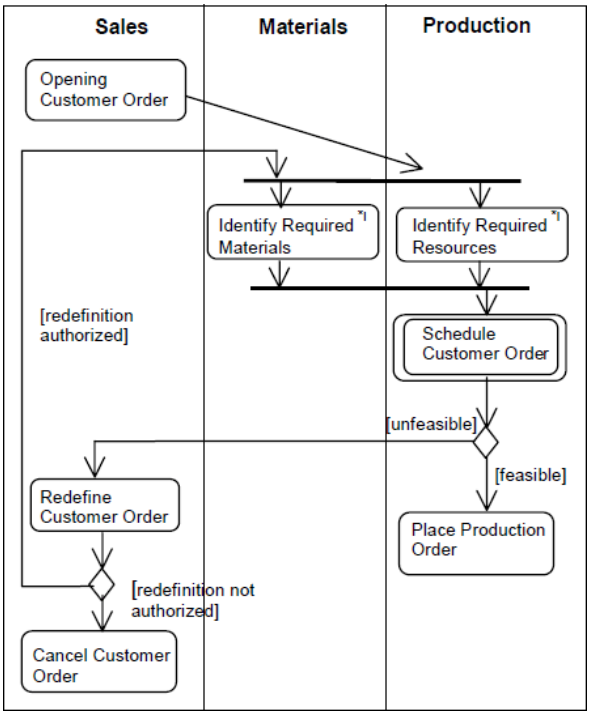

Figura 3. Diagrama de atividade[22]

O Diagrama de atividade é um diagrama comportamental e mostra o comportamento do sistema ao realizar uma atividade que são representados por um retângulo como podemos verificar na Figura 3 que pode estar em diferentes níveis de abstração. Poderá representar todos

os procedimentos do sistema dependendo do seu tamanho e do seu nível de abstração. Os diagramas de atividade também mostram as decisões do sistema, sendo estas decisões representadas por um losango perante certas atividades.

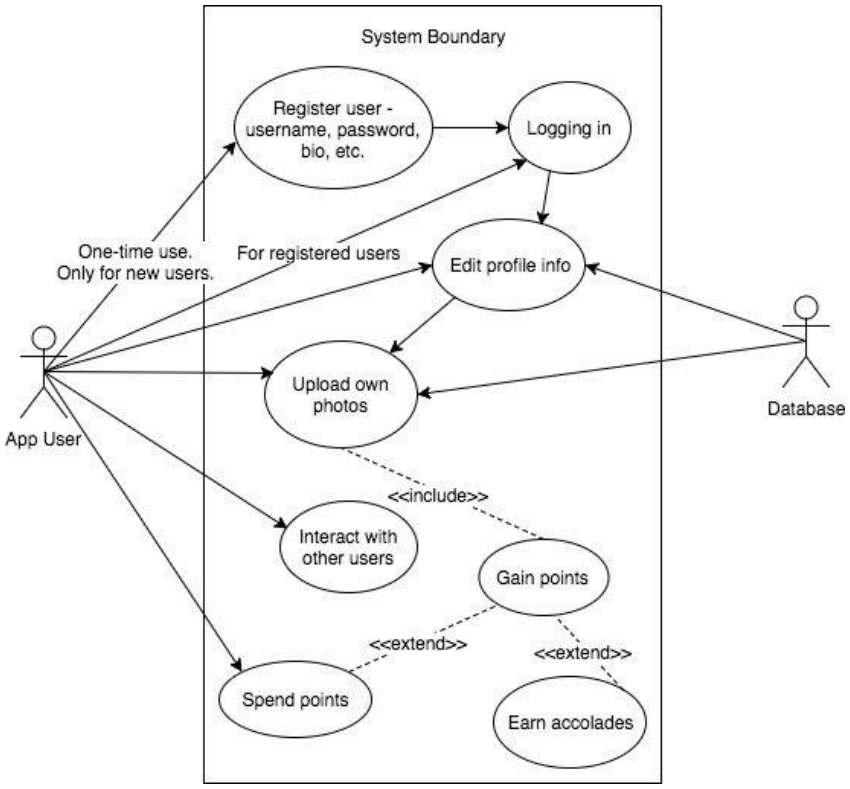

Figura 4. Diagrama de casos de utilização [23]

O Diagrama de casos de utilização é um diagrama comportamental e como é possível verificar na Figura 4 mostra os utilizadores do sistema, que são chamados de atores e nesta Figura existem dois atores (App User e Database). Mostra também o que cada ator diferente poderá realizar no sistema de acordo com as suas diferentes escolhas.

Os diagramas de interação são um conjunto de diagramas comportamentais que representam o controlo do fluxo de dados no software, mostrando a relação entre objetos no software.

Como é possível verificar na Figura 5, com este diagrama consegue-se saber melhor quando é que os dados são consultados ou guardados na base de dados como por exemplo quando o utilizador faz o Upload abstract &paper os dados são guardados na base de dados.

Com a informação obtida através destes diagramas UML poderá ser mais fácil o desenvolvimento do sistema podendo também ser utilizados como modelo de negócio para a gerar código automaticamente utilizando uma abordagem como o MDA.

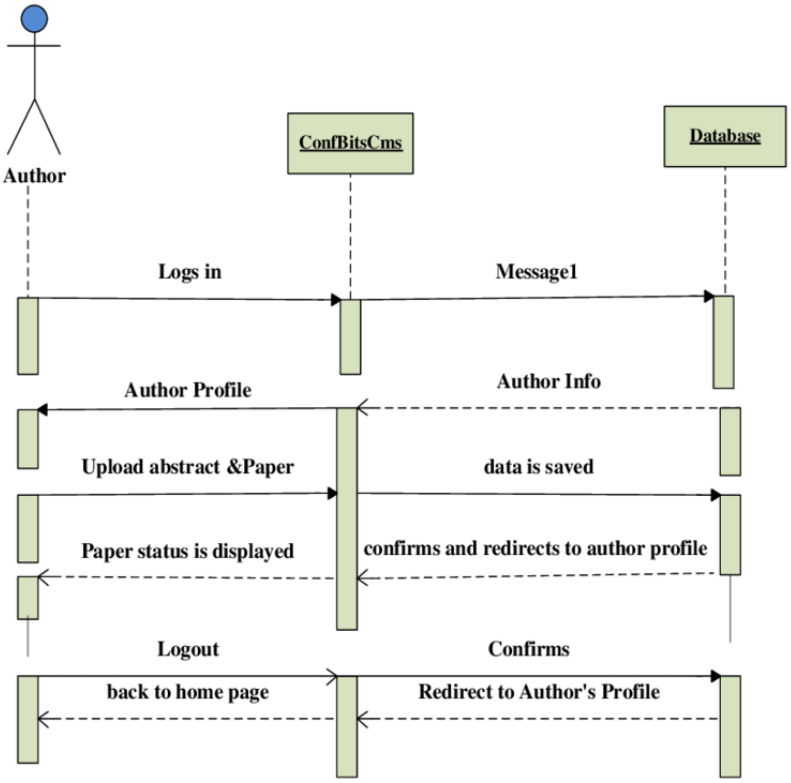

Figura 5. Diagrama de interação ou de sequência[24]

2.4 Método de Investigação

O trabalho de pesquisa teve por objetivo descobrir ferramentas de geração de código, se possível com o sistema a funcionar, com base na modelação de tarefas, preferencialmente de processos de negócio.

Foi definido o motor de pesquisa Google Académico como uma fonte fiável de informação científica e também por ser fácil de encontrar artigos científicos e ser possível encontrar artigos com variadas fontes de publicações como o IEEE ou o Research Gate ou qualquer outra fonte fiável.

Foram definidas frases de pesquisa que refletem distintas abordagens para encontrar a informação desejada:

- Geração de código independentemente de qualquer modelo:
  - "Automatic Code generator"
- Geração de código baseada em MDA:
  - "Automatic Code generation using MDA"
- Geração de código baseada em MDA e XML:
  - "Automatic Code generation using XML with MDA"
- Geração de código baseada em UML:
  - "Automatic code generation using UML"

Tendo em conta cada termo de pesquisa, utilizando o Google Académico forma encontrados muitos resultados, no qual foram analisados os primeiros 20 resultados, tendo em atenção o ano de publicação e aos critérios de inclusão. Para tal, foram aplicados os seguintes critérios de inclusão:

- Tem que fazer modelação de alguma parte do negócio, pelo menos tarefas do utilizador;
- Tem que gerar código.

Os critérios de exclusão aplicados foram:

- Não fazer geração de código;
- Não utilizar modelos para a geração de código;
- Artigos repetidos durante as frases de pesquisa;
- Ser mais focalizado noutro tema que não seja geração de código.

Por forma a conseguir processar todos os resultados, foi feita a seguinte abordagem de análise:

- Leitura do Resumo (Abstract);
- Análise dos modelos apresentados;
- Leitura da especificação de geração de código.

Das 4 frases de pesquisa, como já foi salientado acima foram considerados os primeiros 20 resultados, originando um total de 80 artigos. Destes 80 artigos, 19 foram repetidos, 1 já estava relacionado com o trabalho [25], 38 resultados foram excluídos de acordo com os critérios de exclusão, e 22 foram incluídos. Dos 22 artigos incluídos, 4 já estavam relacionados com outros artigos encontrados mais recentes, originando um total de 18 artigos incluídos neste trabalho.

2.5 Resultados Obtidos

Os resultados obtidos foram separados em categorias para facilitar o estabelecimento de uma relação entre estes, e assim a sua comparação. As categorias propostas foram as seguintes:

- Utiliza a abordagem MDA e UML;
- Utiliza a abordagem MDA sem a utilização de UML;
- Utiliza o UML, mas não utiliza a abordagem MDA;
- Não utiliza a abordagem MDA nem UML.

Dos artigos encontrados foram incluídos 18 artigos que depois foram distribuídos pelas categorias acima indicadas.

Dos 18 artigos encontrados 7 foram incluídos na categoria “Utiliza a abordagem MDA e UML”, 1 foi incluído na categoria “Utiliza a abordagem MDA sem a utilização de UML”, 6 foram incluídos na categoria “Utiliza o UML, mas não utiliza a abordagem MDA”, e 4 artigos foram incluídos na categoria “Não utiliza a abordagem MDA nem UML”.

2.5.1 Geração de Código com a Utilização do MDA e UML

Nesta categoria foram encontrados 6 artigos [26]–[32], que utilizaram a abordagem MDA e UML e também tiveram como foco a geração de código automática.

O artigo de 2020 “*Automatic Code Generation of MVC Web Applications*” [26], tinha como objetivo a geração de código Java através do CIM que é um modelo muito abstrato. É utilizada uma ferramenta criada chamada de “xGenerator” que através do CIM consegue gerar o PIM, o PSM e o código Java de uma aplicação Web MVC. A abordagem utiliza transformações automáticas em todos os níveis do MDA sendo baseados em metamodelos para casos de utilização e diagramas de classes.

Outro artigo de 2018 “*Automatic code generation within MDA approach for cross-platform mobiles apps*” [27], do mesmo autor de [33], que tinha como objetivo o desenvolvimento automático de aplicações móveis para múltiplas plataformas de acordo com o princípio “*Develop Once, Run Everywere*”, focado numa metodologia baseada na abordagem MDA. Foram desenvolvidas meta-classes necessárias para a geração da aplicação. A abordagem aplica um modelo para a geração de classes C # que permitem operações CRUD em um ambiente Windows mobile a partir de diagramas de classes UML, usando Acceleo como linguagem de transformação. O objetivo da utilização da abordagem MDA foi transformar modelos documentais em modelos produtivos com a utilização de um conjunto de operações nesses

modelos para produzir aplicações. Apesar dos resultados, a abordagem ainda está numa fase preliminar, pois o código deve ser substituído manualmente e a aplicação em outros sistemas operacionais, como o iOS, ainda não é alcançada.

Posteriormente foi encontrado um artigo do ano de 2016 que tinha como título "*Automatic Code Generation Using UML Activity and Sequence Models*" [28], que tinha como objetivo formular um algoritmo composto por duas sub-rotinas capaz de gerar código proveniente de um modelo que é a combinação entre o diagrama de atividade e de sequência. A abordagem MDA foi utilizada para a geração de código, sendo desenvolvida uma ferramenta chamada *AutoKode* para a transformação de modelos. Foram desenvolvidos algoritmos capazes de gerar a definição de classes e de métodos, embora, no momento, cobrindo apenas a lógica de negócio (C) de um modelo MVC.

Depois foi encontrado um artigo de 2010 com o título "*MDA development of Manufacturing Execution System based on automatic code generation*" [29], que tem como objetivo propor um método que gere código automaticamente para o software Manufacturing Execution System (MES) que gera código dependendo da interface do sistema de controlo lower-level. O foco era melhorar a produtividade do desenvolvimento de controlo lower-level dentro do MES. Depois o método proposto é comparado com o método convencional. Os autores concluem que a abordagem melhora muito a produtividade, gerando apenas parte do código com base em modelos.

Posteriormente foi encontrado um artigo de 2006 com o título "*MDA-based approach for embedded software generation from a UML/MOF repository*" [30] que tem como objetivo a implementação de uma ferramenta de geração de código Java através de uma abordagem baseada no Model Driven Architecture (MDA) e que consegue gerar ficheiros em Java a partir de diagramas de classes UML usando templates Java. A interface do utilizador e a geração do código da base de dados estavam fora do foco da abordagem.

Depois foi encontrado ainda um artigo de 2005 com o título "*WSDL Automatic Generation from UML Models in a MDA Framework*" [31] que se baseia numa abordagem para gerar WSDL (Web Service Description Language), ou seja, a descrição das operações de um serviço web, com base na modelação do serviço web usando uma extensão UML, específica para esse propósito. Uma ferramenta CASE (Computer Assisted Software Engineering), MIDAS-CASE, é utilizada para modelar o serviço web e gerar o WSDL. É um artigo que apresenta a geração automática da respetiva descrição Web Service Description Language (WSDL) para o serviço Web modelado. São utilizados modelos de casos de utilização que depois são transformados em WSDL. Este artigo não tem em conta a interface do utilizador e a geração de código da base de dados.

Posteriormente foi encontrado um artigo com o título "*Design and Implementation of Code Generator for Model-driven Software Development*" [32] que teve como objetivo a criação de uma ferramenta desenvolvida no Visual Studio 2008, baseada no MDA, que dava para criar modelos UML como casos de utilização, diagrama de classes e diagrama de sequência e que depois através destes modelos foi possível criar classes de código havendo a necessidade da intervenção do programador para funcionar.

2.5.2 Geração de Código com a Utilização do MDA, mas Sem o UML

Nesta categoria foi encontrado 1 artigo [34] com o título "*Maximizing automatic code generation: Using XML based MDA*" que envolvia apenas a utilização da abordagem MDA e que apresentava soluções para a geração automática de código. Este artigo tinha como objetivo propor a ideia de introduzir o XML como PIM para gerar código em vez da utilização do UML. A ideia tinha sido proposta com o intuito de gerar maior número de código executável e reduzir a mudança manual no código por parte do desenvolvedor fazendo uma melhoria na geração automática de código. Para a transformação do XML em código foi utilizada uma ferramenta

(Visual XML) que usava o *swing* para o *frontend*, a base de dados MYSQL e o XML como PIM. Como resultado, as classes Java para conetividade da base de dados são geradas, não examinando o código da interface de utilizador ou o código da base de dados, embora também necessitem de ajustes por parte do programador.

2.5.3 Geração de Código Com UML e Sem a Utilização do MDA

Nesta categoria foram encontrados 5 artigos [35]–[40] que não utilizavam a abordagem MDA e que apenas tinham o foco na geração automática de código através da utilização de digramas UML.

Foi encontrado um artigo de 2019 com o título "*Automatic code generation from UML state chart diagrams*" [37], que resume e é uma evolução dos resultados de [41]–[43]. Este artigo descreve de que forma a que pode ser implementada a geração de código através de diagramas de estado. Esta implementação não era totalmente suportada pelas linguagens de programação existentes porque os elementos de programação não podiam implementar dois componentes principais com eficácia do diagrama de estado como a hierarquia e a simultaneidade. Para tal foi proposto um novo padrão de design para a implementação do diagrama de estado que incluía estados hierárquicos e simultâneos. As transições de estado dos estados paralelos seriam enviadas à classe do estado composto, sendo esta abordagem implementada e comparada com outras ferramentas. Apenas as classes Java são geradas.

Posteriormente foi encontrado um artigo de 2011 com o título "*Generating Java code from UML class and sequence diagrams*" [36], que apresentava uma abordagem de geração de código em que a partir de modelos UML seria capaz de gerar código estrutural e comportamental. Esta abordagem foi validada com uma ferramenta desenvolvida que captura um modelo UML, constituído por um diagrama de classes e vários diagramas de sequência, e gera código Java a partir deste. Foi apresentado um estudo de caso referente à geração de código, que não está completo e deve envolver a intervenção do programador. Para este artigo a abordagem utilizada não podia gerar código completo porque a granularidade do diagrama de sequência eram as invocações do método.

Depois foi encontrado um artigo de 2010 com o título "*Automatic code generation for embedded systems: From UML specifications to VHDL code*" [35] encontra a utilização de especificações UML para gerar HDL (*Hardware Description Language*). Seria possível criar classes em VHDL (*Very High Speed Integrated Circuit HDL*) com o auxílio de uma ferramenta chamada *GenERTiCA*, que gera classes VHDL incluindo os elementos modelados. A interface de utilizador e a geração do código da base de dados estão fora do objetivo da abordagem.

Um artigo de 2009 com o título "*Automatic Generation of Java Code from UML Diagrams using UJECTOR*" [38], mostra uma visão geral de uma ferramenta criada chamada de UJECTOR e que fazia a geração de código Java executável através de diagramas de classes e diagramas de atividade e diagramas de sequência. O objetivo principal do UJECTOR seria gerar um código funcional, completo e também compreensível como um valor acrescido para o desenvolvimento da lógica de negócios, sem considerar a interface de utilizador e a geração do código da base de dados. O autor acaba por concluir que o código gerado é funcional e compreensível.

Posteriormente foi encontrado ainda um artigo de 2005 com o título "*Automatic code generation from a UML model to IEC 61131-3 and system configuration tools*" [39], que mostra o desenvolvimento de uma abordagem que permite gerar código IEC 61131-3 a partir de um modelo UML, ou seja, permite gerar código para controladores lógicos programáveis (PLC) e importá-lo para soft-PLCs, automaticamente. A interface de utilizador e o código da base de dados estão fora do objetivo da abordagem.

Depois foi encontrado um artigo de 2015 com o título "*Automatic code generation for cross-platform, multi-device mobile apps: some reflections from an industrial experience*" [40] que tem

como objetivo apresentar os resultados de uma pesquisa para encontrar a melhor estratégia para uma empresa de desenvolvimento de software, interessada em produzir uma ferramenta MDD para criar e desenvolver aplicações móveis. É feita uma comparação entre as várias abordagens de geração automática de código que utilizam modelos de negócio.

2.5.4 Geração de Código Sem a Utilização da Abordagem MDA e do UML

Nesta categoria foram encontrados 4 artigos [44]–[47] que tinham o foco na geração automática de código a partir de outros modelos que não fizessem parte do UML e nem utilizavam da abordagem MDA.

O artigo de 2016 com o título "*Automatic code generation with business logic by capturing attributes from user interface via XML*" [47] tinha como objetivo fazer a geração automática de código gerado através de especificações XML da interface de utilizador realizadas utilizando o RPB (*Rapid Project Builder*). É apresentada uma ferramenta criada com o nome de "*Rapid Project Builder*" que seria uma ferramenta que dava para criar a interface com o utilizador e depois seria gerado código Java ou C++. O RPB gera formulários de interface de utilizador, o suporte para a lógica de negócios e o código da base de dados para criar as tabelas para operações CRUD. O código de destino poderia ser Java, C # ou C ++ e a base de dados de destino MS-Access ou MySQL. A abordagem encontra-se em fase experimental.

Posteriormente foi encontrado um artigo de 2014 com o título "*Automatic code generation from Matlab/Simulink for critical applications*" [44] que visa a geração de código para sistemas de aviação críticos e propõe a geração de código C por meio de modelos Simulinks e scripts de Matlab. A experiência neste momento gerou drivers com os quais foi possível controlar um motor BLDC (*brushless direct current electric*), do qual está implícita (não modelada) a análise da tarefa. O objetivo é gerar não apenas softwares de tarefas isoladas incluindo a interface interativa do utilizador, mas também sistemas que permitam combinar tarefas críticas complexas. Esta proposta fazia com que houvesse possibilidade de criar um sistema incluindo os Drivers para uma plataforma.

Depois foi encontrado um artigo de 2011 com o título "*Automatic code generation from event-B models*" [45] que apresentava uma ferramenta de tradução que gerava código automaticamente de uma linguagem de programação em C, C++, Java e C# com a especificação formal dos modelos Event-B (métodos de modelação) visando a análise de problemas complexos (não modelados). O objetivo seria gerar código através de uma expressão lógica (Event-B). Visando a geração de código, foi desenvolvida uma ferramenta de tradução como um conjunto de plug-ins Rodin para a ferramenta de desenvolvimento Eclipse, que permitem a interpretação do modelo e a geração do respetivo código, cobrindo necessidades específicas da lógica de negócio, enquanto a interface do utilizador e a base de dados não são consideradas nesta abordagem.

Foi encontrado ainda nesta categoria um artigo de 2008 com o título "*A Framework for Modeling, Simulation and Automatic Code Generation of Sensor Network Application*" [46], que apresentava uma *framework* baseada em Simulink, Stateflow e Embedded Code para criar componentes de rede de sensores que podem ser usados para modelar, simular e gerar código automaticamente para diferentes plataformas e sistemas operativos. Uma aplicação WSN (Wireless Sensor Network) foi gerada para os sistemas operativos TinyOS e MANTIS.

2.6 Resultados Excluídos

Com esta pesquisa apareceram resultados que foram excluídos de acordo com os critérios já apresentados no método de investigação utilizado para este trabalho. Desta forma foram excluídos os seguintes 36 resultados encontrados:

1. UML and SystemC -- A Comparison and Mapping Rules for Automatic Code Generation[48];
2. Automatic code generation for many-body electronic structure methods: the tensor contraction engine [49];
3. Automatic C-to-CUDA code generation for affine programs [50];
4. Using UML as Front- End for Heterogeneous Software Code Generation Strategies [51];
5. Automatic code generation from design patterns [52];
6. Constraint Support in MDA Tools: A Survey [53];
7. Automatic code generation for distributed memory architectures in the polytope model[54];
8. High-speed moving horizon estimation based on automatic code generation [55];
9. Automated code generation support for BI with MDA TALISMAN [56];
10. Optimization of the energy consumption of industrial robots for automatic code generation [57];
11. Template- and modelbased code generation for MDA-tools[58];
12. A Framework for Generating Query Language Code from OCL Invariants[59];
13. Requirements to Design to Code : Towards a Fully Formal Approach to Automatic Code Generation [60];
14. Challenging the interoperability between computers in industry with MDA and SOA [61];
15. Automatic Code Generation for High-Performance Discontinuous Galerkin Methods on Modern Architectures [62];
16. Automatic Code Generation within Student's Software Engineering Projects [63];
17. Automatic code generation from unified modelling language sequence diagrams [64];
18. Generating Android graphical user interfaces using an MDA approach [65];
19. Ocarina : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications [66];
20. Automatic code generation for real-time convex optimization [67];
21. Automatic Code Generation From Uml Diagrams: the State-of-the-Art [68];
22. Processes of community-led social enterprise development:Learning from the rural context [69];
23. An MDA Approach for the Generation of Communication Adapters Integrating SW and FW Components from Simulink [70];
24. Automatic code generation from high-level Petri-Nets for model driven systems [71];
25. vMAGIC—Automatic Code Generation for VHDL [72];
26. Object-Process Methodology (OPM) vs. UML - a Code Generation Perspective [73];
27. An MDE Approach for Automatic Code Generation from UML/MARTE to OpenCL [74];
28. GUI code generation for Android applications using a MDA approach [75];
29. Automatic code generation for PLC controllers [76];
30. Code generation using model driven architecture: A systematic mapping study [77];
31. Automatic Code Generation for Language-Learning Applications [78];
32. Modeling and automatic code generation for wireless sensor network applications using model-driven or business process approaches: A systematic mapping study [79];
33. IP reuse in an MDA MPSoPC co-design approach [80];
34. A co-design approach for embedded system modeling and code generation with UML and MARTE [81];

35. Experimental validation of nonlinear MPC on an overhead crane using automatic code generation [82];
36. Modeling SystemC Design in UML and Automatic Code Generation [83];
37. Automatic code generation of convolutional neural networks in FPGA implementation [84];
38. MDABench: Customized Benchmark Generation Using MDA [85];

2.7 Análise Comparativa

Para ter uma melhor perceção da comparação dos resultados encontrados, foi feita uma tabela que indica para cada artigo que tipo de modelo (se possível) é que utilizavam para a geração de código, o que seria utilizado para a transformação, a linguagem que seria gerada assim como a especificação do que poderia ser gerado. Também foi proposto indicar na tabela se a geração de código era capaz de gerar código executável sem a intervenção do desenvolvedor e para cada artigo também indicar se era proposto desenvolver uma ferramenta de geração de código. Para tal foi obtida a seguinte tabela:

Tabela 1. Tabela de Comparação dos Resultados

Artigo	Modelação	Transformação	Linguagem	O que é gerado?	Intervenção do programador (S/N)	Tem ferramenta de geração de código (S/N)
[26]	Diagrama de classes e Casos de utilização	xGenerator	Java	Aplicações Web MVC. O PIM, PSM e o Código são gerados automaticamente com "xGenerator"	N "MVC"	S
[27]	Diagrama de classes	Acceleo	C# XAML	Aplicação para o Windows Phone que realiza as operações CRUD na base de dados	N	N
[28]	O sistema é modelado usando diagramas de atividade e de sequência	AutoKode tool	Java	Classes e métodos para uma operação ATM "AutoKode tool"	S "C"	S
[29]	Diagramas UML	XLS Macros	C#, VB	Interface com utilizador para sistemas MES	N	N
[30]	Modelo UML	Model Mapping Rules	Java	Ficheiros em Java	N	N
[31]	Casos de utilização	MIDAS-CASE	Web Service Description Language (WSDL)	Web Services para diferentes tecnologias	N	N
[32]	Diagrama de Classes, Casos de utilização e diagrama de sequência	Model Driven Code Generator	VB	Gera classes incompletas para a linguagem VB	S	S

[34]	Os artefactos gerados estão implicitamente relacionados a uma tarefa	Visual XML	Java	Código de uma classe em Java para a conexão de uma base de dados	S	N
[37]	Máquina de estados UML	SMConverter	Java	Classes de estado, de evento e de contexto em Java	S "C"	S
[36]	Diagrama de classes e de sequência	GenCode	Java	Código estrutural e comportamental em Java	S	N
[35]	Diagramas UML	GenERTiCA	VHDL	Classes em VHDL	N	N
[38]	Diagramas de classe e diagrama de sequência	UJECTOR	Java	Código executável em Java	N (C)	S
[39]	Modelo UML (diagrama de classes)	CASE Tool	IEC 61131-3 code	Gera código IEC 61131-3 que consiste em ST e SFC	N	N
[40]	Diagrama de classes UML e Modelo IFML para a interação com o utilizador	PhoneGap e AppCelerator	Objective C, Java	Geração de aplicações móveis nativas	N	N
[47]	Design da interface de utilizador	Rapid Project Builder	C++, Java	Gera código Java através da interface do utilizador criada (RPB - Rapid Project Builder)	S "MVC"	S "Rapid Project Builder (RPB)"
[44]	Análise implícita de tarefas	Matlab Simulinks	C	Drivers para um conjunto de microcontroladores	N	N
[45]	Análise de problemas complexos implicitamente relacionados a uma tarefa	Eclipse Rodin plugins	C, C#, Java, C++	A tradução da especificação Event-B para uma linguagem de programação	N	N
[46]	Implícito no modelo da aplicação	TLC Scripts	Código para TinyOS, MANTIS	Aplicação WSN	N	N

Posteriormente foram selecionados aqueles artigos que supostamente tinham uma ferramenta de geração de código desenvolvida originando um total de 5 artigos. O objetivo da seleção destes artigos foi verificar que tipo de estrutura de código é que geravam, se gerava uma interface, uma lógica de negócio ou se gerava dados, com a utilização das siglas M (dados), V (interface) e C (lógica de negócio). Destes 5 artigos, 3 conseguiram gerar uma lógica de negócio ("C" na tabela 1) e 2 fizeram referência a geração de uma estrutura completa (interface de utilizador, lógica de negócio e dados ("MVC" na tabela 1), originando uma participação limitada de 11% do total (2 em 18).

Numa visão geral dos resultados, é relevante que 72,22% (13 em 18) das abordagens utilizem UML, dando ênfase a importância da UML, neste caso visando a geração de código, uma proporção que é inclusive superior a 44,44% (8 em 18) das abordagens que usam MDA. Do total, também uma parte relevante, 38,89% (7 em 18) usam tanto UML quanto MDA, e apenas uma fração menor, 22,22% (4 em 18), não usa MDA nem UML.

2.8 Conclusão

Foi realizada uma pesquisa para entender melhor alguns conceitos relacionados com a geração de código, com o Unified Modeling Language (UML) e com o Model Driven Architecture (MDA). O objetivo da pesquisa destes conceitos foi para tentar perceber melhor alguns conceitos que pudessem aparecer durante os artigos relacionados que aparecessem utilizando o método de pesquisa utilizado e referido com mais detalhe na subseção 2.4 "Método de Investigação".

O objetivo do método de investigação utilizado foi descobrir ferramentas de geração de código, se possível com o sistema a funcionar, com base na modelação de tarefas, preferencialmente de processos de negócio.

Foram encontrados no total 80 artigos, utilizado o método de investigação, no qual 19 foram repetidos utilizando as frases de pesquisa, 1 já tinha sido encontrado antes de efetuar a pesquisa, 38 não foram analisados (excluídos), 22 artigos foram incluídos e 4 destes 22 não foram analisados porque já teriam sido encontrados artigos mais recentes dos mesmos autores ou do mesmo tema.

Com os artigos pesquisados foi possível verificar que a geração automática de código é um tema que está a ser melhorado ao longo dos anos, com a geração de cada vez mais linguagens e com cada vez mais métodos gerados. Alguns dos trabalhos encontrados são para transformar linguagem de modelação em linguagem natural, outros para gerar linguagem de programação através de uma expressão lógica. Existem outros trabalhos que estão mais relacionados com a geração automática de código com o objetivo de melhorar a construção de aplicações móveis. Muitos dos trabalhos dão destaque a geração de código através de um modelo e apenas dois dos artigos encontrados referem a possibilidade de gerar a lógica de negócio, interface de utilizador e base de dados, no entanto os que desenvolvem a ferramenta de geração de código precisam de alguns ajustes no código por parte dos desenvolvedores.

Para este tema seria importante realizar uma ferramenta que conseguisse gerar código automaticamente através de um modelo de negócio, mas que não fossem necessários ajustes por parte do desenvolvedor. Esta ferramenta deveria ser capaz de gerar software automaticamente.

3. Descrição do Problema

O estabelecimento de um processo de desenvolvimento genérico e que permita a implementação de um sistema de informação baseado numa arquitetura de software também bem definida e com sucessos recorrentes, ainda não foi alcançado na engenharia de software [2], [4], [5]. A inconsistência entre a análise de requisitos de desenvolvimento de software e o desenho de software, tem sido um problema durante a fase de implementação de um projeto, nomeadamente na perspetiva de modelação de negócios e software e rastreabilidade de implementação [1], [86]. Este problema faz com que a fase de desenvolvimento de um projeto, seja um processo mais demorado, na medida em que os modelos poderão sofrer muitas alterações, visto que os requisitos poderão sofrer alterações ao longo do tempo.

Tendo em conta que os requisitos de um projeto muitas vezes estão mal definidos ou não foram bem analisados ou poderão ainda ter a necessidade de sofrer alterações, a modelação do processo de negócio também terá de ser alterada e à medida que isto ocorre é necessário alterar o código manualmente, pois o que já tinha sido implementado também terá de ser modificado devido a uma possível mudança no funcionamento do sistema.

O problema destas várias mudanças na implementação, é que, por vezes, a correção de erros não é assim tão simples, podendo fazer com que as alterações por vezes tenham um custo elevado, pois por vezes estes erros não são fáceis de corrigir, e os programas raramente funcionam sempre com sucesso, sendo necessária a alteração do código para a resolução dos erros [2].

O processo de desenvolvimento de algumas componentes de software na Universidade da Madeira envolve um processo que é baseado em abordagens como o Goals Approach, modelação Wisdom e *framework* Hydra, que se apresentam ainda nesta secção.

Para este processo de desenvolvimento, primeiro, são analisados os requisitos do processo de negócio e, depois da análise, é realizada a sua modelação tendo em conta o modelo *Goals Approach* e o método Wisdom (método utilizado para fazer a modelação do processo de negócio). Esta modelação é realizada de uma forma informal ou documental (sem a utilização de nenhuma ferramenta de modelação) servindo apenas para depois conseguir saber melhor quais os conceitos e o funcionamento que posteriormente deverá ser implementado.

Após a modelação, é realizada a implementação do código a ser utilizado pelo *Hydra Code Generator*, em python de forma manual e numa estrutura específica. Este código implementado é posteriormente utilizado pelo *Hydra Code Generator* para fazer a geração de ficheiros numa estrutura da *framework* Hydra, de forma a conseguir implementar as interfaces de utilizador, lógica de negócio e base de dados.

O problema que ocorre durante este processo de desenvolvimento é que por vezes os requisitos do processo de negócio necessitam de ser alterados e por sua vez a modelação do processo de negócio pode ser bastante alterada envolvendo a alteração do código que é implementado manualmente para o *Hydra Code Generator*, e por vezes esta alteração poderá levar a um tempo muito mais alargado e imprevisto para o desenvolvimento de software.

Nesta secção, depois da descrição do problema proposto por este projeto, é possível encontrar também a descrição de algumas das ferramentas que poderão ser utilizadas durante a fase de desenvolvimento de software como o Wisdom, Goals Approach e a *framework* Hydra, uma análise do ao processo de negócio utilizado pela Universidade da Madeira, assim como uma conclusão com uma pequena reflexão sobre as limitações e o problema já referido anteriormente.

3.1 The Goals Approach

A modelação dos processos de negócio segue o modelo Goals Approach, que especifica o caso de utilização essencial como uma tarefa completa do utilizador, que é compatível com a definição de caso de utilização do Wisdom e de Espaço de Agregação de Hydra.

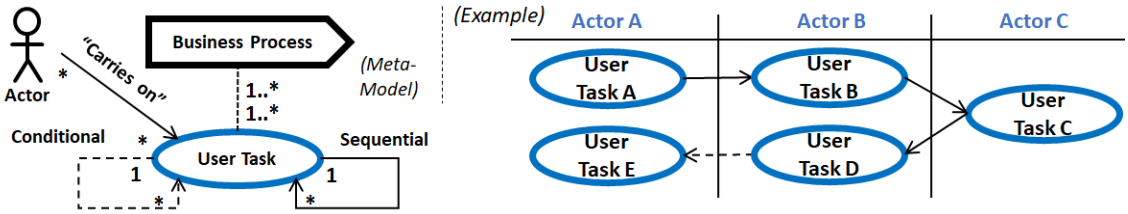

Figura 6. Modelação de processos de negócio com base no modelo Goals Approach

A Figura 6 apresenta a modelação de um processo de negócio, e também permite a identificação dos tipos de classes utilizados. São utilizados três tipos de classes de modelação relevantes, como o tipo de classe caso de utilização essencial representado com um estereótipo oval azul, o tipo de classe utilização representado por uma linha inteira ou tracejada, e o tipo de classe ator que é representado com texto no topo de uma *swim-lane* ou como um estereótipo de utilizador UML conectado a um caso de utilização com um estereótipo de utilização. Cada caso de utilização será então modelado com base no modelo Wisdom como pode ser observado na Figura 7.

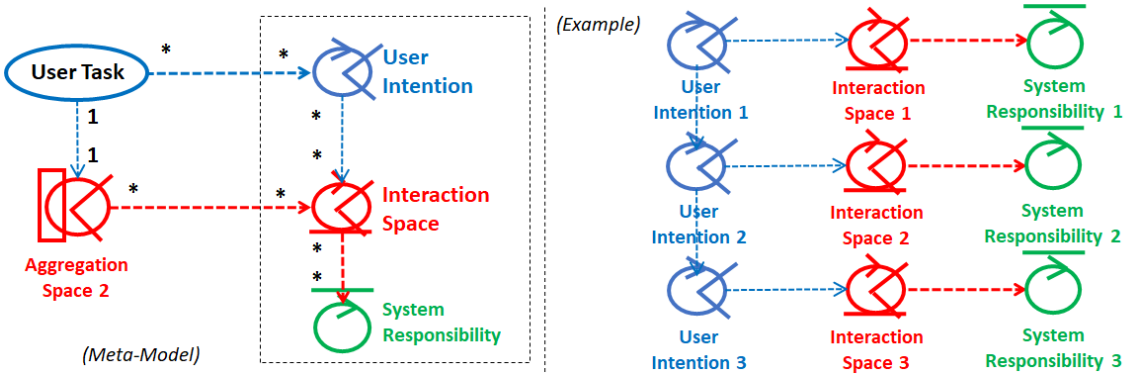

Figura 7. Modelação de casos de utilização com base no modelo Wisdom

Cada caso de utilização é detalhado tendo em conta as intenções do utilizador com um diagrama de atividades, e cada intenção do utilizador é relacionada ao Espaço de Interação onde ocorre. Cada Espaço de Interação, por sua vez, utiliza uma responsabilidade do sistema que é responsável por enviar e receber os dados entre o Espaço de Interação e a base de dados.

Com a modelação de cada caso de utilização é possível obter informações para realizar o *design* da Interface do utilizador e é possível definir a perspetiva de um ator específico em relação a um Espaço de interação, que tem uma relação direta de um para um com o Espaço de Agregação, um componente específico de software.

3.2 Wisdom

O Wisdom é um método de desenvolvimento de aplicações interativas utilizado em engenharia de *software*, que assim como o *Goals Approach*, utiliza e estende o *Unified Modeling Language* (UML), para suportar técnicas de interação humano-computador, ajudando também a

especificar, visualizar e documentar os artefactos do desenvolvimento do projeto. É um método desenvolvido para ajudar a reduzir problemas no processo de desenvolvimento do software e facilitar as fases de análise e desenvolvimento da arquitetura [7]. O modelo Wisdom é utilizado pela *Goals Approach* para modelar processos de negócios.

O Wisdom é constituído pelas componentes: processo, arquitetura e notação.

O processo é baseado num modelo de protótipo evolutivos como é possível verificar na figura 8, adaptado para empresas pequenas com equipas com uma iniciativa rápida, flexibilidade e boa comunicação.

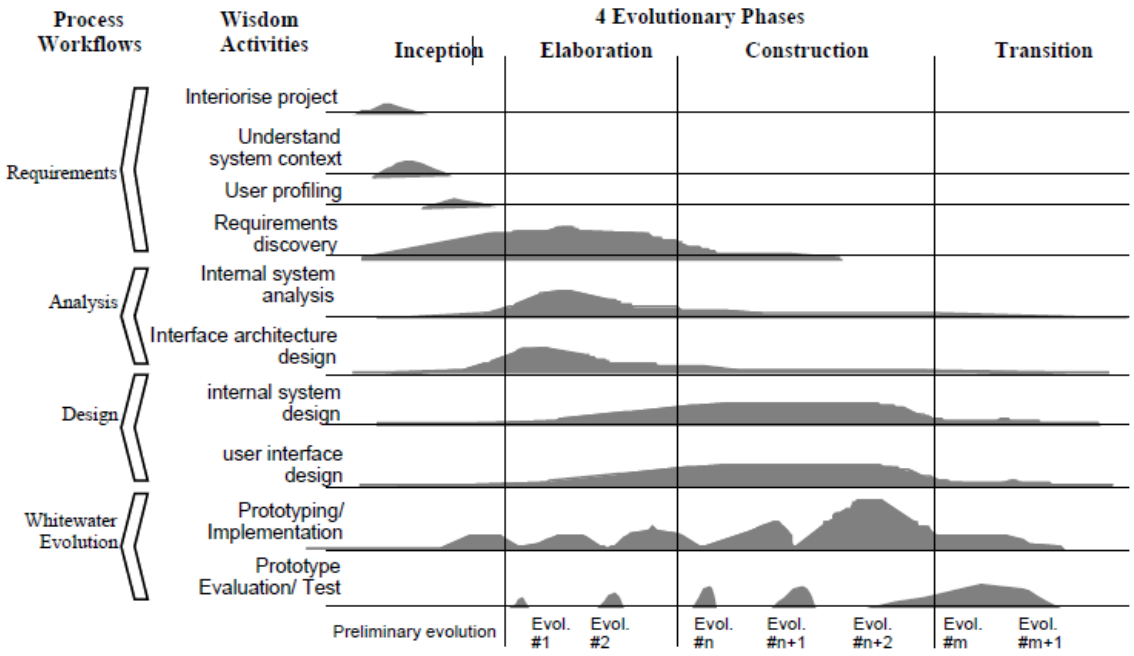

Figura 8. Processo Wisdom[7]

A arquitetura define os modelos UML para as fases de desenvolvimento do processo Wisdom, usados para a criação de sistemas interativos.

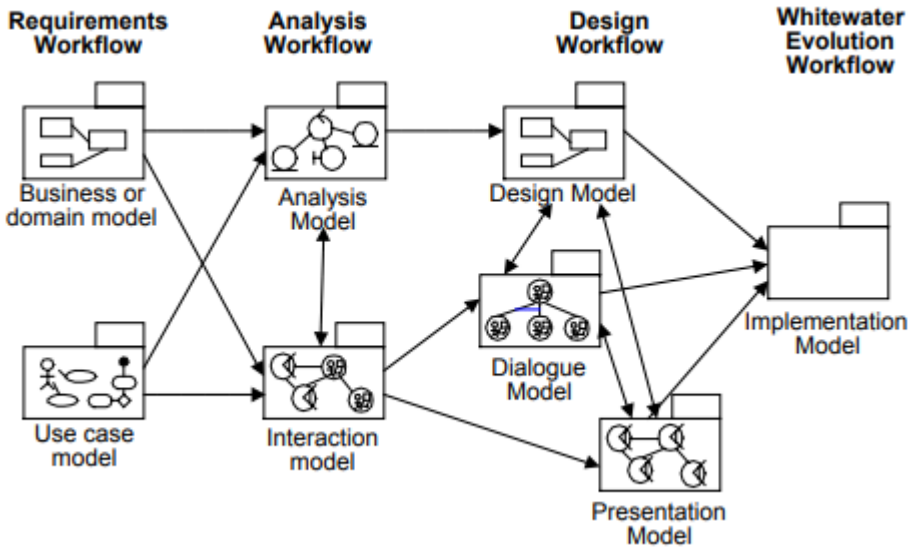

Figura 9. Arquitetura do modelo Wisdom[7]

Como é possível verificar na Figura 9, são propostos sete modelos no Wisdom, que suportam as diferentes fases de desenvolvimento do seu processo. O Modelo da implementação é excluído representando todos os artefactos gerados para implementar o sistema [7].

A Notação é um conjunto de notações desenvolvidas para a extensão do UML adaptadas ao desenvolvimento de sistemas interativos, sendo constituída por um conjunto de extensões como pode ser visualizado na Figura 10.

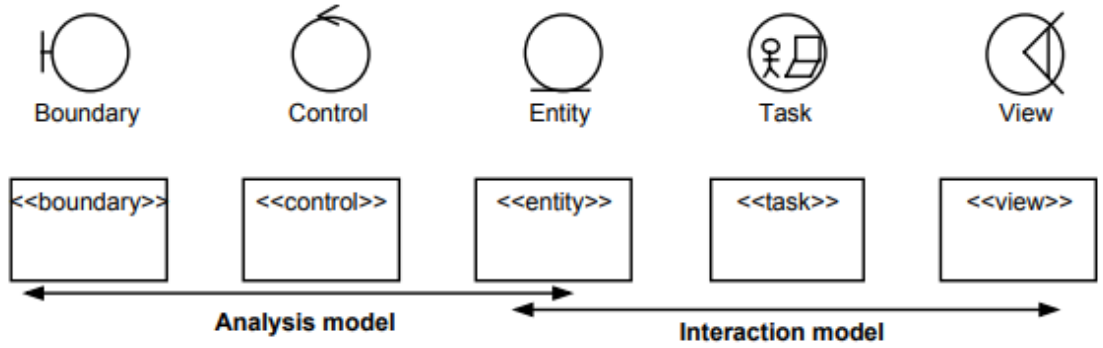

Figura 10. Extensões Wisdom para o modelo de interação e análise[7]

- A classe B*oundary* normalmente representa sistemas externos, podendo representar por exemplo a utilização de uma API de outro sistema.
- A classe *Control* representa coordenação, transações e o controlo dos outros objetos. Poderá servir para criar a ligação entre a *Task* e a *Entity*, através de uma consulta a base de dados por exemplo.
- A classe *Entity* normalmente representam uma estrutura lógica de dados.
- A classe *Task* normalmente representa de que forma a que o utilizador poderá interagir com o sistema, podendo representar todas as atividades que o utilizador pode fazer no sistema. Esta *Task* pode ser visualizada no diagrama de casos de utilização que envolve os utilizadores e as suas atividades no sistema.
- A classe *View* é, normalmente, utilizada para representar a interação entre o sistema e o utilizador.

3.3 Hydra Framework

A Hydra é uma *framework* criada com base no método Wisdom e em padrões de interação para automatizar a camada de apresentação com o utilizador e dar suporte ao desenvolvimento web, sendo implementados mecanismos de interface com o utilizador. É uma *framework* que facilita o desenvolvimento de aplicações porque permite a implementação através da parametrização para operações CRUD (Create, Read, Update e Delete), estruturas de apresentação como listas ou coleções de dados, navegação entre módulos, Espaço de Agregação e Espaços de Interação, troca de dados e interligação entre espaços de interação. O Hydra também oferece serviços de suporte para a gestão de ligações a base de dados, ligações a *Webservices*, gestão de sessão e de autenticação, gestão de menus da aplicação gestão de *logs*, ligação a ferramentas externas de "*reporting*" e regras de bloqueio de campos [3].

A *framework* Hydra foi inicialmente concebida apenas para a implementação da interface de utilizador, mas, através da repetição dos mesmos padrões de implementação para lógica de negócios e base de dados, esses padrões também se tornaram uma extensão da *framework*, e neste momento, a lógica de negócios e a base de dados apresentados neste artigo são considerados, como também parte de Hydra.

3.3.1 Estrutura da Interface do Utilizador

A estrutura da interface de utilizador da framework Hydra é composta por Espaço de Agregação (Aggregation Space), Espaço de Interação (Interaction Space), e Elementos do Espaço de Interação (Interaction Objects/Objeto de Interação).

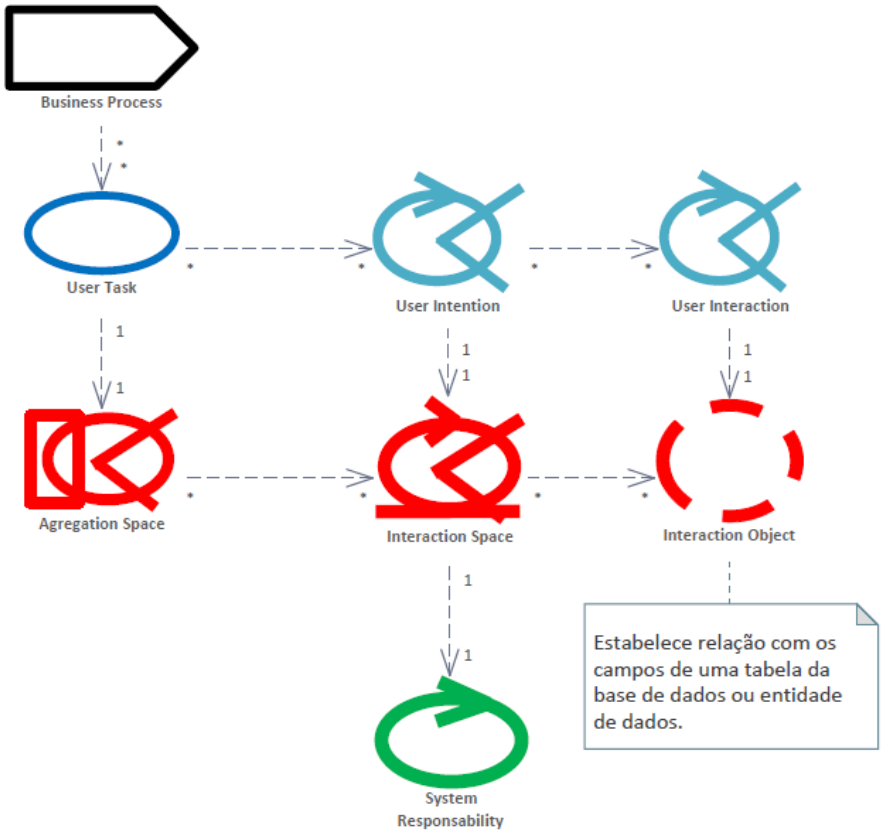

Figura 11. Relação entre a modelação do processo de negócio e os elementos da interface de utilizador da Hydra

Na Figura 11 é possível verificar a relação existente entre a modelação, realizada com base no Goals Approach e Wisdom, através de tarefas do utilizador (de *User Task*) e intenções do utilizador (de *User Intention*), e os elementos da interface de utilizador: Espaço de Agregação e Espaço de Interação, respetivamente. Cada processo de negócio (de *Business Process*) é constituído por tarefas de utilizador. Cada tarefa do utilizador será constituída por intenções de utilizador que por sua vez serão constituídas por interações de utilizador. A cada tarefa de utilizador corresponderá um Espaço de Agregação da Hydra, a cada Intenção de utilizador corresponderá a um Espaço de Interação, e a cada Interação do Utilizador corresponderá um Objeto de Interação. Estes Objetos de Interação irão permitir estabelecer uma relação com um campo da Entidade de Dados, através de uma fonte que irá conter o nome da tabela (entidade de dados) da base de dados.

Cada Espaço de Interação será suportado por uma Responsabilidade do Sistema que inclui os procedimentos de escrita e de leitura da lógica de negócio, objetos que neste contexto, também são considerados *framework* Hydra, o que será possível verificar com mais detalhe na próxima subsecção “Lógica de negócio e base de dados”.

3.3.2 Lógica de Negócio e Base de Dados

Com o desenvolvimento continuado para a *framework* Hydra criaram-se estruturas de lógica de negócio e base de dados que incluem os aspetos que é possível verificar na Figura 12.

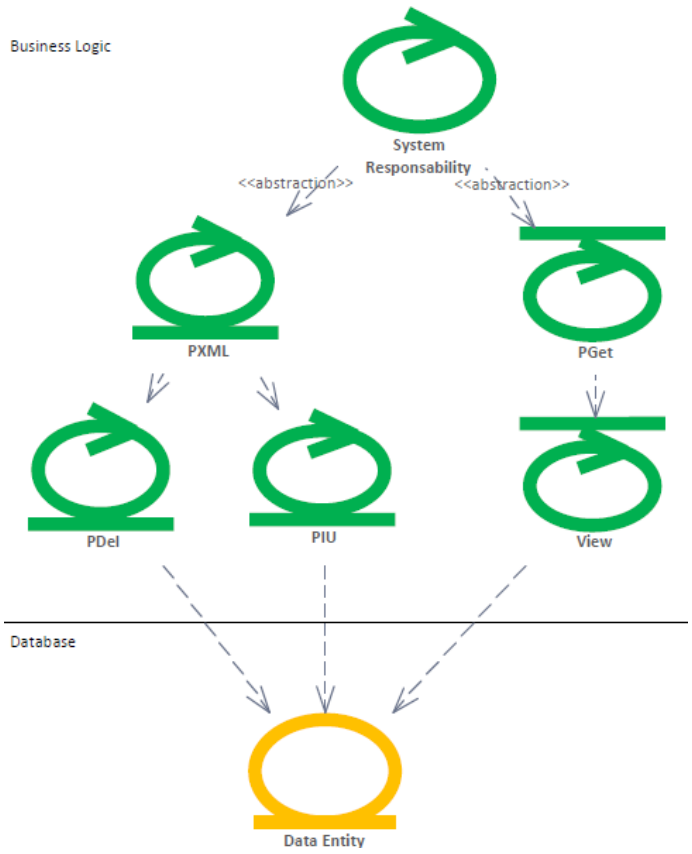

Figura 12. Lógica de negócio e base de dados da Hydra

Na Figura 12 é possível identificar objetos da lógica de negócio e base de dados. A Responsabilidade do Sistema (de *System Responsability*) que também já foi referido na Figura 11, é uma abstração dos procedimentos de escrita (PXML, PIU e PDel) e leitura (View, PGet) da Hydra. Ainda na Figura 12 é possível verificar que existe uma relação entre estes procedimentos de leitura e de escrita com uma entidade de dados (que no atual contexto é uma tabela). Logo, a Responsabilidade do Sistema irá permitir realizar uma leitura ou uma escrita à base de dados.

3.4. Análise ao Processo de Desenvolvimento

Este processo de desenvolvimento de software foi aplicado em vários projetos sendo todos realizados na Universidade da Madeira. A Universidade da Madeira contém um processo de desenvolvimento de software no qual primeiro é realizada a modelação do processo de negócio.

Esta modelação é realizada sem a utilização de nenhuma ferramenta ficando apenas documentada de forma informal. Para a modelação são tidos em conta três níveis de abstração, no qual o primeiro é constituído por tarefas de utilizador, o segundo nível em que cada tarefa de utilizador terá intenções de utilizador e o terceiro nível em que cada intenção de utilizador terá interações de utilizador.

O código para a *framework* Hydra é implementado manualmente e definido numa arquitetura bem definida que é teoricamente baseada no modelo Wisdom, ou seja, tendo em conta a modelação do processo de negócio realizada.

Durante a realização deste processo de desenvolvimento de software, cada vez que fossem feitas atualizações nos processos de negócio devido às alterações ou atualizações nos requisitos do sistema ou alterações em alguns dos seus regulamentos, era necessário também alterar o

código utilizado para a *framework* Hydra ao nível da interface de utilizador, lógica de negócio e base de dados (MVC) e aplicar várias correções neste código. Estas modificações, dependendo da mudança da modelação do processo de negócio, muitas vezes poderiam não ser tão fáceis de corrigir, provocando um impacto negativo ao nível do tempo de implementação de software.

O tempo de implementação também poderia ser afetado por outra razão, pois em alguns casos estabelecer uma relação direta entre a modelação do processo de negócio e a implementação não é assim tão fácil. As atualizações dos processos de negócio também poderão não ser tão simples se o desenvolvimento for realizado por equipas diversas, principalmente se for modelado um processo de negócio muito complexo, pois quanto maior for a complexidade da modelação do processo de negócio mais difícil será estabelecer a relação direta entre a modelação e a implementação e também o mais provável será ter mais variedade de erros que ocorrem durante a implementação.

3.5 Conclusão

O problema que ocorre durante o processo de desenvolvimento de software é que por vezes os requisitos ou regulamentos do processo de negócio necessitam de ser alterados e por sua vez a modelação do processo de negócio pode ser bastante alterada, envolvendo a alteração do código que é implementado manualmente para a *framework* Hydra.

Este problema poderá fazer com que, por vezes a adaptação de um sistema, já implementado ou a implementar, não seja simples, podendo fazer com que estas alterações tenham um custo elevado, tal como um impacto negativo relativamente ao tempo de implementação.

Tendo em conta que o código utilizado para a *framework* Hydra envolve um processo que poderá ocupar um tempo relevante por parte do desenvolvedor, o objetivo deste projeto seria arranjar uma possível solução que pudesse ajudar a acelerar o processo de implementação de uma DSL que fosse utilizada para a *framework* Hydra, sendo esta implementação obtida com base na modelação de um processo de negócio baseado no modelo Wisdom.

Para tal, seria bom que existisse algo que fosse capaz de fazer a geração automática do código DSL após a modelação do processo de negócio de forma a permitir reduzir o tempo perdido ou utilizado para corrigir os erros de código ocorridos durante a implementação manual da DSL. Uma outra vantagem que poderá existir na integração da modelação com a geração de código é a estabilização das funcionalidades já implementadas, bem como da interpretação visual (através da modelação) dos requisitos do projeto, permitindo também facilitar o estabelecimento da relação direta entre a modelação e o código implementado para a Hydra.

4. Especificação da Solução

Nesta secção será apresentada a proposta de solução e contribuição do projeto, de que forma irá ser feita a modelação dos processos de negócio para os casos de estudo, bem como a proposta de solução para o problema, que passa pela geração de uma *Domain Specific Language* (DSL) através da modelação dos processos de negócio, que posteriormente será utilizada para gerar código para a *framework* Hydra ao nível da interface, lógica de negócio e base de dados.

A solução proposta para este projeto passa por fazer com que através da modelação de um processo de negócio se consiga obter software funcional, de forma a reduzir o problema deste projeto referido na secção anterior. Para tal será modelado um processo de negócio utilizando uma ferramenta de modelação, que consiga utilizar e estender o UML. Depois o objetivo será, através da utilização da modelação já realizada, gerar o código específico (DSL Python) que será utilizado para gerar código para a *framework* Hydra.

O código para a *framework* Hydra é gerado através de um ficheiro Python, utilizando o *Hydra Code Generator*, que por sua vez, com a solução proposta, passa também a ser gerado utilizando apenas a modelação de um processo de negócio, com o intuito de otimizar assim o processo de desenvolvimento de software.

Resumindo, o objetivo será escolher uma ferramenta de modelação, exportá-la para um ficheiro e depois fazer a interligação da modelação do processo de negócio com a geração da DSL, com o intuito de permitir acelerar o processo durante a fase do desenvolvimento de software e fazer com que que as alterações das modelações não impliquem bastante tempo imprevisto para o desenvolvimento de software.

Para a realização desta solução é necessário ter conhecimento acerca do código DSL utilizado pela ferramenta "*Hydra Code Generator"*, pelo qual é definido mais abaixo noutra secção (5.1 Ferramenta de geração de código da *framework* Hydra), e também decidir de que forma a que a informação da modelação do processo de negócio poderá ser retirada para depois fazer a transformação e conseguir fazer a geração do código (DSL) posteriormente utilizado pela ferramenta de geração de código da *framework* Hydra.

A automatização deste processo poderá contribuir para que, fazer as alterações na modelação não provoque um impacto muito grande no tempo da implementação do código, que também é um dos problemas referidos na secção anterior.

4.1 Ferramenta de Modelação e Geração de Código

A ferramenta de modelação a utilizar terá de conseguir restringir e estender a *Unified Modeling Language* (UML), e a partir da modelação de um processo de negócio, esta será exportada para um ficheiro.

Depois, o ficheiro exportado será lido e transformado para a DSL Python, que irá tornar possível, através do *Hydra Code Generator* (ferramenta já existente) gerar código para a *framework* Hydra (PHP para a Interface de utilizador e SQL Server para as tabelas de lógica de negócios e base de dados).

4.1.2 Modelação

Será efetuada a modelação do processo de negócio com casos de utilização e a modelação de cada caso de utilização como uma tarefa. Esta modelação será realizada com a utilização do UML, que será *General Purpose Language* (GPL), do projeto que depois terá de ser transformada numa DSL que consiga gerar código com a utilização da ferramenta de geração de código para a *framework* Hydra ("*Hydra Code Generator*").

A modelação do processo de negócio irá ter pelo menos 2 níveis de abstração, no qual é possível ter o primeiro nível que é constituído por apenas tarefas de utilizador e um segundo nível em que para cada tarefa é possível ter uma ou mais intenções de utilizador. Para além destes dois níveis de abstração ainda poderá existir um terceiro nível de abstração em que cada intenção de utilizador poderá ter uma ou mais interações de utilizador.

Em cada nível de abstração poderão existir palavras-chave, que serão definidas ainda neste artigo na secção de implementação, e que poderão ser utilizadas no nome definido para as tarefas de utilizador, intenções de utilizador ou interações de utilizador, para depois conseguir fazer com que ferramenta a implementar consiga alcançar autonomia e seja possível aplicar o padrão CRUD (Create / Read / Update / Delete) [87] que é implementado pela estrutura Hydra sobre o SQL Server para cada Entidade de dados (DE) gerenciada.

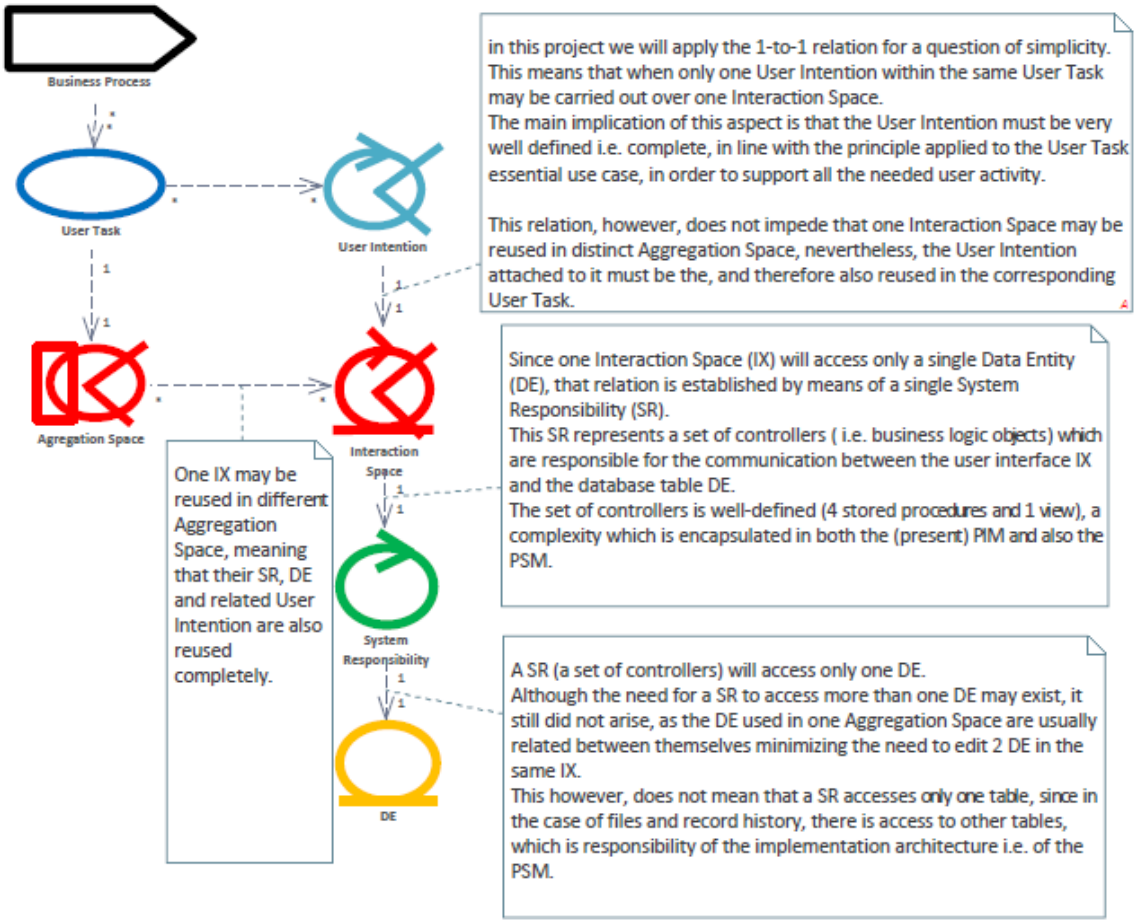

Figura 13. Meta-modelo do PIM do Projeto

Como podemos verificar na Figura 13, para cada caso de utilização podemos ter uma ou mais intenções de utilizador ou um Espaço de Agregação, que por sua vez cada intenção de utilizador terá um Espaço de Interação que por sua vez poderá utilizar uma responsabilidade do sistema.

Cada intenção do utilizador será transformada, num Espaço de Interação e cada Responsabilidade do Sistema será uma lógica de negócio (uma tabela) ou entidade de dados.

4.1.3 Geração de Código

A geração de código é feita com base no DSL (código utilizado para a geração de código para a Hydra), em que para cada tarefa de utilizador é gerado um Espaço de Agregação e para cada intenção do utilizador é gerado pelo menos um Espaço de Interação. A DSL irá gerar código

baseado na *framework* Hydra e SQL Server. Depois a *framework* irá utilizar este código para fazer a implementação da interface de utilizador, base de dados e lógica de negócio (MVC), através da utilização de ficheiros gerados pelo "*Hydra Code Generator*" que são definidos mais abaixo neste artigo na secção da Implementação (5.1).

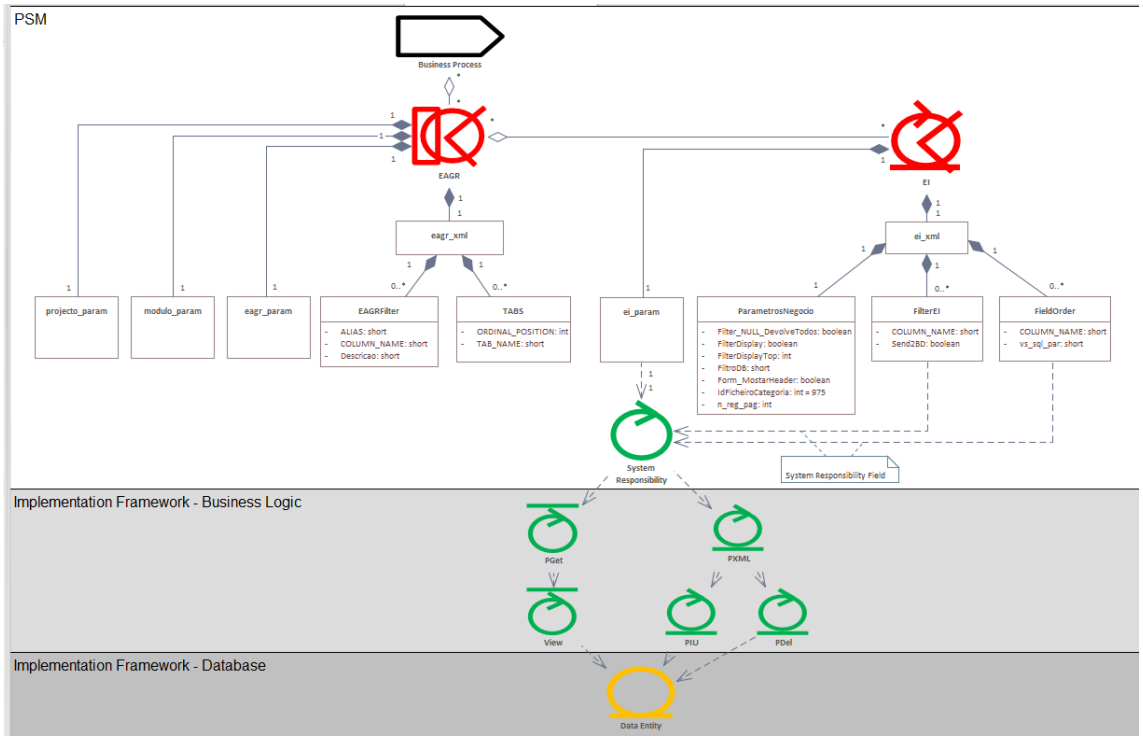

Figura 14. Meta-modelo do PSM do projeto

Na figura 14 é apresentado o meta-modelo do *Plataform-Specific Model* (PSM) onde é possível estabelecer uma relação entre "EAGR" e Espaço de Agregação e "EI" e Espaço de Interação. O Espaço de Agregação terá uma relação de 1 para 1 com o "User Task" e o Espaço de Interação terá por sua vez uma relação de 1 para 1 com o "User Intention". Cada Espaço de Interação terá uma "System Responsability" como já tinha sido referenciado na Figura 13. Esta "System Responsability" será por sua vez uma abstração dos procedimentos de leitura e de escrita da *framework* Hydra como também já tinha sido referido na Figura 12, na secção 3.3.2. Na Figura 14 também é possível verificar os elementos que são necessários definir no código específico para o Hydra Code Generator ("projeto_param", "modulo_param", "eagr_param", "eagr_xml", "ei_param" e "ei_xml") e estabelecer a relação com o EAGR e o EI. Para cada EAGR terá de ser definir um "projeto_param", um "modulo_param", um "eagr_param" e um "eagr_xml". Para cada EI será necessário definir um "ei_param" e um "ei_xml". O estabelecimento desta relação, com a solução proposta por este projeto, será feito de forma automatizada acelerando e facilitando assim o processo de desenvolvimento de software.

4.2 Implementação do DSL Generator

A solução proposta passa pela implementação de uma ferramenta que consiga gerar a DSL utilizada pelo *Hydra Code Generator* que terá como nome "*DSL Generator*". Para a implementação do *"DSL Generator"* será necessário primeiro escolher a ferramenta a utilizar para a realização da modelação e entender a estrutura do ficheiro que é possível exportar a partir da modelação. Também será necessário entender o funcionamento da estrutura da DSL

que é posteriormente utilizador pelo *Hydra Code Generator* para gerar código numa estrutura Hydra.

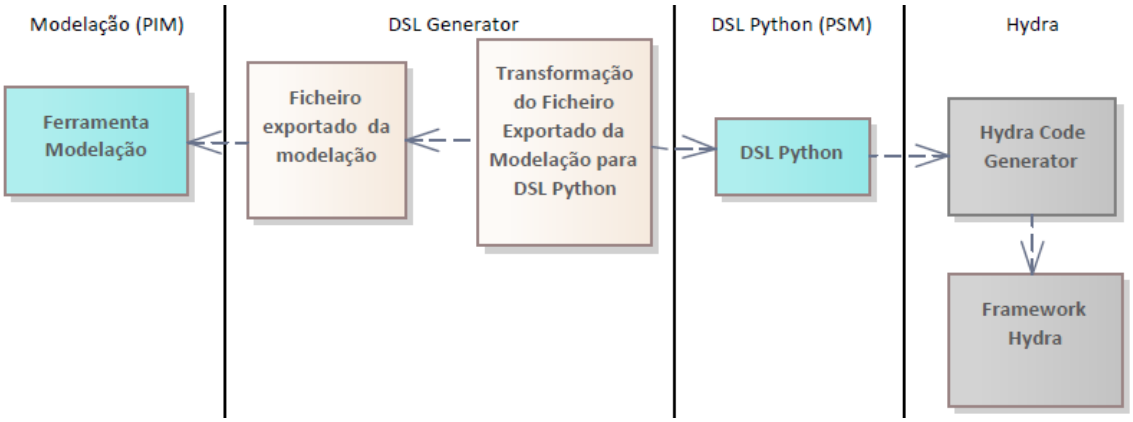

Figura 15. Solução Proposta

Na Figura 15 é possível verificar a solução proposta, que será conseguir implementar o "*DSL Generator"*. O "*DSL Generator*" deverá conseguir fazer a exportação da modelação de forma automática através da ferramenta de modelação e fazer a transformação da modelação para a DSL Python. Para tal, primeiro será necessário escolher uma ferramenta já existente que seja capaz de fazer a modelação do processo de negócio com suporte para extensões de UML, como a utilização do Goals Approach e Wisdom e que depois também seja capaz de fazer a exportação da respetiva modelação para um ficheiro. Depois da escolha da ferramenta a utilizar, o objetivo será tratar da informação do ficheiro exportado, a partir da modelação, e conseguir gerar código DSL que por sua vez será utilizado pelo Hydra Code Generator, já implementado, para fazer a geração da interface de utilizador, lógica de negócio e base de dados para a *framework* Hydra.

Para conseguir fazer a geração da DSL, será necessário entender a sua estrutura e também conseguir entender o que foi exportado da modelação, para depois conseguir tratar da informação de forma a estabelecer uma relação entre o que foi exportado a partir da modelação e o que é necessário para conseguir gerar a DSL.

A Figura 16 mostra como pode ser possível estabelecer relação entre os elementos da modelação (PIM) e do DSL Python (PSM). A solução proposta passa por conseguir fazer a transformação da modelação (com a implementação do "DSL Generator") para código DSL que possa ser utilizado para gerar para a *framework* Hydra, para que a partir da modelação do processo de negócio se consiga gerar o código DSL Python tornando o processo de desenvolvimento de software mais rápido e automático.

Como já foi referido, para fazer a transformação será necessário entender ou perceber a estrutura do DSL para depois também saber como conseguir utilizar a informação, obtida através da modelação, e implementar a geração da DSL com base na sua estrutura. Na Figura 16 também é possível verificar uma relação existente entre a modelação, realizada utilizando o Wisdom e o Goals Approach, e o código DSL que é definido para que o "*Hydra Code Generator*" consiga gerar código para a *framework* Hydra.

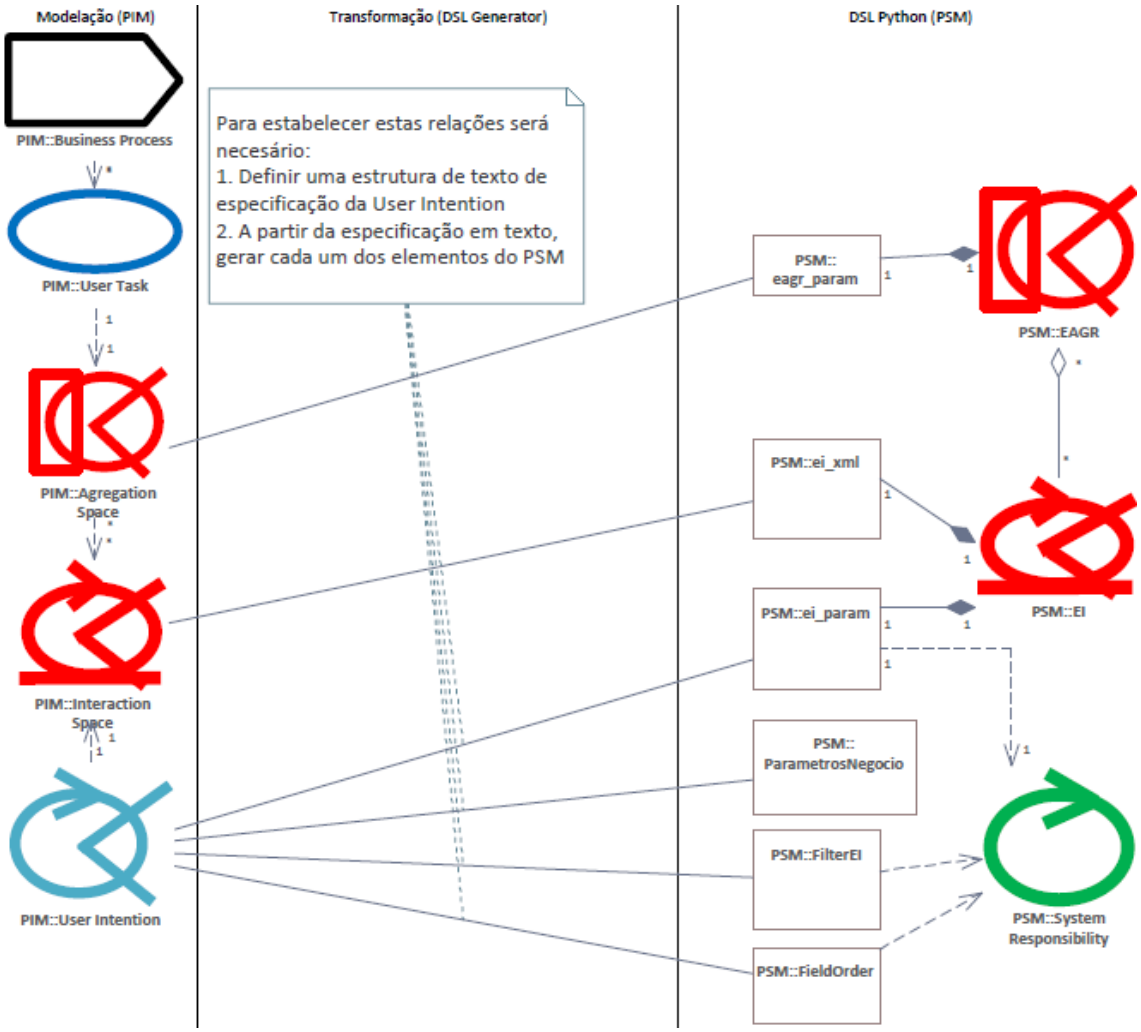

Figura 16. Transformação do PIM para o PSM

Para a transformação da modelação para o DSL é necessário entender a função de algumas das variáveis que são necessárias definir para a implementação do *DSL Generator* como "eagr_param", "ei_xml", "ei_param", "ParametrosNegocio", "FilterEI", "FieldOrder"). Atualmente esta relação é feita manualmente, no qual o programador tendo em conta os processos de negócio que são modelados e que são realizados apenas de forma documental, consegue definir através de uma linguagem DSL as variáveis necessárias para que a *framework* consiga implementar a interface de utilizador, a lógica de negócio e a base de dados. Cada processo de negócio é modelado com uma ou mais *User Tasks* sendo esta modelação realizada com base no *Goals Approach* como já foi referido anteriormente. Logo, cada vez que a modelação é alterada devido a falta de algum requisito, é necessário fazer alterações também no código implementado, podendo levar algum tempo imprevisto durante a fase de implementação de software.

A implementação do "DSL Generator" poderá permitir tornar o trabalho manual da implementação existente do DSL para uma implementação de código para a *framework* de uma forma mais rápida. Desta forma, a implementação do "DSL Generator" será a principal contribuição deste projeto, pois com esta implementação é possível estabelecer a relação direta entre a modelação e o DSL Python de uma forma automatizada facilitando assim o trabalho de implementação, potencialmente eliminando o tempo de implementação.

4.3 Conclusão

Este projeto consiste no desenvolvimento de uma ferramenta de geração de código para ajudar a acelerar o processo de desenvolvimento de software. A ferramenta seria capaz de gerar código automaticamente para a *framework* Hydra através de um modelo de negócio, sem a necessidade de trabalho manual por parte do desenvolvedor. A integração da geração de código para a *framework* Hydra visa a conceção de um sistema com a capacidade de geração automática de software funcional. A ferramenta de modelação utilizará uma GPL que será transformada em DSL que por sua vez irá gerar o código baseado na *framework* Hydra e SQL Server.

Esta ferramenta será aplicada para a implementação de um ou mais processos de negócio reais da Universidade da Madeira.

Com esta solução o objetivo é conseguir obter de forma automatizada a DSL Python para a *framework* Hydra de forma a tornar a implementação da DSL mais rápida e eficaz e permitir reduzir o número de erros ocorridos, cada vez que é alterada a modelação de um processo de negócio, que por sua vez não acontece na implementação manual.

Para tal será realizada a modelação do processo de negócio utilizando uma ferramenta que seja capaz de utilizar e estender o UML e que depois seja capaz de exportar para um ficheiro em XML a respetiva modelação, e após a modelação ser concluída e ser exportada para um ficheiro, este será lido e transformado numa estrutura DSL em Python que será utilizada pelo "*Hydra Code Generator"*, acionando a geração do código, que posteriormente será testado no envio e na consulta dos dados da base de dados com a ajuda da *framework* Hydra.

5. Implementação

Nesta secção é possível encontrar, primeiro, uma breve descrição da ferramenta de geração de código da *framework* Hydra ("*Hydra Code Generator*"), e depois, a informação relacionada com a implementação do gerador de código *Domain Specific Language* (DSL) "*DSL Generator*", o método utilizado, a descrição detalhada dos ficheiros implementados, bem como informação acerca do DSL Python gerado com o *DSL Generator*.

O *DSL Generator* inclui a modelação de processos de negócio, que poderia ser realizada com várias ferramentas de modelação, mas para este projeto foi decidido, por questões de fiabilidade e conhecimento da ferramenta pelo GDAI, o Enterprise Architect.

Para conseguir fazer a transformação foi necessário entender primeiro de que forma a que deveria ser realizada a extração do XML a partir da ferramenta de modelação, e quais seriam os dados importantes que deveriam ser extraídos para que depois pudessem ser utilizados para fazer a transformação para a DSL.

Depois de ser escolhida a ferramenta utilizada para a modelação, foi modelado um processo de negócio e depois foi realizada uma extração exploratória para um ficheiro XML desta modelação. Posteriormente, com o ficheiro XML extraído, foi utilizada a linguagem Python para fazer a transformação para DSL Python de acordo com a sua estrutura (definida na secção 5.1.1), tendo-se seguido a restante implementação, tal como apresentada no presente capítulo.

5.1 *Hydra Code Generator*

A ferramenta de geração de código tem como objetivo estruturar os conceitos do metamodelo, do Espaço de Agregação, do Espaço de Interação, da Responsabilidade do Sistema e Entidade de Dados e gerar o código correspondente para a *framework* Hydra para depois colocá-lo nos servidores SQL e PHP.

Cada Espaço de Interação é implementado com a utilização de 4 ficheiros (2 ficheiros em PHP, 1 ficheiro em XML e 1 ficheiro em HTML), cada Espaço de Agregação é implementado com a utilização de 3 ficheiros (1 ficheiro em PHP, 1 ficheiro em XML e 1 ficheiro em HTML), cada módulo é implementado com a utilização de 3 ficheiros (1 ficheiro em PHP, 1 ficheiro em XML e 1 ficheiro em HTML) e 9 pastas que incluem todos os ficheiros anteriores.

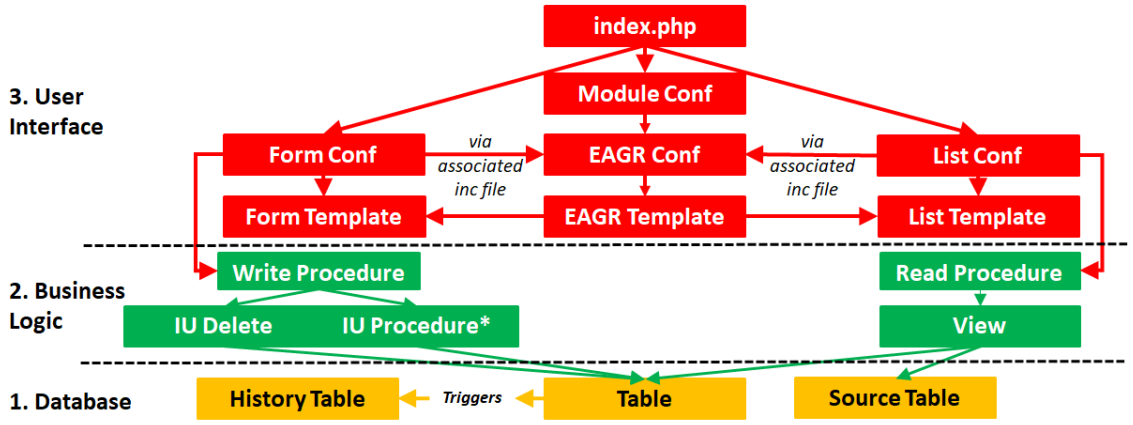

Figura 17. Framework Hydra representada em três camadas de acordo com o padrão arquitetural MVC

A Figura 17 representa o modelo da *framework* Hydra em termos de objetos onde o Espaço de Agregação é representado por "EAGR" com ficheiros "Conf" (XML) e "Template" (HTML), o Espaço de Interação é representado por "Form" e "List", sendo ambos ficheiros "Conf" e "Template", no qual estão relacionados com procedimentos da lógica de negócio "Write" (XML),

IU e “Del”, e “Read” (Objetos "Get" e "View"). Estes procedimentos são então relacionados às tabelas que representam a entidade de dados. Neste caso, os espaços de interação “List” e “Form” fornecem um suporte padrão para uma operação CRUD. Os ficheiros do módulo “Conf” (XML) e “index.php” (PHP) também são representados na Figura 17.

O *Hydra Code Generator* deverá conter uma DSL implementada em Python com conteúdo XML, que invoca procedimentos guardados no servidor SQL com o nome de “sp_GenerateCode_Modulo”, “sp_GenerateCode_Table”, “sp_GenerateCode_EAGR” e “sp_ GenerateCode” que respetivamente cria o módulo (pastas no servidor PHP dentro de um projeto onde os ficheiros Hydra são armazenados), a Entidade de Dados, o Espaço de Agregação e as responsabilidades do sistema e entidade de dados.

A DSL é implementada especificando:

- Os servidores PHP e SQL de destino incluindo os utilizadores e as senhas para poder desenvolver o código gerado (acedido pelo código Python num ficheiro separado);
- O projeto e o nome do módulo;
- O nome do Espaço de Agregação com base no nome da tarefa de utilizador;
- O nome do Espaço de Interação com base no nome da intenção de utilizador que corresponde a uma tabela (entidade de dados) da base de dados existente ou a ser criada.

O acesso aos servidores PHP e SQL de destino incluindo os utilizadores e as senhas para poder guardar o código gerado são acedidos pelo código Python estando isolados num ficheiro separado.

5.1.1 DSL Python

O código DSL Python tem a seguinte estrutura de variáveis Python:

- **projecto_param** – que contém o nome do projeto existente dentro do servidor PHP.
- **modulo_param** – que contém o nome da pasta a ser criada dentro da pasta do projeto.
- **eagr_param** – que contém o nome do Espaço de Agregação.
- **eagr_xml** – que contém o XML com os parâmetros de um específico Espaço de Agregação. Este XML tem a seguinte estrutura:
  - **EAGRFilter (COLUMN_NAME; Descricao; vs_sql_par)** - que especifica o filtro de pesquisa a ser aplicado aos Espaços de Interação para o Espaço de Agregação correspondente. A ferramenta buscará uma entidade da base de dados existente com o nome “T” ou “v” mais “COLUMN_NAME” e o campo “COLUMN_NAME”, colocará um “texto input” com “Descrição” e ordenará os dados do filtro por meio do SQL “vs_sql_par”.
- **ei_param** – que contém o nome da entidade pelo qual será gerada uma lista e formulários de Espaços de Interação para permitir as operações CRUD.
- **ei_xml** – que contém o XML com os parâmetros para a lista e formulários do Espaço de Interação e que contém a seguinte estrutura:
  - **ParametrosNegocio** - que contém:
    - **Filter_NULL_DevolveTodos**: Se o filtro não for especificado, deve retornar todas as linhas existentes;
    - **FilterDisplay:** Se for especificado, contém quantas linhas devem ser retornadas;
    - **n_reg_pag:** Se for especificado, contém o número de linhas a serem exibidas por página de lista;
    - **Form_MostarHeader:** Se deve exibir o cabeçalho do formulário ou não.
  - **FilterEI** – que contém:
    - **Send2BD**: Se filtra a entidade desse Espaço de Interação;

- **COLUMN_NAME**: composto por "entidade" mais "_" mais "COLUMN_NAME". Se "entidade" for igual a "EAGRFilter", então é um filtro do Espaço de Agregação.
- **FilterDisplay**: Se esse filtro deve ser exibido na lista e no formulário dos espaços de Interação.

o **FieldInfo** – que contém:

- **COLUMN_NAME**: campo da entidade que corresponderá ao campo do Espaço de Interação do utilizador.
- **Label**: Se for especificado, contém o SQL com a concatenação com outros dados de campos (para facilitar a interpretação do utilizador);
- **vs_sql_par**: Se for especificado, contém o SQL com a ordem dos dados a serem exibidos nesse campo específico.

CodeGeneration_MAIN.CodeGeneration_MAIN(projecto_param, modulo_param, eagr_param, ei_param, eagr_xml, ei_xml) – É o um comando que invoca a geração do código após cada "ei_xml", gerando todo o código, inclusive do Espaço de Agregação, uma vez para cada grupo de listas e formulários de Espaços de Interação. Esta função é invocada no DSL Python para gerar código utilizando a ferramenta "*Hydra Code Generator*" que por sua vez gera código para a *framework* Hydra. Posteriormente a Hydra implementa a interface de utilizador, lógica de negócio e base de dados com o código gerado.

5.2 Método de Implementação do "*DSL Generator*"

O método de desenvolvimento utilizado foi empírico, no qual inicialmente começou por uma experiência exploratória de extração da informação, mais relevante da modelação, a partir do XML. Foi desenvolvido primeiro um ficheiro Python ("XML_ExtractData_NATIVE.py") com o objetivo de extrair a informação do XML obtido a partir do diagrama, chegando-se à conclusão de que seria melhor trabalhar com o XML do que com XMI (opção padrão do Enterprise Architect para a extração da modelação), pois seria mais habitual trabalhar com XML do que com XMI, também devido a estruturação ser diferente.

5.2.1 Breve Descrição da 1º Fase de Implementação

Para a implementação do ficheiro "XML_ExtractData_NATIVE.py" foi necessário descobrir qual a melhor forma de trabalhar com o XML e qual a biblioteca a utilizar. Neste ficheiro o objetivo foi colocar em listas a informação obtida, através do XML da modelação do processo de negócio, nomeadamente "UserTasks", "UserIntentions", "UserInteractions" e "Connectors".

Posteriormente foi desenvolvido outro ficheiro Python ("XMLModelTransform.py") com o objetivo de fazer a transformação com o Python das listas obtidas a partir do XML (através do ficheiro "XML_ExtractData_NATIVE.py") para a estrutura da DSL. Inicialmente o objetivo foi terminar primeiro, pelo menos, até ao 2º nível de abstração da modelação (em que as tarefas são constituídas por intenções de utilizador) para depois passar para o 3º nível de abstração da modelação (em que as intenções de utilizador são constituídas por interações de utilizador), ou seja, na primeira fase foram definidos os "eagr_param" e o "eagr_xml" (variáveis da estrutura da DSL).

Depois de fazer uma reestruturação necessária no código, e ao passar para a transformação do 3º nível da modelação foi realizado um plano de implementação com três fases, no qual foi necessário, na primeira fase, definir os "FilterEI" (variável da estrutura do DSL) de cada Espaço de Interação. Para tal foi necessário compreender de que forma e quando é que deveria ser acrescentado um "FilterEI" em cada "ei_xml" de cada espaço de interação. Na segunda fase foi necessário definir uma linguagem que pudesse ser utilizada na modelação para a definição do nome das interações de utilizador, que é possível verificar nas tabelas 2 e 3 na secção 5.4 e 5.5,

respetivamente. A importância da definição dos nomes da interação de utilizador foi para depois conseguir fazer a definição dos "FieldInfo" (variável da estrutura do DSL), pois os "FieldInfo" dependem do nome de cada interação de utilizador definido na modelação do processo de negócio, no Enterprise Architect.

Depois de concluir estas duas fases anteriores o objetivo foi realizar a 3ºfase do plano de implementação, que foi conseguir fazer com que através de um ficheiro Python fosse possível extrair automaticamente, a modelação do Enterprise Architect para o XML (opção "native" do Enterprise Architect).

5.2.2 Breve Descrição dos Primeiros Testes

Posteriormente passou-se para uma fase de testes, no qual foi gerada uma interface parcialmente a funcionar, sendo necessário entender o que não estava correto para depois conseguir fazer alterações ao código para gerar o código DSL e colocá-lo a funcionar de forma correta com a *framework* Hydra. Depois deste teste ainda foi necessário colocar no código a definição de "FilterDisplay" no caso de um filtro poder estar presente, ou não, nas "UserInteractions".

Foi necessário também realizar alguns acertos para conseguir gerar o código na totalidade, fechando assim uma primeira fase de implementação do projeto conseguindo obter uma primeira versão de funcionamento do "*DSL Generator*".

5.2.3 Breve Descrição da 2º Fase de Implementação

Na segunda fase do projeto foram adicionadas especificações que poderiam estar no nome das "UserInteractions" na modelação (no Enterprise Architect), para dar mais opções de geração de código ao "*DSL Generator*", ou seja, esta segunda fase teve como objetivo fazer algumas melhorias ao nível da automatização da geração de código de acordo com a definição do nome das interações de utilizador definidos na modelação do Enterprise Architect, isto para conseguir obter mais opções para a geração de "FieldInfo" em "ei_xml" de um Espaço de Interação.

Posteriormente foram realizados mais testes com exemplos mais complexos de forma a tentar reduzir os possíveis bugs ainda existentes, sendo realizado ainda um exemplo teórico com "UserTasks", "UserIntentions" e "UserInteractions" genéricas.

Para finalizar a implementação foram necessários alguns acertos que ainda ocorreriam em modelações muito grandes e muito complexas.

5.2.4 Breve Descrição da Estrutura e do Funcionamento

Para a implementação foram utilizados três ficheiros Python no qual a cada um destes foram atribuídas várias responsabilidades. O "ExportToXML.py" tem a responsabilidade de fazer a exportação da modelação para XML utilizando métodos de automatização do Enterprise Architect que depois devolve como Output o XML da modelação. O "XML_ExtractData_NATIVE.py" tem como responsabilidade receber o XML e através do "xml.etree.ElementTree" (biblioteca do Python) devolver as listas dos diagramas e objetos da modelação que são posteriormente necessários para o "XMLModelTransform.py". O "XMLModelTransform.py" tem como responsabilidade transformar as listas devolvidas pelo "XML_ExtractData_NATIVE" numa DSL que posteriormente pudesse ser utilizada para gerar código para a *framework* Hydra. Esta separação de responsabilidades deve-se ao facto de posteriormente, se necessário, tornar a alteração do código mais fácil, pois para a modelação poderá ser possível a utilização de outra ferramenta diferente do Enterprise Architect e alterar o "ExportToXML.py", ou alterar a transformação apenas com a alteração do ficheiro "XMLModelTransform.py" ou ainda modificar o tratamento da informação da modelação ao alterar apenas o ficheiro "XML_ExtractData_NATIVE.py.

Para a utilização do "*DSL Generator*" é necessário colocar o ficheiro da modelação na mesma pasta onde se situa o "*DSL Generator*", porque o ficheiro XML extraído automaticamente é gerado na mesma pasta em que se situa a modelação e não é gerado na pasta do "*DSL Generator*".

Nas subseções seguintes é possível verificar com mais detalhe o que foi desenvolvido em cada ficheiro bem como o que foi realizado na segunda fase de implementação do projeto.

5.3 Ficheiro "XML_ExtractData_NATIVE.py"

Para conseguir obter XML foi necessário fazer primeiro uma pequena modelação de um processo de negócio com o nome de "Teses" com os vários níveis de abstração, sendo o primeiro nível constituído por tarefas de utilizador, o segundo nível constituído por intenções de utilizador e o terceiro nível constituído por interações de utilizador. Esta modelação foi realizada com a utilização do software "Enterprise Architect". Depois foi gerado o XML desta modelação.

Com o XML gerado, foi necessário entender e decidir de que forma a que deveriam ser extraídos os dados para uma lista em Python, que são considerados mais importantes para este projeto como por exemplo as tarefas de utilizador com os seus nomes e números de identificação (Id's).

Para a extração foi então criado um ficheiro Python com o nome "XML_ExtractData_NATIVE.py" e utilizada uma biblioteca da linguagem Python com o nome "xml.etree.ElementTree". Depois foi extraído do XML os diagramas e os seus objetos no qual cada diagrama poderá representar um nível de abstração e os seus objetos poderão representar a sua composição, ou seja, uma intenção de utilizador poderá ser um objeto de uma tarefa de utilizador ou poderá ser um diagrama composto por uma ou mais interações de utilizador.

Posteriormente, foi colocada a informação das tarefas de utilizador, das intenções e das interações de utilizador em listas que depois poderiam ser utilizadas para filtrar a informação de toda a modelação.

Depois ainda foi necessário colocar na lista a informação dos conetores para conseguir fazer a passagem de parâmetros, ou seja, saber a intenção de utilizador anterior a atual.

5.4 Ficheiro "XMLModelTransform.py"

Foi desenvolvido um ficheiro Python com o nome de "XMLModelTransform.py" com o objetivo de fazer a transformação das listas em Python obtidas a partir do XML (com XML_ExtractData_NATIVE.py) para a DSL. O objetivo foi, através das listas que continham a informação da modelação, conseguir obter um ficheiro DSL Python com a estrutura que foi mostrada anteriormente na secção 5.1.1 "DSL Python". Para conseguir obter a estrutura correta foi necessário para cada tarefa de utilizador, percorrer a lista das suas intenções de utilizador e procurar por aquela que tinha no seu nome "I Go To", isto para que depois fosse retirada a estrutura "projeto_param -> modulo_param", ou seja, no caso de existir uma intenção de utilizador com "I Go To sidoc -> investigação" no seu nome o "projeto_param" seria definido como "sidoc" e o "modulo_param" seria definido como "investigação".

Depois foi definido o "eagr_param" como sendo igual ao nome da tarefa de utilizador, e em "eagr_xml" foi acrescentado um campo "EAGRFilter" cada vez que, ao percorrer a lista das intenções de utilizador, fosse encontrada uma intenção de utilizador com "Search" no seu nome, sendo o "COLUMN_NAME" do "EAGRFilter" definido através do nome da respetiva intenção de utilizador e a "descricao" definida através da utilização do "withdescricao" no nome da respetiva intenção de utilizador. Estas variáveis anteriores (COLUMN_NAME e descricao) seriam definidas de acordo com o nome que é definido para uma intenção de utilizador no segundo nível de abstração da modelação do processo de negócio.

O “ei_param” foi definido tendo em conta o nome das intenções de utilizador com “CRUD” no seu nome (intenção CRUD) que apareciam na composição de cada tarefa de utilizador, e para definir o “ei_xml” de cada Espaço de Interação foi necessário saber se existiam intenções CRUD com o mesmo nome para depois saber se o “Send2BD” (variável definida em cada “FilterEI” da DSL) deveria ser igual a um ou a zero, isto porque poderiam existir intenções de utilizador com o mesmo nome noutra tarefa sendo necessário verificar as suas interações de utilizador que por sua vez irá definir o valor de “Send2BD”. Se o campo pertencesse à respetiva intenção de utilizador então “Send2BD” deveria ser igual a um, caso contrário deveria ser igual a zero. Para tal foi necessário também obter os campos definidos em “EAGRFilter” (através das intenções de utilizador com “Search” no seu nome) e depois verificar se cada um destes campos pertencem ou não à intenção de utilizador atual. Posteriormente, para o “ei_xml” do Espaço de Interação, os “FieldInfo” foram definidos tendo em conta o nome de cada uma das interações de utilizador que existiam na composição da respetiva intenção de utilizador com “CRUD” no seu nome (ou intenção CRUD).

Tabela 2. Especificação da DSL dependendo do nome da interação de utilizador no Enterprise Architect

Descrição da Funcionalidade	Nome da Interação do Utilizador no EA	Texto no DSL Python
Lista de seleção de um campo de uma entidade de dados já existente (do tipo “int” por predefinição)	“**Select**” [COLUMN_NAME]	<FieldInfo COLUMN_NAME = “**Id** + **COLUMN_NAME**” DateTypeLen = “**int**” />
É inserido um ficheiro a uma entidade de dados	“**Insert**” [COLUMN_NAME] **FILE**	<FieldInfo COLUMN_NAME = “**IdFicheiro** + **COLUMN_NAME**” />
É inserido um campo a uma entidade de dados que é do tipo “datetime”	“**Insert**” [COLUMN_NAME] **DATA**	<FieldInfo COLUMN_NAME = “**COLUMN_NAME**” DateTypeLen = “**datetime**” />
É inserido um campo a uma entidade de dados (do tipo varchar(200) por predefinição)	“**Insert**” [COLUMN_NAME]	<FieldInfo COLUMN_NAME = “**COLUMN_NAME**” DateTypeLen = “**varchar(200)**”/>
É inserido um campo a uma entidade de dados “varchar” com um tamanho específico	“**Insert**” [COLUMN_NAME] “**SIZE**” **SIZE**	<FieldInfo COLUMN_NAME = “**COLUMN_NAME**” DateTypeLen = “**varchar(SIZE)**” />
É inserido um campo a uma entidade de dados que não pode ser editado	“**Insert**” [COLUMN_NAME] “**READ_ONLY**”	<FieldInfo COLUMN_NAME = “**[COLUMN_NAME**]” DateTypeLen = “varchar(200)” READ_ONLY=“**1**” />
É inserido um campo obrigatório	“**Insert**” [COLUMN_NAME] “**MANDATORY**”	<FieldInfo COLUMN_NAME = “**[COLUMN_NAME]**” DateTypeLen = “varchar(200)” IS_NULLABLE=“**0**” />

Na tabela 2 é possível verificar o que deverá ser gerado para o DSL Python de acordo com o nome que é dado a cada interação de utilizador na modelação do Enterprise Architect. É possível ainda verificar as funcionalidades da interface, dependendo das palavras-chave utilizadas na modelação, para o nome da interação de utilizador, no Enterprise Architect. As interações de utilizador poderiam conter no seu nome “Select” ou “Insert”, sendo para cada um dos casos, definido o “COLUMN_NAME” do “FieldInfo” de forma diferente. No caso de a interação de

utilizador conter “Insert” no seu nome, depois também poderia conter no seu nome as palavras “SIZE”, “READ_ONLY” e “MANDATORY”, para definir o valor do tamanho do “varchar”, do “READ_ONLY” e do “IS_NULLABLE” respetivamente.

Foi necessário filtrar também do XML da modelação, a informação dos conetores, isto para saber a ordem em que as intenções CRUD aparecem na modelação do processo de negócio. Esta informação foi importante a partir do momento em que era preciso obter do Espaço de Interação atual os parâmetros do Espaço de Interação anterior e tal só era possível ao saber qual a intenção CRUD anterior à atual.

5.5 Ficheiro “ExportToXML.py”

Posteriormente também foi necessário utilizar mais um ficheiro ou classe em Python com o nome de “ExportToXML.py” de forma a automatizar o processo de exportação da modelação para um ficheiro XML, isto para depois este ser utilizado pelo “XML_ExtractData_NATIVE.py” anteriormente referido. Para tal foi decidido utilizar um método de automatização do Enterprise Architect já existente com o nome de “ExportPackageXMI” cujo objetivo deste método foi fazer de forma mais automática a exportação para XML. Para a utilização deste método foi necessário utilizar a biblioteca Python com o nome “win32com.client” e depois com a utilização do “dispatch”, que é uma função da biblioteca anterior, foi possível se conectar ao Enterprise Architect e depois ter acesso às suas funções de automatização.

Para a exportação é realizada a verificação da existência de algum projeto aberto no Enterprise Architect e no caso de existir um projeto aberto é utilizada a função de automatização necessária para a exportação do ficheiro XML. Para a utilização desta função foi necessário, descobrir qual era o Id do “Package” e saber qual era o número da opção que seria igual a “*Native*”, para que esta realizasse a exportação para XML. Para saber o id do Package foi necessário utilizar a função de automatização com o nome “EnumPackages” para devolver o xml com o Id do Package e para utilizar esta função foi também necessário descobrir o id do projeto, que foi possível obter a partir da função de automatização “EnumProjects” que devolve o XML com o id do projeto.

A partir deste momento já foi possível conseguir obter a informação necessária para utilizar a função “ExportPackageXMI” e fazer a exportação do XML do respetivo package. Depois da execução deste ficheiro Python, era então criado um ficheiro XML (“nativeXML.xml”) na mesma pasta onde se encontrava o projeto EAPX do Enterprise Architect. Caso não existisse um projeto aberto, depois de executar o ficheiro Python este não fazia nada.

5.6 Estrutura do “*DSL Generator*”

Para obter uma melhor organização e perceção do código foram definidas funções Python de forma a reduzir e organizar (modularizar) o código, para depois, se for necessário, ser mais fácil efetuar alterações (modificabilidade).

O Diagrama da Figura 18 representa os ficheiros envolvidos no desenvolvimento do “*DSL Generator”*. O “*DSL Generator*” permite a geração do “DSL_Python.py” através do “NativeXML.xml” que é obtido executando o ficheiro “ExportToXML.py” (que contém as funções de automatização do Enterprise Architect) que acede à modelação do processo de negócio.

É possível verificar na figura 18 a existência do ficheiro “XML_ExtractData_NATIVE.py” que acede ao XML gerado (ficheiro predefinido como nativeXML.xml) com o “ExportToXML.py”. Depois de aceder ao XML gerado, este irá tratar da informação que pode ser retirada do XML, colocando a respetiva informação em listas para depois devolvê-las, que por sua vez depois possam ser utilizadas pelo ficheiro “XMLModelTransform.py”. O Ficheiro “XML_ExtractData_NATIVE.py” irá devolver listas com a informação das “UserTasks”, “UserIntentions”, “UserInteractions” e “Connectores”, tornando possível estabelecer a relação

direta entre a modelação e a DSL que deverá ser gerada com o ficheiro "XMLModelTransform.py".

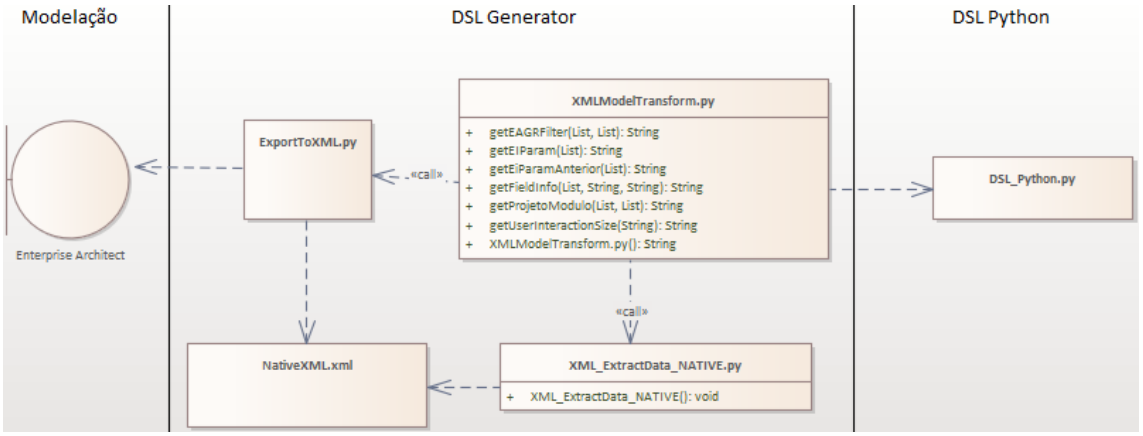

Figura 18. Diagrama representativo da implementação

No ficheiro "XMLModelTansform.py", o "eagr_param" é definido cada vez que seja encontrada uma tarefa de utilizador ao percorrer a lista "userTasks_list", isto porque, como já foi referido no diagrama do meta-modelo do PIM, cada tarefa de utilizador terá apenas um Espaço de Agregação, ou seja um "eagr_param" para cada "userTask". Como um Espaço de Agregação pode ter um ou mais Espaços de Interação e um Espaço de Interação pode ter um ou mais Espaços de Agregação, depois foi necessário verificar se existia uma intenção CRUD com o mesmo nome, pois poderia existir dois "ei_param" com o mesmo nome, mas que pertencessem a um "eagr_param" diferente. Cada "ei_param" definido corresponderá a uma "userIntention" encontrada ao percorrer a lista com o nome "userIntention_list", pois cada intenção de utilizador terá apenas um Espaço de Interação como foi definido no meta-modelo do PIM.

Foi definido ainda o método "getEIParam", que recebe uma intenção de utilizador como atributo, para que depois fosse possível definir o nome do "ei_param" para um específico Espaço de Interação. Para este ficheiro é definido um "projeto_param", um "modulo_param", um "eagr_param" e um "eagr_xml" para cada "userTask", pois como foi referido no meta-modelo do PIM cada tarefa de utilizador terá apenas um Espaço de Agregação, logo um Espaço de Agregação terá também um "projeto_param", um "modulo_param", um "eagr_param" e um "eagr_xml" como foi referido no meta-modelo da DSL.

Para cada Espaço de Interação é definido um "ei_param" e, como uma intenção de utilizador corresponderá a um Espaço de Interação, foi implementado no ficheiro para que fosse definido um "ei_param" cada vez que fosse encontrada uma intenção de utilizador na "userIntention_list". Cada Espaço de Interação tem apenas um "ei_xml" que por sua vez é constituído por "parametrosNegocio" e "filterEI" e cada "filterEI" corresponderá a uma responsabilidade do sistema e a uma entidade de dados que podem ser definidos através do terceiro nível de abstração do modelo de negócio que é constituído por interações de utilizador, isto para saber se é necessário fazer uma operação "Select" ou um "Insert" à base de dados.

A função "getFieldInfo" foi criada com o intuito de saber qual das operações ("Select" ou "Insert") irá ser necessário realizar, podendo descrever até detalhes acerca da inserção, dependendo do nome dado a uma interação de utilizador específica. Para ajudar na inserção também foi definida uma função com o nome "getUserInteractionSize" no qual tem o objetivo de verificar se um campo de interação do utilizador tem ou não a palavra "SIZE" no seu nome,

isto para que depois na inserção fosse definido o “DateTypeLen”, que como já foi referido anteriormente se não houver “SIZE” no nome o seu valor será predefinido com valor igual a 200.

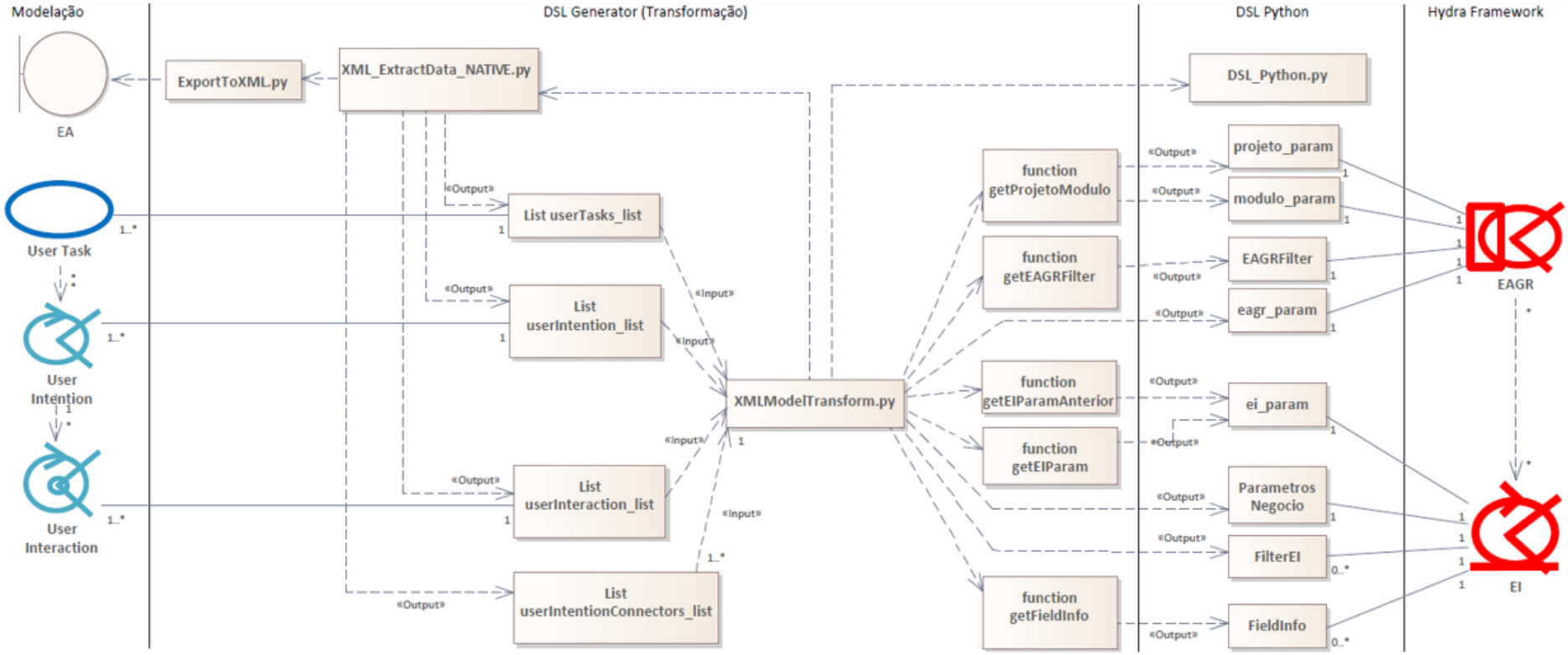

Figura 19. Diagrama geral da implementação da transformação

Com o diagrama da Figura 19 é possível verificar a relação existente entre um meta-modelo do Goals Approach e Wisdom (com "User Tasks", "User Intentions" e "User Interactions") com as listas de "Output" do ficheiro XML_ExtractData_Native.py ("userTasks_list", "userIntentions_list" e "userInteractions_list"). Com este diagrama também é possível verificar a relação existente entre as funções de "Output" do ficheiro XMLModelTransform.py ("getProjetoModulo", "getEAGRFilter", "getEIParamAnterior", "getEIParam", "getUserInteractionSize" e "getFieldInfo") e "Projeto", "Modulo", "EAGR", "EI" e "Field" da estrutura da Hydra. É possível também verificar os ficheiros que foram implementados neste projeto e que estão incluídos no DSL Generator, como os ficheiros "ExportToXML.py", "XML_ExtractDATA_NATIVE.py" e "XMLModelTransform.py". O "Output" do "XMLModelTransform.py" é o DSL utilizado pelo "*Hydra Code Generator*", que era implementado de forma manual, e com este ficheiro já é possível obter o DSL utilizado para a geração de código para *framework* Hydra de forma automatizada, melhorando assim a velocidade da sua implementação e reduzindo de erros obtidos.

5.7 Primeiros Testes com o DSL Gerado

Para testar o código foi realizado primeiro no Enterprise Architect um processo de negócio real da universidade da Madeira com o nome "Candidaturas", em que o objetivo seria gerar o DSL de forma automatizada. Nesta primeira fase de testes foi utilizado o Processo "Candidaturas" porque era o que estava a ser desenhado naquele momento, não sendo intersetado com o trabalho do processo de Candidaturas apresentado na secção 6.1, pois o "*DSL Generator*" poderia ser testado com qualquer outra modelação.

Também foi necessário verificar os nomes das Intenções de utilizador com "CRUD" no seu nome e das Interações de Utilizador que estariam na sua composição (3º nível de abstração), para verificar se já existia alguma Entidade de Dados ou campo com o mesmo nome na base de dados, para também não correr o risco de criar uma Entidade de dados ou um campo com o nome diferente, mas com o mesmo propósito na base de dados.

Posteriormente foi gerado o XML utilizando o código do ficheiro Python com o nome "ExportToXML.py" e com o ficheiro EAPX da modelação do processo de negócio. Depois foi utilizado o "XMLModelTransform.py" para gerar o DSL Python. Para este primeiro DSL criado ocorreram alguns problemas com os nomes de algumas variáveis, pois alguns deveriam ser "strings" com letras minúsculas e para outras variáveis deveriam ser retirados os espaços em branco das "strings".

Outro problema encontrado com o teste foi que os "ei_param" na DSL não ficavam por uma ordem correta, sendo necessário voltar a reformular o código para que fosse possível obter uma ordem aceitável para as intenções CRUD. Para tal foi criada uma lista com o nome "CRUDIntention_list" em que o objetivo foi percorrer as intenções de utilizador e colocar primeiro na lista os primeiros CRUD's. Depois com a ajuda dos conetores foi possível colocá-los na lista pela ordem mais correta através da criação de um ciclo, pois com a informação dos conetores seria possível saber qual a intenção CRUD anterior à intenção CRUD atual, ao percorrer a lista das intenções de utilizador. Este problema teve origem na utilização de uma variável "sequence" que supostamente corresponderia a ordenação dos CRUD's de acordo com o seu valor, porém foi descoberto que esta variável apenas os coloca pela ordem de criação no Enterprise Architect e não pela ordem correta, ou seja, se as intenções CRUD não fossem criadas pela ordem correta a lista ficaria pela ordem errada, daí ser necessário a reformulação do código para não correr o risco de as intenções estarem na DSL pela ordem errada.

Posteriormente foi necessário corrigir um problema que fazia com que houvesse repetição de "parametrosNegocio" para uma intenção CRUD, este problema teve origem na existência de

intenções CRUD iguais em duas ou mais tarefas de utilizador diferentes. Outro problema análogo também apareceu, mas foi para as interações de utilizador que eram iguais para uma intenção CRUD que repetia em duas ou mais tarefas de utilizador diferentes. Este problema não só fazia com que houvesse repetição de "parametrosNegocio" como também de "FieldInfo" para um certo "ei_xml", do "ei_param" que também repetia.

Posteriormente também foi descoberto que para funcionar corretamente seria necessário colocar o campo da intenção de utilizador em "FieldInfo" cada vez que o "Send2BD" em "FilterEI" tivesse o valor igual a 1, mesmo que o campo da interação de utilizador não fosse encontrado em nenhuma interação de utilizador de uma respetiva intenção CRUD.

Depois foi acrescentado também no caso do "Send2BD" ser igual a 1 o "FilterDisplay", que poderá ter valor igual a 0 se o campo não pertencer às interações de utilizador do CRUD atual, e poderá ter valor igual a 1 caso o campo pertença às interações de utilizador do CRUD atual. Para a resolução deste problema foi criada uma lista com o nome "CRUD_userInteractions" que contivesse todas as interações de utilizador da intenção de utilizador atual, isto para que depois fosse possível verificar se o campo, no caso do "Send2BD" ser igual a 1, pertencia ou não a lista. Se o campo pertencesse a lista "CRUD_userInteractions", era colocado o "FilterDisplay" com valor igual a 1 em "FilterEI", caso contrário ficava com o valor de "FilterEI" igual a 0.

Com estas alterações foi considerada concluída uma primeira fase do projeto, no qual foi obtido o DSL Python para a geração do processo de negócio.

5.8 Segunda Fase de Implementação do DSL Generator

A segunda fase do projeto teve como objetivo fazer algumas melhorias ao nível da automatização da geração de código de acordo com o nome das interações de utilizador localizadas na modelação do Enterprise Architect, isto para conseguir obter mais opções para a geração de "FieldInfo" em "ei_XML", ou seja, acrescentar mais "keywords" para fazer a verificação do nome das interações de utilizador, dado na modelação. Para a segunda fase do projeto também foi importado o ficheiro "ExportToXML.py" de forma a permitir a geração do XML, a partir da modelação, ao executar apenas o ficheiro "XMLModelTransform.py" sendo depois testado o seu funcionamento.

Tabela 3. Adição de Especificações do DSL dependendo do nome da interação de utilizador no Enterprise Architect

Descrição da Funcionalidade	Nome da interação de utilizador no EA	Texto que aparece no DSL Python
Seleciona um campo de uma entidade de dados específica como fonte	"**Select**" [COLUMN_NAME] "**table**" **[Entity_DATA]**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" source="**Entity_DATA**"/>
Seleciona um campo de uma entidade de dados já existente com uma largura específica	"**Select**" [COLUMN_NAME] "**width**" **[WIDTH]**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" width = "**WIDTH**"/>
Seleciona um campo de uma entidade de dados já existente do tipo "text"	"**Select**" [COLUMN_NAME] "**TEXT"**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" DateTypeLen = "**text**"/>
Seleciona um campo de uma entidade de dados já existente do tipo "tinyint"	"**Select**" [COLUMN_NAME] "**TINY_INT"**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" DateTypeLen = "**tinyint**"/>
Seleciona um campo de uma entidade de dados já existente do tipo "bit"	"**Select**" [COLUMN_NAME] "**BIT"**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" DateTypeLen = "**bit**"/>

Seleciona um campo de uma entidade de dados já existente por uma ordenação específica	"**Select**" [COLUMN_NAME] "**orderby**" **[OrderBY]**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" vs_sql_par= "**OrderBY**"/>
Seleciona um campo de uma entidade de dados já existente com um "label" específico	"**Select**" [COLUMN_NAME] "**label**" **[LABEL]**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" + label="**LABEL**"/>
Seleciona um campo de uma entidade de dados já existente com a especificação de uma ajuda (para o preenchimento desse campo)	"**Select**" [COLUMN_NAME] "**HELP**" **[Help]**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" Ajuda= "**Help**"/>
Seleciona um campo de uma entidade de dados já existente com a especificação de uma descrição	"**Select**" [COLUMN_NAME] "**DESCRICAO**" **[Descricao]**	<FieldInfo COLUMN_NAME = "**Id** + **COLUMN_NAME**" Descricao= "**Descricao**"/>
Seleciona um campo de uma entidade de dados já existente que pode não ser preenchido	"**Select**" [COLUMN_NAME] "**NULL**"	<FieldInfo COLUMN_NAME = "**Id + COLUMN_NAME**" DateTypeLen = "varchar(200)" **IS_NULLABLE="1"**/>
Faz uma inserção à base de dados de um campo que contém com uma largura específica	"**Insert**" [COLUMN_NAME] "**width**" **[WIDTH]**	<FieldInfo COLUMN_NAME = "**COLUMN_NAME**" width = "**WIDTH**"/>
Faz uma inserção à base de dados de um campo que é do tipo "text"	"**Insert**" [COLUMN_NAME] "**TEXT"**	<FieldInfo COLUMN_NAME = "**COLUMN_NAME**" DateTypeLen = "**text**"/>
Faz uma inserção à base de dados de um campo que é do tipo "tinyint"	"**Insert**" [COLUMN_NAME] "**TINY_INT"**	<FieldInfo COLUMN_NAME = "**COLUMN_NAME**" DateTypeLen = "**tinyint**"/>
Faz uma inserção à base de dados de um campo que é do tipo "bit"	"**Insert**" [COLUMN_NAME] "**BIT"**	<FieldInfo COLUMN_NAME = "**COLUMN_NAME**" DateTypeLen = "**bit**"/>
Faz uma inserção à base de dados de um campo com a especificação de uma descrição	"**Insert**" [COLUMN_NAME] "**DESCRICAO**" **[Descricao]**	<FieldInfo COLUMN_NAME = "**COLUMN_NAME**" Descricao= "**Descricao**"/>
Faz uma inserção à base de dados de um campo que pode não ser preenchido	"**Insert**" [COLUMN_NAME] "**NULL**"	<FieldInfo COLUMN_NAME = "**COLUMN_NAME**" DateTypeLen = "varchar(200)" **IS_NULLABLE="1"**/>
Faz uma inserção à base de dados de um campo do tipo "int"	"**Insert**" [COLUMN_NAME] "**INTEGER"**	<FieldInfo COLUMN_NAME = "**COLUMN_NAME**" DateTypeLen = "**int**"/>

A Tabela 3 representa a adição de algumas especificações do DSL que foram implementadas para esta segunda fase do projeto, no qual foram acrescentadas palavras-chave, que poderiam ser colocadas na definição das interações de utilizador na modelação do Enterprise Architect, como o "table", o "orderby", o "label", o "MANDATORY", o "width", "TEXT", "TINY_INT", "BIT", "HELP", "DESCRICAO", "NULL" e entre outros. Cada uma destas palavras-chave dão origem a um "FieldInfo" diferente que aparece no DSL Python em "ei_xml" de um Espaço de Interação.

O último caso representado na Tabela 3 está relacionado com a definição do "DateTypeLen" no caso de ser inteiro, este apenas deverá ser definido no caso da interação de utilizador ser ter "Insert" no seu nome porque o "DateTypeLen" para o "Insert" é predefinido como varchar(200). No caso de a interação de utilizador ser "Select" não é necessário colocar "INTEGER" porque o "DateTypeLen" para o "Select" é predefinido como int. O "SIZE", o "READ_ONLY" e o "MANDATORY" definidos na Tabela 2 também foram acrescentados no caso de a interação de utilizador ter no seu nome de modelação "Select".

Para acrescentar estas palavras-chave foram adicionadas mais quatro funções ao ficheiro "XMLModelTransform.py" com o nome "getUserInteractionWidth", "getUserInteractionTable", "getUserInteractionLabel", "getUserInteractionOrderBy, "getUserInteractionAjuda" e "getUserInteractionDescricao", com o objetivo de verificar se existe a palavra "width", "table", "label", "orderby", "ajuda" e "descrição" respetivamente, no nome definido para a interação de utilizador, na modelação do Enterprise Architect.

Posteriormente foi realizada uma modelação em Enterprise Architect com o nome "teste.eapx" apenas para efeitos de teste com objetivo de verificar ou detetar se ocorria algum erro ao gerar a DSL com o "XMLModelTransform.py", ou seja, para descobrir se existe alguma forma de obter resultados errados tentando exigir o máximo possível da implementação. Para tal foram utilizados nomes genéricos para "UserTasks", "UserIntentions" e "UserInteractions" para testar o seu funcionamento. Foi descoberto um erro que poderia ocorrer quando existia mais de três intenções de utilizador "CRUD" seguidas. Este erro fazia com que não fosse possível passar de forma correta todos os parâmetros anteriores da intenção de utilizador atual e não devolvia todos os "FilterEI" do Espaço de Interação atual com "Send2BD" de valor igual a 0. Para resolver este erro foi necessário atualizar o "*DSL Generator*", e foi decidido utilizar a recursividade na função "getParametrosAnteriores". A função que "getParametrosAnteriores" ao ser invocada devolvia a lista dos parâmetros anteriores da intenção de utilizador atual.

Posteriormente foram definidos métodos no "XMLModelTransform.py" que recebem como atributo o nome de uma interação de utilizador, fazendo a verificação da existência de palavras-chave no atributo tal como foi realizado anteriormente para as interações de utilizador.

Tabela 4. Especificação DSL dependendo do nome da interação de utilizador "CRUD" no Enterprise Architect

Descrição da Funcionalidade	Nome da intenção de utilizador no EA	Texto que aparece no DSL Python
Define se deverá devolver todas as linhas existentes da entidade de dados (deve ter valor 1 ou 0)	CRUD [nome] **DT[FND]**	<ParametrosNegocio **Filter_NULL_DevolveTodos=[FND]**
Define se deverá mostrar ou não o cabeçalho da informação (deve ter valor 1 ou 0)	CRUD [nome] **MostHead[FMH]**	<ParametrosNegocio **Form_MostarHeader=[FMH]**
Define se deverá listar o formulário (dever ter valor 1 ou 0)	CRUD [nome] **FList[FL]**	<ParametrosNegocio **Form_List=[FL]**

Define o sql que deverá ser utilizado em edição	CRUD [nome] **ED_SQL[EED]**	<ParametrosNegocio **EmEdicaoDesc_SQL=[EED]**
Define a descrição do editar no formulário	CRUD [nome] **EditarDesc[ED]**	<ParametrosNegocio **EditarDesc=[ED]**
Define o número mínimo de linhas que deverão ser exibidas no formulário	CRUD [nome] **MinLinhas[ML]**	<ParametrosNegocio **MinLinhas=[ML]**
Se for definido é uma "view" alternativa para utilizar como fonte de dados	CRUD [nome] **VAltName[VAltN]**	<ParametrosNegocio **View_AltName=[VAltN]**
Define uma descrição para um botão "new"	CRUD [nome] **BtNewDesc[BND]**	<ParametrosNegocio **bt_new_desc=[BND]**
Remove o botão "del" do formulário (deve ter valo 1 ou 0)	CRUD [nome] **BtDel[BDR]**	<ParametrosNegocio **bt_del_remover=[BDR]**
Remove o botão "new" do formulário (deve ter valor 1 ou 0)	CRUD [nome] **BtNewRem[BNR]**	<ParametrosNegocio **bt_new_remover=[BNR]**
Define um botão no formulário para guardar a informação inserida no formulário com uma descrição específica	CRUD [nome] **BtSaveDesc[BSD]**	<ParametrosNegocio **bt_save_desc=[BSD]**
Define se o formulário apenas deverá ser listado sem poder ser editado	CRUD [nome] **List_Only**	<ParametrosNegocio **List_Only="1"**
É um caso muito específico em que se for especificado é para não sobrepor a "view" (é um caso de adaptabilidade)	CRUD [nome] **ViewSkip**	<ParametrosNegocio **View_Skip="1"**
Se for especificado coloca um padrão de cores nas linhas do formulário	CRUD [nome] **NoZebra**	<ParametrosNegocio **list_zebra="0"**

Na Tabela 4 é possível verificar as palavras-chave que podem ser utilizadas na definição do nome da intenção de utilizador ao definir o segundo nível de abstração da modelação no "Enterprise Architect". Para tal foram definidas funções para fazer a verificação do nome da intenção de utilizador (uma função para cada palavra-chave).

Nesta segunda fase de implementação do "DSLGenerator", foi ainda decidido que as intenções de utilizador com "Search" no seu nome também tivessem mais palavras-chave pois só continham o "withdescricao" como palavra-chave para definir a "descricao" para um "EAGRFilter" do Espaço de Agregação da DSL Python.

Tabela 5. Especificação DSL dependendo do nome da interação de utilizador "Search" no Enterprise Architect

Descrição da Funcionalidade	Nome da intenção de utilizador no EA	Texto que aparece no DSL Python
Define qual o filtro do espaço de agregação (que deverá existir em cada Espaço de Interação)	Search **[nome]**	<EAGRFilter COLUMN_NAME=**[nome]** Descricao="">

Define o filtro com uma descrição	Search **[nome]** withdescricao **[DESCRICAO]**	<EAGRFilter COLUMN_NAME=**[nome]** Descricao="**[DESCRICAO]**">
Define o filtro com um "label"	Search **[nome]** withlabel **[LABEL]**	<EAGRFilter COLUMN_NAME=**[nome]** label="**[LABEL]**">
Define que o filtro deverá ser realizado por uma ordenação específica	Search **[nome]** orderby **[ORDERBY]**	<EAGRFilter COLUMN_NAME=**[nome]** vs_sql_par="**[ORDERBY]**">
Define qual o filtro que deverá existir para uma entidade de dados específica	Search **[nome]** withtable **[TABLE]**	<EAGRFilter COLUMN_NAME=**[nome]** SOURCE="**[TABLE]**">

Na Tabela 5 é possível verificar as palavras-chave que podem ser definidas nas intenções de utilizador com "Search" no nome definido na modelação do Enterprise Architect. É ainda possível verificar que foi acrescentado o "withlabel", "orderby" e "widthtable". O "widthdescricao" já existia na primeira fase da implementação.

Depois da implementação das palavras-chave referidas na Tabela 2, 3, 4 e 5, foi decidido ainda, nesta fase, que o DSL Generator gerasse a DSL Python em vários ficheiros sendo cada um para cada tarefa de utilizador, isto para facilitar uma possível correção de erros, pois seria mais fácil corrigir e identificar o erro na DSL tendo em conta apenas uma tarefa de utilizador no mesmo ficheiro. Para tal, o "DSL Generator" em vez de gerar um ficheiro "DSL Python.py", que continha a DSL para todas as tarefas de utilizador, passou a gerar um ficheiro para cada tarefa, sendo necessário que a tarefa seja também composta por pelo menos uma intenção de utilizador para o ficheiro ser gerado.

5.9. Conclusão

Foi implementada uma ferramenta que consegue gerar o código DSL necessário para o *Hydra Code Generator*, através da modelação do processo de negócio realizada no Enterprise Architect de forma a conseguir estabelecer uma ligação entre a modelação realizada e o código necessário para a *framework* Hydra.

Dos módulos implementados, o maior desafio deste projeto foi a implementação do ficheiro "XMLModelTransform.py" em geral, mais especificamente na utilização da informação obtida dos conectores para conseguir fazer a passagem dos parâmetros anteriores, sendo, para tal, necessário saber qual a intenção de utilizador anterior.

Uma limitação da ferramenta são os termos utilizados como palavra-chave na modelação que podem ser demasiado técnicos, porém é algo que foi necessário definir para conseguir gerar o código da forma pretendida e que pode ser melhorado com o tempo com possíveis atualizações a ferramenta.

Na próxima secção serão apresentados os casos de estudo, que por sua vez, irão servir para testar o que foi desenvolvido durante todo o projeto, apresentando em cada um deles uma modelação que depois será utilizada pelo "DSL Generator" para gerar o DSL. Também será possível encontrar resultados obtidos acerca dos testes efetuados em cada caso de estudo.

6. Casos de estudo

Nesta secção serão apresentados os casos de estudos para este projeto, no qual o objetivo será testar o DSL Generator e verificar se a ferramenta consegue fazer corretamente a transição entre a modelação e a implementação (geração) do DSL Python de forma automática, sem a necessidade de qualquer alteração por parte do desenvolvedor, e se está a contribuir para a aceleração da implementação bem como para a redução de erros.

Os casos de estudo que se seguem deverão ter como finalidade responder a uma mesma questão de investigação através dos testes efetuados em cada caso sem que o investigador tenha uma participação ativa (no caso investigado) [88].

A ferramenta já tinha tido uma primeira fase de testes, referida na secção 5.7, assim como também já foi referido na mesma secção que já tinham sido corrigidos vários erros, não sendo possível, no entanto responder, nesta primeira fase de testes, à questão de investigação uma vez que o foco foi a correção de erros.

Em cada caso de estudo irá ser possível verificar a sua descrição, a modelação e também o resultado dos testes efetuados.

Em cada caso de estudo, depois dos testes efetuados deverá ser possível responder à questão de investigação, que já teria sido referida na introdução, mas que foi reformulada de acordo com a solução implementada e apresentada na secção 5 "Implementação":

"Será possível gerar, com o "*DSL Generator*", a DSL Python totalmente correta, para a *framework* Hydra?"

Nesta secção são apresentados os seguintes três casos de estudo:

1. Processo de Candidaturas: trata-se de um processo de negócio real da Universidade da Madeira e que foi implementado para as candidaturas do ano letivo 2021/2022;
2. Processo de Correspondência: realizado para facilitar o processo de Correspondência da Universidade da Madeira de forma a evitar algumas lacunas que ocorreriam com a aplicação já existente, e;
3. Processo Teses: realizado para melhorar e automatizar o processo de submissão e acompanhamento dos trabalhos científicos para os cursos da Universidade da Madeira.

6.1 Processo de Candidaturas

O atual caso de estudo tem como objetivo contribuir para a melhoria do processo de submissão e acompanhamento de candidaturas para os cursos da Universidade da Madeira.

Este caso de estudo foi proposto para automatizar o processo de candidaturas e contribuir para o seu melhor funcionamento a eliminar o número de dúvidas dos candidatos em relação as candidaturas.

Para a realização da modelação do processo de negócio foi elaborada primeiro uma análise de requisitos para conhecer melhor o funcionamento deste processo de candidaturas da universidade da Madeira. Para tal foi necessário ter conhecimento de todos os procedimentos que ocorrem durante o processo de candidaturas através de reuniões que foram realizadas com a participação da Unidade de Assuntos Académicos (UAA) da Universidade da Madeira. Após entender melhor estes procedimentos já foi possível realizar a modelação e a implementação do processo.

6.1.1 Descrição do Processo de Negócio

Durante o processo de candidaturas, primeiro é realizada a parametrização do concurso que inclui a data de início da fase do concurso, a calendarização do concurso, a data de início e fim de cada fase do concurso e as vagas, que é uma tarefa realizada pela Unidade de Assuntos Académicos (UAA).

Posteriormente o candidato ou aluno deverá submeter a candidatura com as suas informações (identificação, morada, contacto, percurso académico e documentação) e durante esta submissão deverá ser possível visualizar o estado da presente candidatura antes da sua submissão, ou seja, deverá ser possível ir atualizando à medida que a informação vai sendo introduzida pelo aluno ou candidato.

Depois deverá ser realizado um pagamento pelo candidato para ocorrer a validação da candidatura e o candidato deverá ter opção de alterar a candidatura até a uma data final de candidatura. Dependendo do curso a ser inscrito pelo candidato, poderá ser necessário, ou não, a realização de uma prova de acesso. Posteriormente o secretariado marca um horário para as provas se existirem alunos inscritos para a realização das provas de acesso.

No caso de o candidato já ter sido aluno da universidade, a validação dos documentos já não deverá ser necessária, pois o aluno já teria a sua informação no sistema, devendo esta fase de validação ser mais rápida e automatizada.

Depois o candidato deverá receber uma notificação com a informação de que a candidatura foi aceite ou validada, mas que o candidato ainda não foi colocado. O candidato também deverá receber uma notificação no caso de ser necessário corrigir alguma informação que foi mal submetida na candidatura, tendo também um prazo para efetuar as alterações necessárias na candidatura.

Posteriormente deverão ser escolhidos docentes para fazerem parte do júri do concurso. Os docentes do júri deverão admitir e selecionar os candidatos de acordo com os critérios de acesso ou de seleção.

Após o júri submeter uma lista com a informação dos candidatos selecionados ou admitidos, seguindo os critérios de acesso, a Unidade de Assuntos Académicos (UAA) ainda poderá alertar para algum possível engano durante a seleção dos candidatos.

Posteriormente deverá ser colocada uma lista provisória dos candidatos admitidos com um prazo de reclamação antes de ser colocada a lista definitiva.

Nesta fase, mesmo que se tenha conhecimento da lista definitiva dos candidatos admitidos, ainda não é possível ter conhecimento do número de vagas que poderão abrir para a próxima fase para um curso específico, isto porque, os candidatos ainda não foram matriculados e nesta fase é apenas possível ter a informação dos candidatos que foram admitidos e que poderão ser matriculados. Só depois de ocorrer a fase das matrículas a que poderá ser possível definir o número de vagas para um determinado curso, para a próxima fase de candidaturas.

Existem duas fases para as candidaturas no qual a primeira é quando a candidatura vai ser analisada para concurso e ainda poderão ocorrer alterações da candidatura por parte dos candidatos de acordo com a validação da UAA. A segunda fase de candidaturas é quando ocorre o concurso, ou seja, quando ocorre a admissão e serialização dos candidatos realizada pelo júri do concurso.

Este processo de candidaturas atualmente ocorre apenas com o preenchimento de um formulário no qual o candidato insere os dados ou informação pedida e a informação sobre a candidatura só poderá ser vista através do edital.

Para tal seria importante haver uma interface mais intuitiva para o candidato se inscrever nas candidaturas e para facilitar o processo fazendo com que este se torne automatizado e permita reduzir as dúvidas colocadas pelos candidatos que ocorrem durante a inscrição dos cursos.

6.1.2 Modelação do Processo de Negócio

Para a implementação deste processo de negócio é realizada a modelação do processo de negócio utilizando uma ferramenta de modelação, o *Enterprise Architect* (EA), que foi escolhida para a implementação do "*DSL Generator*". A modelação do processo de negócio foi realizada tendo em conta a informação obtida sobre o que ocorre durante todo o processo das candidaturas na Universidade da Madeira, tendo em conta os requisitos e os regulamentos do sistema.

A implementação deste processo de negócio irá permitir também testar o que foi desenvolvido neste projeto, servindo para visualizar se existe mais algum erro na implementação, para depois fazer possíveis correções ao código desenvolvido.

Depois de analisar o processo de negócio "Candidaturas", foi realizada a sua modelação utilizando o EA. O resultado dessa modelação é apresentado na Figura 20 sendo possível verificar a modelação do primeiro nível de abstração do processo de candidaturas da Universidade da Madeira. Para este projeto ficou previsto a existência de um total de 6 tarefas de utilizador ("Parametrizar Curso", "Parametrizar Concurso", "Submeter Candidatura", "Validar Candidaturas", "Marcar Horário" e "Admitir e Seriar Candidatos"), no qual a tarefa "Marcar Horário", com a solução proposta por este projeto, irá continuar a ser realizado de forma manual pelo secretariado, pois a marcação de horário é realizada utilizando um programa específico e não a plataforma implementada pela *framework* Hydra.

O processo de negócio Candidaturas contém um número total de quatro atores ou utilizadores envolvidos (Unidade de Assuntos Académicos (UAA), Secretariado, Candidato que também poderá ser um antigo aluno e Júri do Concurso que será um docente da Universidade da Madeira), no qual a Unidade de Assuntos Académicos (UAA) terá como tarefas parametrizar o concurso e validar as candidaturas podendo verificar a informação submetida por cada candidato. O candidato ou aluno irá ter apenas a possibilidade de submeter a candidatura e os docentes pertencentes ao júri do concurso irão admitir e seriar candidatos de acordo com a submissão realizada por estes. A parametrização do Curso irá ser uma tarefa de utilizador realizada por um docente, no qual o docente irá ter em conta os critérios de acesso do curso.

Em cada tarefa de utilizador é modelada a sua composição (segundo nível de abstração com intenções de utilizador), ficando apenas a tarefa "Marcar Horário" sem intenções de utilizador na sua composição visto que é utilizada outra ferramenta para esta tarefa como já foi referido anteriormente.

Utilizando a modelação do processo de negócio elaborada no EA e a ferramenta "DSL Generator" implementada para neste projeto, irão ser geradas, para este caso de estudo, cinco *interfaces* utilizando a modelação do EA representado na Figura 20 (primeiro nível de abstração), sendo cada uma destas interfaces para cada tarefa de utilizador.

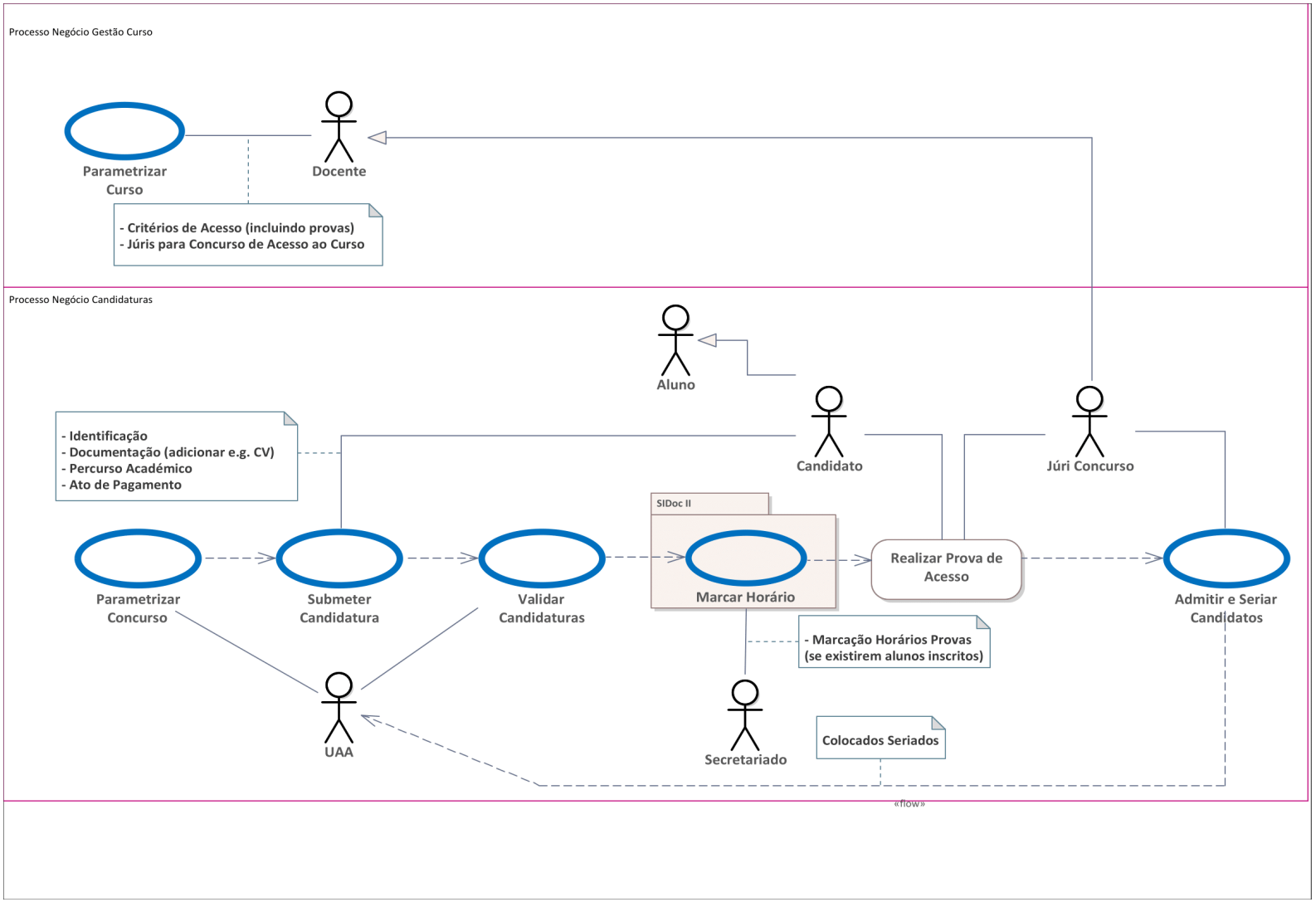

Figura 20. Modelação do primeiro nível de abstração do processo de Negócio "Candidaturas"

Na Figura 21 é possível verificar um exemplo da modelação do segundo nível de abstração para a tarefa de utilizador "Parametrizar Curso" que deverá ser realizada pelos docentes e que poderão ter acesso aos requisitos de acesso do curso pretendido através do Sidoc (Plataforma dos docentes na Universidade da Madeira). Neste nível de abstração é possível verificar a existência de intenções de utilizador com "I Go To", "Search" e "CRUD" no seu nome que são palavras-chave que deverão ser utilizadas na modelação para este nível de abstração, no qual já foi referido na secção 5.

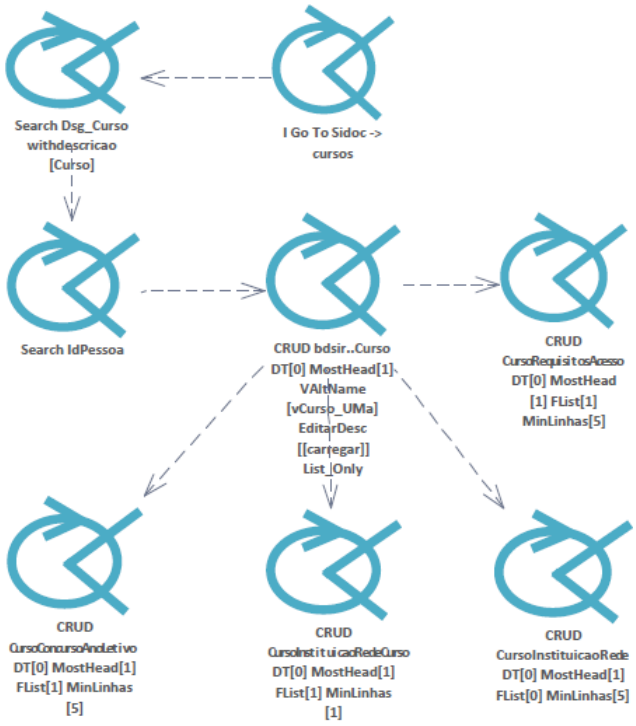

Figura 21. Modelação do segundo nível de abstração da tarefa Parametrizar Curso

Neste caso de estudo irá ser verificado se é possível gerar a DSL Python sem a necessidade de alguma correção por parte do programador. Para fazer esta verificação será realizada a comparação da DSL inicial (se possível sem a necessidade de correção) com uma DSL final, para verificar as diferenças e também para verificar se o que foi gerado está correto e não necessita de mais alguma alteração por parte do desenvolvedor. O objetivo é conseguir verificar se, através da ferramenta de geração de código (*DSL Generator*), é possível obter o DSL Python, sem nenhum erro, para a *framework* Hydra, utilizando o "*Hydra Code Generator*", de forma a reduzir o tempo de implementação e a correção de erros no código, contribuindo para apressar o processo de desenvolvimento de software.

6.1.3 Análise e Implementação

Para este caso de estudo foi possível fazer a geração de cinco ficheiros ("Parametrizar Curso.py", "Parametrizar Concurso.py", "Submeter Candidatura.py", "Validar Candidaturas.py" e "Admitir e Seriar Candidatos.py") cada um com a DSL de uma das tarefas de utilizador definidas no primeiro nível de abstração da modelação que é possível verificar na Figura 20. Não foi gerado nenhum ficheiro para a tarefa "Marcar Horário" porque esta tarefa de utilizador não contém na sua composição qualquer intenção de utilizador definida, porque para a realização desta tarefa não será utilizada a *framework* Hydra para a sua implementação.

Com os ficheiros gerados do “DSLGenerator” foi realizada uma comparação entre estes e ficheiros de DSL implementados e verificados de forma manual para garantir que cumpriam a modelação descrita. Esta comparação, neste caso de estudo, foi realizada de forma manual sem a existência de módulos de teste automático, que também poderá ser algo que pode ser implementado de futuro.

Para as tarefas “Parametrizar Concurso” e “Parametrizar Curso” foi possível gerar código quase sem a presença de erros, tendo apenas uma exceção na tarefa “Parametrizar Curso”. Esta tarefa de utilizador é composta por uma intenção de utilizador com o nome “CRUD bdsir..Curso” (que é possível verificar na Figura 21) e quando o código é gerado tem a presença de um “t” na definição do “ei_param” no DSL Gerado. No entanto é necessário ter em conta que este é um caso muito específico e que o normal será encontrar tabelas com “t” antes do nome em “ei_param”, mas como a tabela já teria sido definida sem o “t”, neste caso será necessário retirar também o “t” da definição do “ei_param” do DSL gerado. Este erro ocorreu porque o nome dado à entidade de dados não estava no formato correto, que é todas a começarem por "t", como tal será necessária à sua correção não sendo um erro importante a ser considerado.

Em relação à tarefa “Submeter Candidatura” o código que foi gerado está correto ficando apenas sem a passagem do parâmetro Candidato porque o código que foi comparado foi implementado manualmente sem ter em conta como é que ficaria a modelação e como tal, para esta tarefa foi decidido não colocar a intenção de utilizador “CRUD candidato” para a comparação. Para o restante código da tarefa “Submeter Candidatura” o código gerado ficou igual ao código implementado manualmente.

Na tarefa “Validar Candidaturas”, com o “DSLGenerator”, foi possível obter código DSL que ficou praticamente igual ao código implementado manualmente, ficando com apenas um caso em que para a intenção de utilizador “CRUD ConcursoFaseCurso” aparece em “ei_xml” um “FieldInfo” a mais. Esta exceção aconteceu devido ao facto de acrescentar uma interação de utilizador “Select ConcursoFase” na composição da intenção de utilizador “CRUD ConcursoFaseCurso” na modelação. Esta interação de utilizador foi acrescentada para que o valor do “Send2BD” para o “FilterEI” com o nome “EAGRFilter_IdConcursoFase” fosse igual a 1 (para poder filtrar o campo ConcursoFase no Espaço de Interação atual).

Na tarefa “Admitir e Seriar Candidatos” o código gerado continha alguns casos em que existe algum “FieldInfo” a mais, analogamente à tarefa “Validar Candidaturas”. Isto deve-se ao facto de que o código implementado de forma manual utilizado na comparação também não tinha sido atualizado para estas tarefas de utilizado fazendo também com que seja reduzida a importância do aparecimento dos “FieldInfo” a mais no código gerado. Em geral o código gerado para a tarefa “Admitir e Seriar Candidatos” ficou praticamente igual ao código implementado manualmente, ficando com apenas alguns casos em que existe um “FieldInfo” a mais como já foi referido.

6.1.4 Conclusões

Em conclusão, para este caso de estudo foi possível obter código com o “DSLGenerator”, podendo o código ser completamente aproveitado existindo algumas das exceções referidas anteriormente que são necessárias ter em conta e adaptar antes de ser utilizado para a geração de código para a Hydra.

Em geral, algo que também não foi tido em conta na comparação do código foi a definição do “IdProcessoNegocio” e “IdProcessoNegocioTarefa” em “ParametrosNegocio” de cada “ei_xml” de cada Espaço de Interação. Foi decidido que a definição destes dois parâmetros fosse incluída numa possível implementação futura no “DSLGenerator”, porque também não faz muito sentido a definição destes dois parâmetros ser realizada através da modelação, o que faz com

que também não seja muito importante, para já, a definição destes dois parâmetros, pois é algo que não impede a geração de código através da modelação.

Para este caso de estudo o resultado foi bastante satisfatório, pois foi possível gerar código de forma correta sem a necessidade de alterações por parte do programador, existindo apenas duas exceções referidas anteriormente que depois poderá ser necessário adaptar no código gerado.

A primeira exceção, que é necessário adaptar, é o aparecimento do "t" no "ei_param" que corresponde a intenção do utilizador com o nome "CRUD bdsir..Curso" da tarefa de utilizador "Parametrizar Curso" que é um caso muito específico referido já anteriormente.

A segunda exceção que não é necessário ter em conta, é a não inclusão da intenção de utilizador "CRUD Candidato" na tarefa de utilizador "Submeter Candidatura", que também não é necessário dar muita importância pois o código gerado foi comparado com outro que foi realizado de forma manual e não foi tido em conta como é que ficaria a modelação.

A comparação realizada para a verificação do DSL gerado foi realizada de forma manual, podendo também estes módulos serem implementados no futuro. A implementação destes módulos de teste poderá facilitar a fase de testes cada vez que sejam realizadas as possíveis atualizações ao "*DSL Generator*".

6.2 Processo de Correspondência

O caso de estudo "Processo de Correspondência" tem como objetivo contribuir para a melhoria do tratamento da correspondência na Universidade da Madeira.

Este caso de estudo foi proposto para automatizar o processo de correspondência e contribuir para o seu melhor funcionamento através da utilização da plataforma "siga" implementada com *framework* Hydra. Para a realização da implementação deste processo de negócio foi primeiro elaborada a modelação baseada no funcionamento do processo, sendo também necessário entender o seu funcionamento. Depois da realização da modelação o objetivo foi testar e verificar o funcionamento da ferramenta "*DSL Generator"* criada neste projeto e detalhada na secção 5.

6.2.1 Descrição do Processo de Negócio

A gestão de correspondência da universidade da Madeira está centralizada no Gabinete de Apoio à Reitoria, que regista as entradas e as saídas gerais, bem como o encaminhamento destas para o gabinete do reitor, bem como para os restantes organismos da universidade (gabinetes, unidades, projetos, faculdades, entre outros).

Este processo de negócio atualmente está implementado numa aplicação no qual têm acesso os catalogadores da correspondência (2 a 3 pessoas), bem como o secretariado do gabinete do reitor de forma a organizar a correspondência diária, sendo que os secretariados dos restantes organismos recebem a correspondência por email.

Porém a implementação deste processo de negócio apresenta algumas lacunas, não sendo possível corrigir dados por parte dos catalogadores através da aplicação, como editar e guardar dados.

Para tal seria importante resolver estas lacunas através da modelação de um novo processo de negócio de manutenção cuja implementação assenta sobre tabelas (ou entidades de dados) já existentes.

Durante o processo de Negócio Correspondência, o responsável pela correspondência deverá ter a possibilidade de poder visualizar os documentos, para depois poder realizar o registo. Deverá ser possível preencher um formulário com os dados da correspondência (Proveniência, Assunto, Data, Tipo, Classificação, Notas Adicionais, Correspondência Associada e Arquivo) para efetuar o registo da correspondência.

Após preencher os dados da correspondência, o responsável pela correspondência deverá também, ter a possibilidade, de fazer o encaminhamento da respetiva correspondência, indicando o seu tipo de destino e os dados da origem e do destino (pessoas e cargos respetivos).

O responsável pela correspondência também deverá ter a possibilidade de verificar o registo das correspondências, e verificar e editar os dados das correspondências que poderão ser de entrada do exterior ou de saída para o exterior ou de entrada no gabinete do reitor ou de saída do gabinete do reitor.

6.2.2 Modelação do Processo de Negócio

Após entender o funcionamento deste processo de negócio, foi realizada uma modelação para o Processo de Negócio Correspondência utilizando uma ferramenta de modelação (Enterprise Architect), que foi escolhida para modelar os processos de negócio dos casos de estudo deste projeto.

A modelação deste processo de negócio foi realizada com base na informação adquirida acerca do funcionamento dos procedimentos que ocorrem durante o Processo de Negócio Correspondência na Universidade da Madeira.

A implementação deste processo de negócio irá permitir testar o funcionamento da ferramenta "*DSL Generator"* que foi desenvolvida neste projeto, para também conseguir verificar se existe ainda algum erro que precise de possíveis correções assim como também chegar a alguns resultados acerca das limitações da ferramenta.

Depois de analisar bem o funcionamento do processo de negócio "Correspondencia" foi então realizada a sua modelação como é possível verificar na Figura 22, sendo possível verificar na modelação o primeiro nível de abstração do processo de negócio "Correspondencia" da Universidade da Madeira. Para este processo de negócio ficou prevista a existência de duas tarefas de utilizador ("Registar Correspondencia" e "Tratar correspondência").

Também é possível verificar na Figura 22 o ator ou utilizador envolvido neste processo de negócio ("responsavel correspondencia") no qual a sua tarefa será realizar o registo da correspondência.

Utilizando a modelação do processo de negócio elaborada no Enterprise Architect e a ferramenta "DSL Generator" implementada neste projeto, irão ser geradas, para este caso de estudo, 2 interfaces utilizando a modelação do Enterprise Architect representado na Figura 22 (primeiro nível de abstração), sendo cada uma destas interfaces para cada tarefa de utilizador.

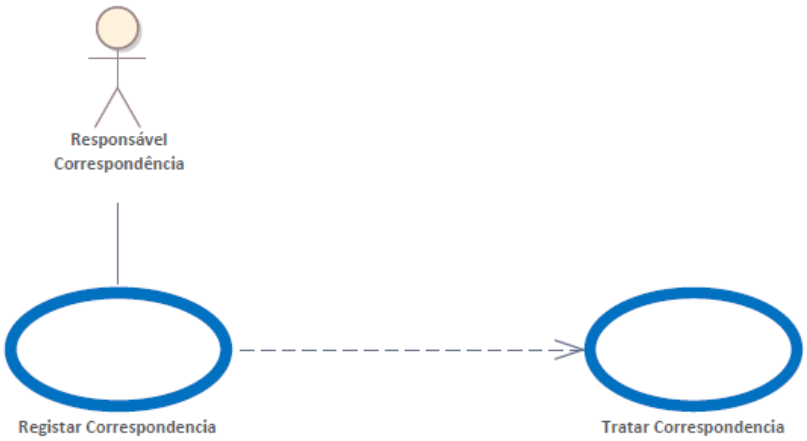

Figura 22. Modelação do primeiro nível de abstração do processo de Negócio "Correspondência"

Na Figura 23 é possível verificar a modelação realizada para o segundo nível de abstração da tarefa de utilizador “Registar Correspondencia” que deverá ser realizada pelos responsáveis da correspondência. Os responsáveis da correspondência poderão ter acesso aos documentos e poderão registar e fazer o encaminhamento das correspondências pretendidas através da plataforma “siga”.

Neste nível de abstração também é possível verificar a existência de intenções de utilizador que contêm “I Go To”, “Search” e “CRUD” no seu nome que são palavras-chave que poderão ser utilizadas na modelação para este nível de abstração, como já foi referido na secção 5.

Cada intenção de utilizador “CRUD” representada na Figura 23 irá ser um formulário que poderá ser visualizado ou preenchido pelo responsável da correspondência (um formulário para o registo da correspondência, um para fazer o encaminhamento e os restantes quatro formulários serão para a visualização dos registos de entrada e de saída de correspondência), dando um total de seis formulários para a tarefa “Registar Correspondencia”. Para a tarefa de utilizador “Tratar Correspondencia” irão existir duas intenções de utilizador CRUD no qual o objetivo será fazer realizar o encaminhamento das correspondências.

Cada intenção de utilizador é constituída por interações de utilizador que serão os campos que poderão ser preenchidos ou visualizados em cada formulário. O nome de cada intenção de utilizador será o nome de uma entidade de dados e as interações de utilizador serão os campos que serão inseridos ou selecionados da mesma entidade de dados. Caso exista um campo que não pertença a mesma entidade de dados foi necessário definir a fonte com a palavra-chave “table” na respetiva interação de utilizador, para dar referência a outra entidade de dados específica que contenha o campo definido.

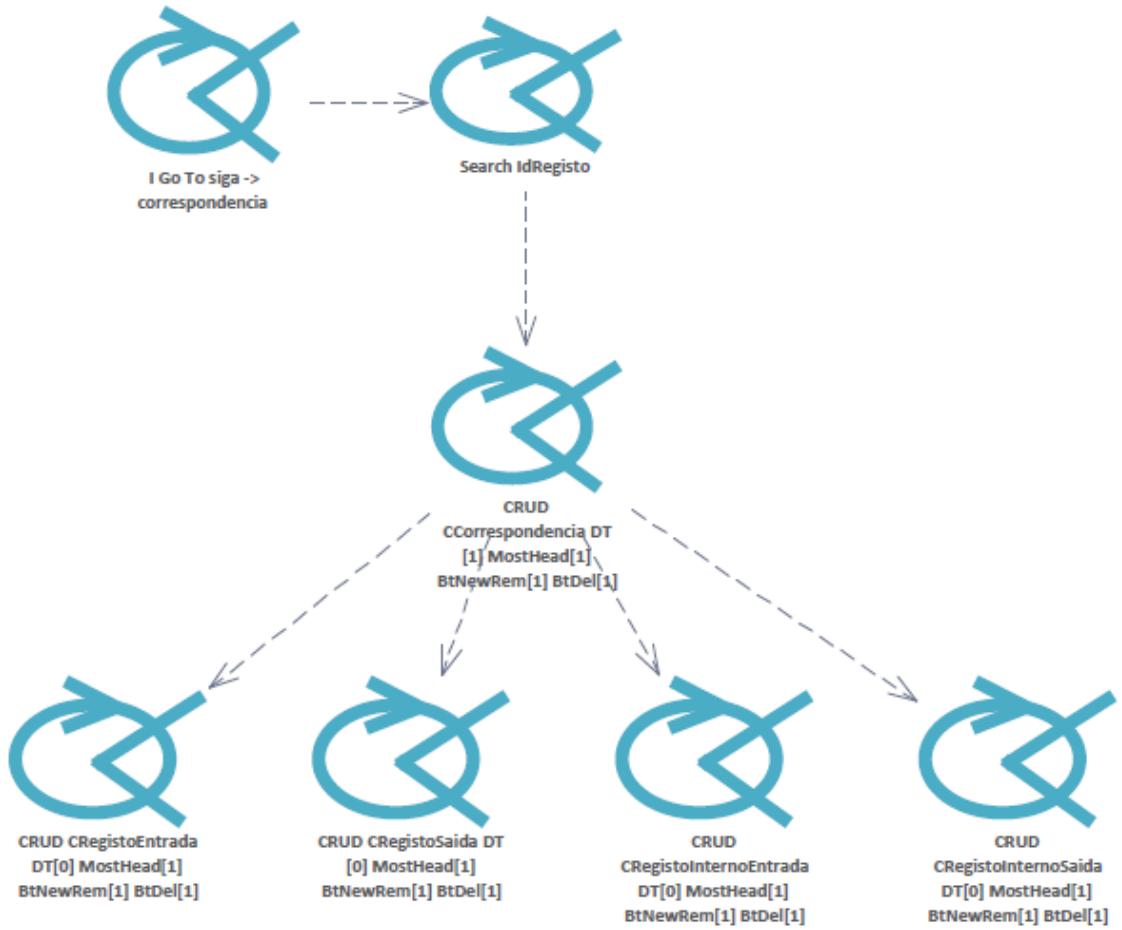

Figura 23. Modelação do segundo nível de abstração da tarefa "Registar Correspondencia"

6.2.3 Análise e Implementação

Para este caso de estudo foi possível fazer a geração de dois ficheiros ("Registar Correspondencia.py" e "Tratar Correspondencia.py"), cada um com a DSL de uma das tarefas de utilizador definidas no primeiro nível de abstração da modelação que é possível verificar na Figura 22 anteriormente referida.

Após a geração deste código é utilizado o "*Hydra Code Generator*" para verificar se o código que foi gerado não contém nenhum erro e não necessita de mais nenhuma alteração.

Para este caso de estudo, os testes são efetuados com base na implementação do código que é gerado pelo "*DSL Generator*", verificando a existência de algum erro, bem como as suas limitações. Também é possível verificar se é possível gerar os formulários pretendidos em cada Espaço de Interação.

Durante a implementação do código foi encontrado um erro no DSL gerado, no qual não era acrescentado "/>" no fim do "FilterEI" do primeiro "Espaço de Agregação", sendo necessária uma pequena atualização no "*DSL Generator*" para a correção do erro.

Durante a primeira geração de código com o "*Hydra Code Generator*" verificaram-se alguns problemas com as relações das entidades de dados sendo necessário alterar as relações para não dar erro, pois as entidades de dados já tinham sido criadas anteriormente. Também foi necessário adaptar os nomes das intenções de utilizador e das interações de utilizador na

modelação para os mesmos nomes que já existiam na base de dados. Esta adaptação foi realizada com o objetivo de não dar erro ou criar campos diferentes, que possivelmente já existiam com o nome diferente e com a mesma finalidade.

Outro erro que foi possível encontrar ao gerar o código com o "*Hydra Code Generator*" foi o facto de este não estar a fazer a comparação dos campos do tipo "text". Para tal, foi necessário atualizar o "*Hydra Code Generator*" para conseguir fazer a comparação entre os campos do tipo "text" e para também conseguir editar os campos, pois cada vez que fosse editado um campo deste tipo não era detetada nenhuma alteração e como tal o campo também não era alterado.

Posteriormente após a geração com "*Hydra Code Generator*" foi necessário alterar ou adaptar a modelação de cada formulário de forma a conseguir fazer com que seja implementado o formulário pretendido em cada Espaço de Interação através da geração de código.

A adaptação realizada nos formulários foi a inserção de descrições nos campos dos formulários utilizando a palavra-chave nas interações de utilizador "DESCRICAO" e o aumento da largura dos campos do tipo "text" utilizando a palavra-chave "width" nas interações de utilizador.

Posteriormente, como nos formulários da tarefa de utilizador "Registar Correspondencia" apenas deveria existir a opção para editar cada correspondência e não seria suposto adicionar nem apagar nenhuma das correspondências foi necessário fazer uma adaptação na modelação para que não fossem criados os botões "novo" e "apagar" que são para adicionar um novo registo e apagar um registo respetivamente. Para tal foi adicionado em cada Intenção de utilizador a palavra-chave "BtNewRem" para não adicionar nos formulários o botão "novo" e a palavra-chave "BtDel" para não adicionar o botão "apagar".

Foi também definido o valor do "Filter_NULL_DevolveTodos" igual a zero em todas as todas as intenções de utilizador da tarefa "Registar Correspondencia" para não estar a devolver todas as correspondências em todos os formulários e otimizar ou acelerar o funcionamento do sistema, para não demorar muito para criar os formulários. Apenas no primeiro formulário, da intenção de utilizador "CRUD Correspondência", a que foi definida a variável anterior ("Filter_NULL_DevolveTodos") igual a 1 para poder visualizar as correspondências sem utilizar apenas o filtro de pesquisa.

Durante este caso de estudo foi necessário fazer uma pequena atualização no "*DSL Generator*" porque apercebido que este não gerava a DSL utilizando a Codificação "UTF-8" e então não adicionava os carateres de forma correta durante a escrita de algumas palavras.

Posteriormente foi ainda necessário atualizar novamente o "DSL Generator" porque foi visto ao gerar a DSL que faltava a palavra-chave "table" nos casos em que as interações de utilizador continham "Insert" no seu nome. Este erro fazia com que não fosse gerado na DSL a "SOURCE" mesmo colocando "table" na modelação.

Posteriormente foi necessário colocar a opção para mostrar o tipo de correspondência (nome em vez do "id") e também para mostrar o registo das correspondências no formulário.

Posteriormente foi decidido que o encaminhamento saísse da modelação, porque não foi possível guardar de forma correta o registo na entidade de dados "CEncaminhamento", sendo uma entidade de dados em que ocorre erros ao gerar o código com o "*Hydra Code Generator*" utilizando a DSL Gerada. Os erros que ocorrem nesta entidade de dados deve-se ao facto de a chave primária estar definida com um nome num formato errado (ter como nome "IdEncaminhamento" em vez de "IdCEncaminhamento").

Na Figura 24 é possível verificar a interface implementada para a tarefa de utilizador “Registar Correspondência” com o código gerado através da modelação do processo de negócio. Nesta interface é possível filtrar as correspondências através do número do registo colocado no canto superior direito a azul e verificar se a correspondência contém um registo de entrada externo ou interno ou se contém um registo de saída externo ou interno. Na Figura 24 é possível verificar o exemplo de um filtro de uma correspondência com registo interno de entrada.

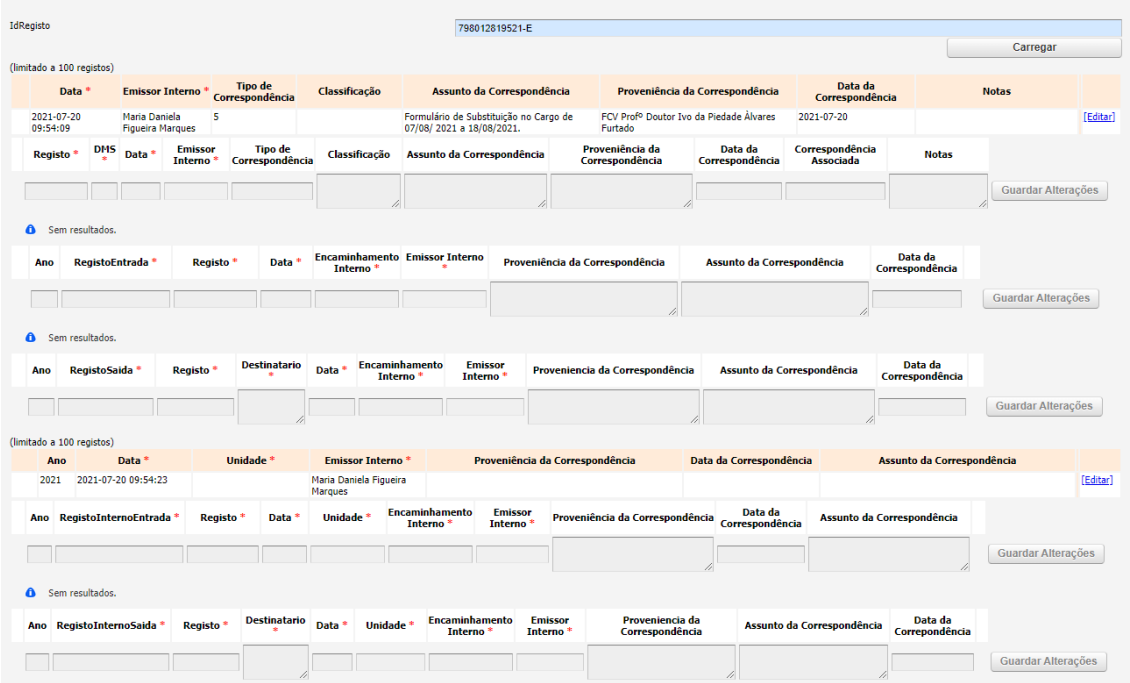

Figura 24. Interface implementada para a tarefa de utilizador Registar Correspondencia

6.2.4 Conclusões

Neste caso de estudo foi utilizado o “*DSL Generator*” para gerar código para a Hydra e foi possível implementar, com o código gerado, formulários em que era possível editar a sua informação. Nestes formulários era possível visualizar as correspondências e editar os seus dados. Desta forma foi decidido desenvolver uma interface por parte dos atuais utilizadores da aplicação, de forma que estes possam corrigir os erros que cometam ou detetem. Para tal foi possível ainda adaptar, através da modelação, os formulários e os seus campos.

Durante este caso de estudo ocorreram algumas limitações em relação aos nomes dos “id’s” das entidades de dados devido ao facto de serem utilizadas entidades de dados já existentes, e para não acrescentar mais nenhuma entidade de dados com o mesmo propósito, foram reutilizados os mesmos nomes já inseridos na base de dados.

Nos Caso de Estudo, do processo de Negócio Candidaturas e do processo de Negócio Correspondência, ocorreram algumas limitações porque as entidades de dados já tinham sido criadas sendo necessário adaptar a modelação com os mesmos nomes das entidades de dados. Porém para o Caso de Estudo do Processo de Negócio Teses já não existiam entidades de dados criadas e as limitações existentes nos outros casos de estudo já não ocorreram ou foram bastante reduzidas.

Em Conclusão, foi possível gerar a DSL totalmente correta e sem erros. Tendo em conta que não ocorreram erros ao nível da geração de código é possível concluir que o código que foi gerado

não continha erros, sendo apenas corrigidos alguns erros existentes que ainda não teriam sido avistados no gerador de código, através de pequenas atualizações.

6.3 Processo de Teses

O caso de estudo "Processo de Teses" tem como objetivo contribuir para a melhoria do processo de submissão e acompanhamento dos trabalhos científicos para os cursos da Universidade da Madeira.

Este caso de estudo foi proposto para automatizar o processo de teses e contribuir para o seu melhor funcionamento através da utilização das plataformas "Sidoc" (Plataforma dos docentes na Universidade da Madeira) e "InfoAlunos" (plataforma dos alunos da Universidade da Madeira) implementadas com *framework* Hydra. Para a realização da implementação deste processo de negócio foi primeiro elaborada a sua modelação baseada no funcionamento do processo, sendo também necessário entender o seu funcionamento e os requisitos necessários. Depois da realização da modelação o objetivo foi testar e verificar o funcionamento da ferramenta "*DSL Generator"* criada neste projeto e detalhada na secção 5.

Este caso de estudo teve como objetivo modelar um processo de negócio no qual o aluno conseguisse através de uma plataforma, poder visualizar os trabalhos científicos propostos pelos docentes, aceitar um dos trabalhos propostos e depois conseguir submeter o trabalho científico. Desta forma seria possível visualizar o estado do trabalho científico de cada aluno, ou seja, visualizar se já escolheu o trabalho, se já submeteu, se já apresentou ou se já recebeu a nota final.

Neste caso de estudo poderão existir algumas diferenças daquilo que acontece na realidade, pois este caso de estudo foi realizado com o objetivo de verificar e testar o funcionamento da ferramenta desenvolvida, bem como conseguir dar resposta a questão de investigação já referida anteriormente. Como tal não deverão ser tidos em conta alguns aspetos que poderão aparecer na modelação do processo de negócio pois poderá não corresponder ao que acontece durante o processo de negócio real da Universidade da Madeira.

6.3.1 Descrição do Processo de Negócio

O processo de Negócio Teses da universidade da Madeira está centralizado no acompanhamento e submissão das teses dos alunos da universidade da Madeira. Para tal são propostos os trabalhos científicos com a indicação dos orientadores e o aluno pode aceitar um dos trabalhos propostos e por fim, depois da submissão do trabalho efetuado é realizada uma defesa e o trabalho é apresentado.

Para tal seria bom a existência de uma plataforma que fosse capaz de verificar o estado de cada trabalho científico proposto, e verificar se a data da defesa dos trabalhos científicos, de forma a contribuir para a melhoria deste processo e tornar mais fácil a visualização do estado de cada trabalho científico proposto e aceite.

Durante o Processo de Negócio Teses, primeiro o professor deverá propor o trabalho científico, no qual deverá indicar o título, e uma descrição do trabalho através de um documento. Também deverá ainda indicar os possíveis orientadores do trabalho. Os orientadores do trabalho serão docentes e deve ser possível ainda indicar a Data de Início e do Fim da orientação.

Posteriormente o aluno deverá poder aceitar um dos trabalhos propostos e depois o diretor de curso deverá indicar se o trabalho é aprovado para aquele aluno. Depois do trabalho ser aprovado pelo diretor de curso, o aluno deverá ainda poder submeter o trabalho científico.

Depois também será marcada a defesa pela unidade de assuntos académicos, indicando a data, o nome do trabalho científico e o Júri da defesa que irá estar presente. O Júri da defesa irá ser constituído por docentes, sendo também possível indicar em cada um, a sua função no Júri.

Posteriormente a Reitoria aprova a defesa do trabalho científico e depois da realização desta defesa o diretor de curso deverá poder indicar a nota final do trabalho. Para que o Diretor de Curso consiga indicar a nota final do trabalho será necessário a inscrição na matrícula por parte do aluno.

6.3.2 Modelação do Processo de Negócio

Após entender melhor o funcionamento deste processo de negócio e da realização de um modelo de classes, foi realizada uma modelação para o Processo de Negócio Teses utilizando a ferramenta de modelação escolhida para fazer as modelações deste projeto (Enterprise Architect). A modelação deste processo de negócio foi realizada com base na informação adquirida acerca do funcionamento dos procedimentos que ocorrem durante o Processo de Negócio Teses na Universidade da Madeira.

A implementação deste processo de negócio irá permitir testar o funcionamento da ferramenta "*DSL Generator"* que foi desenvolvida neste projeto, para também conseguir verificar se existe ainda algum erro que precise de possíveis correções assim como também chegar a alguns resultados acerca das limitações da ferramenta.

Antes da realização da modelação do Processo de Negócio foi realizado primeiro uma modelação das classes que possivelmente estariam envolvidas durante o processo. O objetivo seria conseguir ter todos os conceitos bem definidos que poderiam ser necessário no processo de negócio de forma a tornar também mais fácil de realizar a sua modelação com os atores ou utilizadores e as suas tarefas de utilizador.

Depois de analisar bem o funcionamento do processo de negócio "Teses" foi então realizada a sua modelação no Enterprise Architect como é possível verificar na Figura 25, sendo possível verificar na modelação o primeiro nível de abstração do processo de negócio "Teses" da Universidade da Madeira.

Irão ser geradas, para este caso de estudo, sete interfaces utilizando a modelação representada na Figura 25 (primeiro nível de abstração), sendo cada uma destas interfaces para cada tarefa de utilizador (cada uma com a sua função)

Como é possível verificar na Figura 25, para este processo de negócio ficou prevista a existência de 7 tarefas de utilizador ("Propor Trabalho Científico", "Aceitar Trabalho Científico", "Aprovar Trabalho Científico", "Submeter Trabalho Científico", "Marcar Defesa Trabalho Científico" "Aprovar Defesa Trabalho Científico" e "Avaliar Trabalho Científico"). Também é possível verificar na Figura 25 a possível existência de cinco atores ou utilizadores (Professor, Aluno, Diretor de Curso, Assuntos Académicos e Reitoria) que poderão interagir neste processo de Negócio.

O docente deverá ter como tarefa de utilizador "Propor Trabalho Científico", no qual irá ter um formulário para inserir um trabalho científico com um documento de descrição do tema e para inserir os orientadores.

O aluno deverá ter como tarefas de utilizador: "Aceitar Trabalho Cientifico" e "Submeter Trabalho Cientifico", para aceitar que poderá realizar o trabalho e para submeter a Tese do trabalho realizado respetivamente.

.

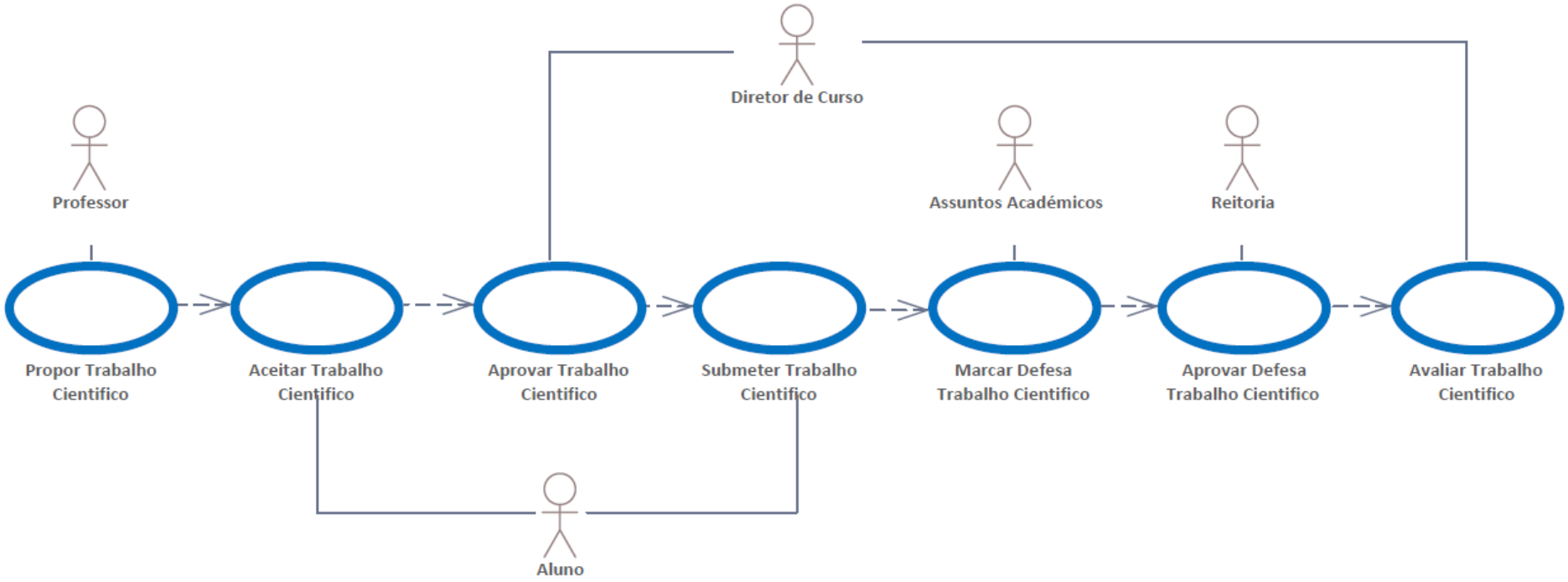

Figura 25. Modelação do Processo de Negócio Teses

O Diretor de Curso terá como tarefas de utilizador “Aprovar Trabalho Cientifico” e “Avaliar Trabalho Cientifico”, para indicar que o aluno ficará com o trabalho cientifico aceite e para lançar a nota do trabalho científico após a defesa, respetivamente.

A Unidade de Assuntos Académicos (UAA) deverá visualizar as Defesas dos trabalho científicos e também poderá criar uma nova Defesa inserindo o nome da trabalho científico, a data da defesa e a constituição do Júri.

Depois a reitoria terá como tarefa de utilizador “Aprovar Defesa Trabalho Cientifico”. Após esta aprovação e depois do aluno realizar a defesa, o Diretor de Curso deverá poder lançar a nota final do respetivo trabalho científico (tarefa de utilizador “Avaliar Trabalho Cientifico”) tendo em conta a opinião do Júri da defesa do trabalho científico.

Posteriormente foi realizada a modelação do segundo nível de abstração para cada tarefa de utilizador. Na Figura 26 é possível verificar a modelação realizada para o segundo nível de abstração da tarefa de utilizador “Propor Trabalho Cientifico” que deverá ser realizada pelos Docentes. Os Docentes poderão ter acesso aos trabalhos científicos propostos e poderão adicionar uma nova proposta para um trabalho científico através da plataforma “Sidoc”.

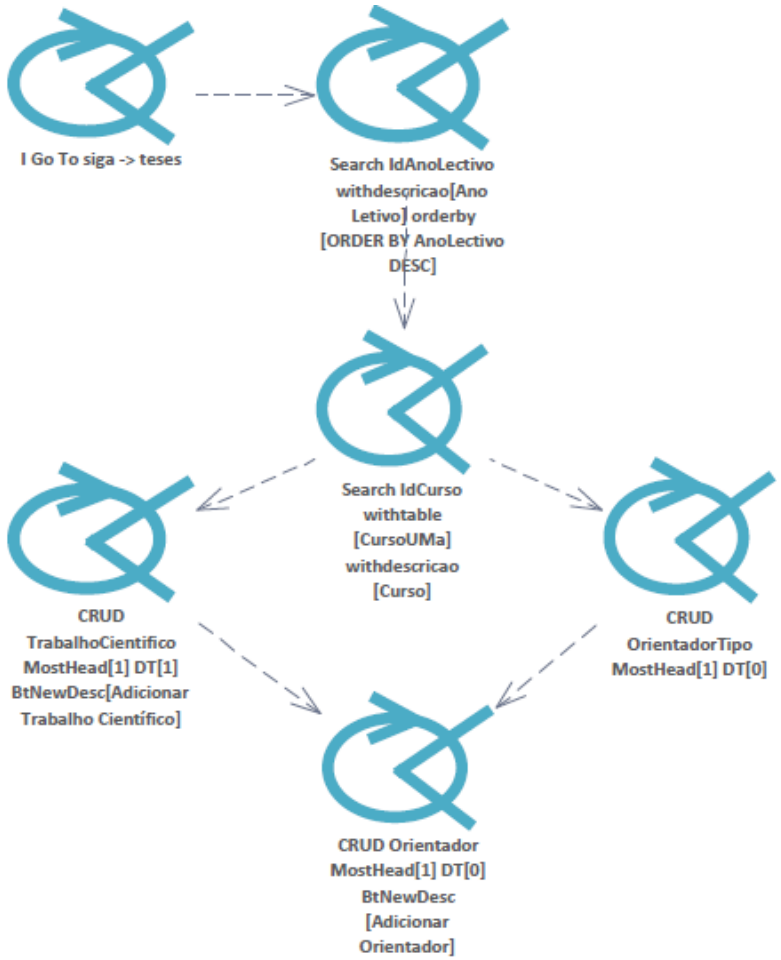

Figura 26. Modelação do segundo nível de abstração para a tarefa Propor Trabalho Científico

Neste nível de abstração também é possível verificar a existência de intenções de utilizador que contêm “I Go To”, “Search” e “CRUD” no seu nome que são palavras-chave que poderão ser utilizadas na modelação para este nível de abstração, como já foi referido na secção 5.

Cada intenção de utilizador “CRUD” representada na Figura 25 irá ser um formulário que poderá ser visualizado ou preenchido por um Docente, dando um total de três formulários para a tarefa “Propor Trabalho Cientifico”.

6.3.3 Análise e Implementação

Para este caso de estudo foi possível fazer a geração de sete ficheiros (“Propor Trabalho Cientifico.py”, “Aceitar Trabalho Cientifico.py”, “Aprovar Trabalho Cientifico.py”, “Submeter Trabalho Cientifico.py”, “Marcar Defesa Trabalho Cientifico.py”, “Aprovar Defesa Trabalho Cientifico.py” e “Avaliar Trabalho Cientifico.py”), cada um com a DSL de uma das tarefas de utilizador definidas no primeiro nível de abstração, da modelação, que é possível verificar na Figura 24 anteriormente referida.

Após a geração deste código é utilizado o “*Hydra Code Generator*” para verificar se o código que foi gerado não contém nenhum erro e não necessita de mais nenhuma alteração.

Para este caso de estudo, os testes são efetuados com base na implementação do código que é gerado pelo “*DSL Generator*”, verificando a existência de algum erro, bem como as suas limitações. Também é possível verificar se é possível gerar os formulários pretendidos em cada Espaço de Interação.

Durante a implementação deste caso de estudo, analogamente ao Processo de Negócio Correspondência, foi necessário adaptar a modelação de forma a conseguir obter os formulários pretendidos para o Processo de Negócio Teses.

Com este caso de estudo foi possível detetar um erro que existia no “*DSL Generator*” quando era acrescentado um campo em que seria necessário acrescentar um ficheiro.

Futuramente pode ser necessário acrescentar uma tarefa de utilizador “Gerir Teses Tabelas Tipo” que contivesse uma intenção de utilizador “CRUD TipoOriendador”. Isto para poder inserir um “TipoOrientador” numa interface à parte, diferente daquela que eram adicionados os trabalhos científicos e os orientadores na tarefa de utilizador “Propor Trabalho Cientifico”. Esta decisão pode ser tomada porque não faria sentido adicionar um trabalho científico e ter de adicionar ainda tipos de orientador na mesma interface.

Para as tarefas de utilizador “Propor Trabalho Cientifico” e “Aceitar Trabalho Cientifico” foi possível implementar os formulários pretendidos com a geração de código através da modelação, podendo adaptar como já foi referido o visual dos formulários. Foi necessário saber o nome a entidade de dados já existente para os cursos da Universidade da Madeira e o nome da entidade de dados dos docentes já existente para que fosse possível filtrar os docentes da Universidade da Madeira.

Durante este caso de estudo foi possível observar alguns erros que ocorreram ao adaptar a modelação devido ao engano ao referir o nome correto de alguma palavra-chave, porém foram erros que ocorreram na modelação e que foram fáceis de detetar.

Utilizando as palavras-chave inseridas na modelação foi possível obter formulários com a informação pretendida e da forma pretendida, podendo existir ainda algumas limitações devido

a inexistência de algumas palavras-chave, mas que podem ser implementadas no futuro como atualização do "*DSLGenerator*".

Uma limitação que também existe no "*DSLGenerator*" é a definição das permissões que cada utilizador tem em relação as entidades de dados que tem de ser feita manual, pois não é possível definir permissões através da modelação. Uma solução que pode ser realizada neste momento para contornar esta limitação é apenas a possibilidade de retirar os botões para poder adicionar elementos na entidade de dados ou eliminar elementos existentes na entidade de dados.

Na Figura 27 é possível verificar o resultado de uma das interfaces implementadas para este caso de estudo utilizando o código gerado pelo "DSLGenerator" através da modelação.

Na Figura 27 é possível identificar, no canto superior direito a azul, que os trabalhos científicos podem ser filtrados nesta interface por Ano Letivo e por Curso. Estes dois filtros foram adicionados porque foi adicionado no segundo nível de abstração duas intenções de utilizador com a palavra-chave "Search" no nome, como é possível verificar na Figura 26.

Figura 27. Interface implementada para a tarefa de utilizador Propor Trabalho Cientifico

Figura 28. Interface implementada para a tarefa de utilizador Aceitar Trabalho Cientifico

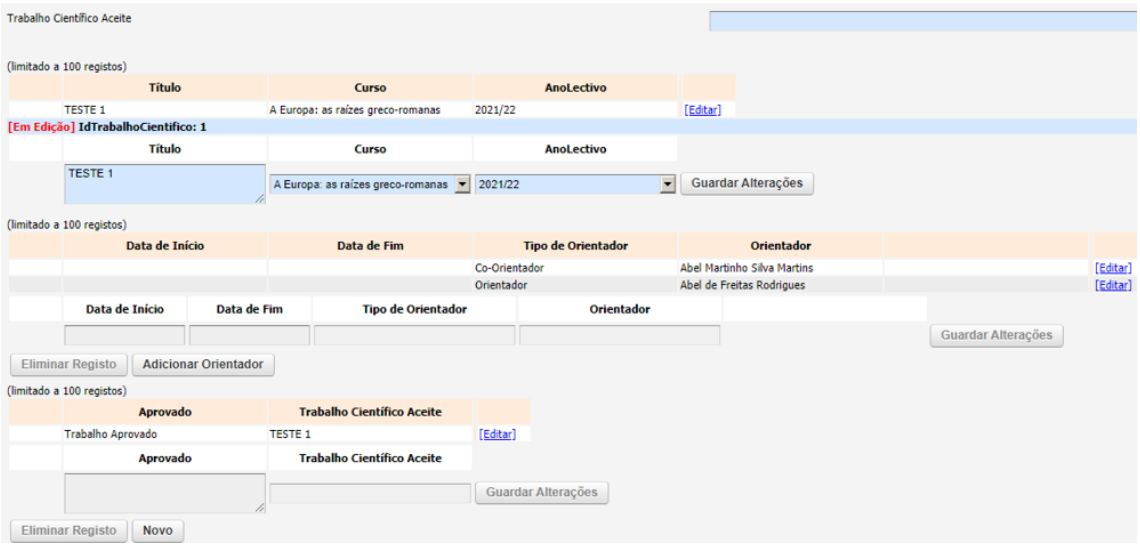

Figura 29. Interface implementada para a tarefa de utilizador Aprovar Trabalho Cientifico

Na Figura 27 é possível verificar que foi possível filtrar um trabalho científico e é possível encontrar os orientadores para aquele trabalho científico. Isto foi possível implementar porque no segundo nível de abstração, representado na Figura 26, foi referido a existência de uma intenção de utilizador "CRUD" para Orientador e outra para Trabalho Científico como intenção de utilizador anterior. Foi adicionada ainda figura 26 um CRUD para o tipo de orientador para que fosse possível adicionar tipos de orientador na interface representada na Figura 27.

Analogamente a Figura 27, na Figura 28 e 29 é possível verificar o resultado de outras interfaces implementadas para este caso de estudo.

6.3.4 Conclusões

Neste caso de estudo é possível concluir que foi gerado código automaticamente de forma correta utilizando a ferramenta "*DSLGenerator*", sendo alguns erros ocorridos por parte da modelação e não em relação a ferramenta.

É possível concluir que uma das limitações é que pode ser necessário inserir uma tarefa de utilizador à parte que esteja relacionada com algum "tipo". Neste caso de estudo pode ser necessário inserir uma tarefa de utilizador com o nome "Gerir Teses Tabelas Tipo" que inclui duas tarefas de utilizador para permitir inserir na base de dados os tipos de Júri e os tipos de Orientador.

Em Conclusão, o "*DSLGenerator*" é capaz de gerar a DSL de forma correta e sem erros podendo apenas existir algumas limitações devido a falta de palavras-chave que podem ser inseridas numa implementação futura. Apesar de o "*DSLGenerator*" ter de sofrer atualizações, este já se encontra numa versão em que já possível gerar um DSL com muito conteúdo apenas com a modelação do Processo de Negócio.

6.4 Conclusão

Com os casos de estudo realizados e para dar resposta a questão de investigação introduzida neste projeto, foi possível concluir que, com a versão atual do "*DSLGenerator*", é possível gerar

a DSL pretendida sem erros e sem a necessidade de correção por parte do desenvolvedor. É ainda possível concluir que a maior parte dos erros ocorridos foram enganos realizados na modelação e que já não estão relacionados com o funcionamento da ferramenta.

Porém a ferramenta neste momento pode ter algumas limitações ao nível da inserção de alguns "ParametrosNegocio" e na inserção dos "FieldInfo" devido a possível falta de algumas palavras-chave que podem ser inseridas na modelação e também devido ao facto de alguns "ParametrosNegocio" não fazerem sentido ser induzidos na modelação. Estas limitações referidas, no entanto, podem ficar como possíveis atualizações futuras ao "*DSLGenerator*".

Outra limitação verificada é que, durante a modelação, se forem adicionadas interações de utilizador por engano numa das intenções de utilizador o campo é criado na base de dados e depois do campo ser adicionado já não é possível retirar o campo apenas com a modelação, sendo necessário eliminar manualmente na base de dados.

Outra limitação que foi possível verificar utilizando o Enterprise Architect neste projeto é que este só permite definir objetos para os diagramas com um máximo de 255 carateres. Esta é uma limitação que pode ser necessária ter em conta durante a definição dos nomes das tarefas de utilizador, intenções de utilizador e interações de utilizador na modelação utilizado o Enterprise Architect. Porém esta última limitação referida já é do Enterprise Architect e não do "*DSLGenerator*", não influenciando o funcionamento da ferramenta ("*DSLGenerator*").

Através dos casos de estudo foi possível observar uma rapidez na implementação no que diz respeito a construção dos formulários pois não haveria necessidade de se preocupar com a estrutura do código DSL sendo algo que foi automatizado através da ferramenta "*DSLGenerator*".

Uma atualização futura que poderia existir no "*DSLGenerator*" seria a implementação de módulos de testes permitir verificar mais facilmente e rapidamente a presença de erros na ferramenta e a implementação ou alteração de mais palavras-chave de forma a reduzir as limitações da ferramenta.

7. Conclusão

Nesta secção é apresentado um pequeno resumo do problema proposto para este trabalho e da solução proposta e são ainda apresentadas as conclusões retiradas da experiência tida nos casos de estudo realizados. Por fim, é também realizada uma reflexão sobre os elementos desenvolvidos e a sua adequação aos objetivos deste projeto de mestrado.

Neste trabalho optou-se, primeiro, por fazer uma pesquisa, com o objetivo de verificar a existência de ferramentas capazes de gerar código, se possível com a geração da estrutura completa do sistema, com base na modelação de tarefas, preferencialmente de processos de negócio.

Foram encontrados um total 80 artigos, utilizado o método de investigação, no qual 19 foram repetidos utilizando as frases de pesquisa, 1 já tinha sido encontrado antes de efetuar a pesquisa, 38 não foram analisados (excluídos), 22 artigos foram incluídos e 4 destes 22 não foram analisados porque já teriam sido encontrados artigos mais recentes dos mesmos autores ou do mesmo tema.

Posteriormente, foram selecionados 5 artigos que tinham uma ferramenta de geração de código desenvolvida. O objetivo desta seleção foi verificar que tipo de estrutura de código a que estes geravam, sendo necessário verificar se geravam uma interface, uma lógica de negócio ou se geravam dados. Destes 5 artigos 3 fizeram gerar uma lógica de negócio e 2 fizeram gerar a estrutura completa (interface de utilizador, lógica de negócio e base de dados).

Destes 2 artigos que fizeram gerar a estrutura completa, um deles gera código a partir de especificações XML da interface de utilizador e não da modelação em UML e outro faz a descrição de uma ferramenta que consegue gerar o PIM, o PSM e código a partir do CIM que é um modelo muito abstrato. Como o CIM é um modelo muito abstrato e ferramenta ainda está em fase experimental poderão existir muitas alterações por parte do desenvolvedor.

A ferramenta implementada neste projeto difere daquelas que foram referidas nos artigos e que fizeram gerar a estrutura completa, porque o código é gerado através do PIM que é um modelo que descreve o funcionamento e os detalhes do sistema de forma a conseguir obter a implementação da interface pretendida, não sendo este um modelo assim tão abstrato como o CIM que é utilizado num dos artigos referidos anteriormente e que geram a estrutura completa.

Com a pesquisa efetuada, foi ainda possível verificar, que a geração automática de código é um tema que está a ser melhorado ao longo dos anos e que ainda é um tema atual, com a geração de cada vez mais linguagens e com cada vez mais métodos gerados.

O desafio abordado neste trabalho é a necessidade, comum, em introduzir alterações automaticamente nas aplicações já desenvolvidas ou em desenvolvimento através da modelação do processo de negócio, devido a, ou requisitos mal definidos ou estes sofrerem constantes atualizações. O problema das várias mudanças no funcionamento do sistema e da modelação é que por vezes a adaptação do sistema já implementado, ou a implementar, não é simples, implicando um custo elevado para o seu desenvolvimento.

Neste trabalho foi proposta a implementação de uma solução, de forma a acelerar o processo de desenvolvimento e tentar reduzir ou eliminar o tempo de implementação de software. Desta forma, esta solução tem o objetivo de contribuir para que as alterações dos requisitos do sistema ou dos regulamentos não tenham um grande impacto em relação ao tempo de implementação de software. Consequentemente a solução proposta também poderá resolver

problemas relacionados com a resolução de erros no código que muito provavelmente podem ocorrer durante a implementação de software.

A solução proposta neste trabalho foi dividida em duas contribuições. A primeira, mais simples, foi a escolha da ferramenta de modelação Enterprise Architect (EA) para a realização da modelação dos processos de negócios. Era fundamental ser uma ferramenta capaz de utilizar e extender o UML e capaz de exportar a modelação realizada para um ficheiro.

A segunda contribuição, mais importante e central para o processo de desenvolvimento, foi a implementação de livrarias de tradução entre a modelação e os formatos descritivos usado pelas livrarias da *framework* Hydra, designadas de "*DSLGenerator*". Esta é uma ferramenta que permite acelerar a fase da implementação de software, bem como otimizar o código a ser gerado. Esta proposta permite partir de uma modelação gráfica do processo de negócio, mais simples de poder ser entendida tanto pelos programadores como pelos clientes da aplicação, e obter os necessários ficheiros para a *framework* Hydra, constituídos pela interface de utilizador, lógica de negócio e base de dados.

A exploração da ferramenta implementada foi feita recorrendo a vários casos de estudo e, além de permitirem a validação do desenvolvimento realizado, contribuíram para dar resposta à questão de investigação definida para este projeto. Tendo por base os resultados dos casos de estudo podemos afirmar que é possível, com a versão atual do "*DSLGenerator*", gerar a DSL para a Hydra, sem erros e sem a necessidade de intervenção desenvolvedor. Os casos de estudos variam entre processos de alguma simplicidade, como o Processo Correspondência, até o Processo de Candidatura, que tem já um número de tarefas e grau de complexidade bastante maior, aumentando assim o grau de confiança nos resultados da ferramenta desenvolvida.

De referir ainda, o processo de candidaturas teve uma utilização durante algumas semanas em ambiente real, sendo criadas mais de 400 candidaturas e não surgindo qualquer problema com o código criado. Apesar de o funcionamento, que ocorre após os testes do código já integrados na *framework* Hydra, confirma e reforça a correta geração realizada pela ferramenta proposta neste trabalho.

A utilização da solução implementada faz tornar o processo de desenvolvimento de software, utilizado na Universidade da Madeira, um processo mais rápido e eficiente, pois para implementar formulários é importante apenas saber fazer a modelação do processo de negócio que depois a ferramenta trata da estrutura do código e com que valor a que devem ser definidas as variáveis na DSL gerada. Minimiza ainda os erros de uma ou múltiplas alterações que surjam, sendo uma forma de otimização do desenvolvimento e manutenção do processo de negócio, gerando código uniforme e que, a surgir algum erro, sendo corrigido poderá ser melhorada toda a base de código anteriormente gerada.

A ferramenta desenvolvida tem ainda algumas limitações, especificamente no que diz respeito à inserção dos parâmetros de negócio e gestão dos campos na entidade de dados. Na inserção de parâmetros de negócio, poderão ainda existem palavras-chave em falta, mas pode ser resolvida com uma atualização simples da ferramenta. Na gestão dos campos, poderão existir entidades de dados que já podem ter sido criadas na Hydra, mas não existirem na modelação. Assim, é necessário colocar o mesmo nome das entidades de dados na modelação para que não sejam acrescentadas entidades de dados e campos com o mesmo propósito. Esta limitação também ocorre devido ao facto de não ser ainda possível alterar nem eliminar a informação dos campos de uma entidade de dados através da ferramenta desenvolvida. Nesta versão, e através

da ferramenta implementada, apenas é possível adicionar um novo campo de uma entidade de dados ou uma nova entidade de dados.

O trabalho desenvolvido neste projeto permitiu verificar que é possível implementar uma ferramenta capaz de gerar código de forma correta apenas com a modelação do processo de negócio, bem como verificar benefícios da geração de código. Os maiores benefícios identificados na geração de código é a aceleração do tempo de implementação de software e a facilidade em estabelecer a relação direta entre a modelação e a implementação, permitindo fazer as adaptações no sistema de forma rápida e eficaz.

A ferramenta implementada teve como contexto a *framework* Hydra e, apesar de não ser possível utilizar esta mesma solução para outro tipo de *framework*, pois o código gerado fica numa estrutura específica da Hydra, o conceito proposto pode ser estendido, com alguma facilidade, para outros contextos. A utilização de uma ferramenta como o EA, genérica e muito divulgada, permite melhorar a possibilidade de, usando outra *framework* e adaptando as livrarias de geração de código, rapidamente a base de processos de negócios já modelados se possa reimplementar num novo sistema.

7.1 Trabalho Futuro

Um primeiro trabalho a desenvolver, poderia ser a melhoria da ferramenta de forma a reduzir as limitações identificadas, como ao nível das palavras-chave implementadas e como ao nível da geração de código. Também poderá ser necessário implementar algo que faça reduzir as limitações na ferramenta em relação a utilização de entidades de dados já existentes.

Outra implementação importante para melhorar o processo de desenvolvimento proposto passaria pela extensão do gerador de código com módulos de teste automatizados. Permitiria adicionar um primeiro nível de validação sobre as atualizações e confirmar de forma mais fácil para o desenvolvedor a inexistência (ou não) de erros.

Referências

[1] A. Bucchiarone, J. Cabot, R. F. Paige, e A. Pierantonio, «Grand challenges in model-driven engineering: an analysis of the state of the research», *Softw Syst Model*, vol. 19, n. 1, pp. 5–13, Jan. 2020, doi: 10.1007/s10270-019-00773-6.

[2] M. A. Babar, A. W. Brown, e I. Mistrik, *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Newnes, 2013.

[3] D. N. F. H. Costa, «Geração automática de interfaces com o utilizador: uma abordagem baseada em MDA para a plataforma PHP», masterThesis, Universidade da Madeira, 2007. Acedido: Dez. 21, 2020. [Em linha]. Disponível em: https://digituma.uma.pt/handle/10400.13/14

[4] F. Brooks, «No Silver Bullet – Essence and Accident in Software Engineering», p. 16, 1986.

[5] I. Ozkaya, «Are DevOps and Automation Our Next Silver Bullet?», *IEEE Software*, vol. 36, n. 4, pp. 3–95, Jul. 2019, doi: 10.1109/MS.2019.2910943.

[6] P. D. Valente, «The GOALS approach: business and software modeling traceability by means of human-computer interaction: enterprise modeling language and method», Dez. 2017, Acedido: Dez. 14, 2020. [Em linha]. Disponível em: https://digituma.uma.pt/handle/10400.13/1755

[7] N. Nunes, «WISDOM: Whitewater Interactive System Development with Object Models», Jan. 2001.

[8] Y. Singh e M. Sood, «The Impact of the Computational Independent Model for Enterprise Information System Development», *International Journal of Computer Applications*, vol. 11, Dez. 2010, doi: 10.5120/1602-2153.

[9] M. I. Mukhtar e B. S. Galadanci, «AUTOMATIC CODE GENERATION FROM UML DIAGRAMS: THE STATE-OF-THE-ART», *The State*, vol. 13, n. 4, p. 14, 2018.

[10] M. Mazanec e O. Macek, «On general-purpose textual modeling languages», *In Proceedings of DATESO*, vol. 12, pp. 1–12, Jan. 2012.

[11] N. Oliveira, M. J. V. Pereira, e P. R. Henriques, «Domain-Specific Languages: A Theoretical Survey», p. 12.

[12] «MDA Tool.pdf». Acedido: Nov. 22, 2020. [Em linha]. Disponível em: https://sparxsystems.com/bin/MDA%20Tool.pdf

[13] W. Ecker, K. Devarajegowda, M. Werner, Z. Han, e L. Servadei, «Embedded Systems' Automation following OMG's Model Driven Architecture Vision», p. 6, 2019.

[14] M. EL Omari, M. Erramdani, e A. Rhouati, «A Model Driven Approach for Generating Angular 7 Applications», *Int. J. Recent Contrib. Eng. Sci. IT*, vol. 8, n. 2, p. 36, Jun. 2020, doi: 10.3991/ijes.v8i2.14131.

[15] «CIM vs PIM vs PSM | Business Rule Solutions». https://www.brsolutions.com/tag/cim-vs-pim-vs-psm/ (acedido Dez. 14, 2020).

[16] G. Booch, I. Jacobson, e J. Rumbaugh, «The Unified Modeling Language for Object-Oriented Development», p. 35, 1996.

[17] R. Breu *et al.*, «Towards a formalization of the Unified Modeling Language», em *ECOOP'97 — Object-Oriented Programming*, Berlin, Heidelberg, 1997, pp. 344–366. doi: 10.1007/BFb0053386.

[18] H. Störrle, «On the impact of layout quality to understanding UML diagrams: Diagram type and expertise», em *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Set. 2012, pp. 49–56. doi: 10.1109/VLHCC.2012.6344480.

[19] B. Bauer e J. Odell, «UML 2.0 and agents: how to build agent-based systems with the new UML standard», *Engineering Applications of Artificial Intelligence*, vol. 18, n. 2, pp. 141–157, Mar. 2005, doi: 10.1016/j.engappai.2004.11.016.

[20] «Class Diagrams – Part 1», *fdiez*, Jul. 10, 2017. http://fdiez.org/class-diagrams-part-1/ (acedido Nov. 06, 2020).

[21] I. Gorton, «Understanding Software Architecture», Mar. 2011, doi: 10.1007/978-3-642-19176-3_1.

[22] R. M. Bastos e D. D. A. Ruiz, «Extending UML activity diagram for workflow modeling in production systems», em *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, Jan. 2002, pp. 3786–3795. doi: 10.1109/HICSS.2002.994510.

[23] T. Kamariddinov e R. Bechara, «Software Projects: Gamification of Healthy Eating», 2018. doi: 10.13140/RG.2.2.32491.49446/1.

[24] E. Adetiba e M. Serem, «ConfBits: A Web Based Conference Management System», *International Journal of Engineering Science Invention*, vol. 2, pp. 92–100, Jul. 2013.

[25] D. Costa, L. Nóbrega, e N. J. Nunes, «An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes», em *Task Models and Diagrams for Users Interface Design*, vol. 4385, K. Coninx, K. Luyten, e K. A. Schneider, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 137–152. doi: 10.1007/978-3-540-70816-2_11.

[26] G. Paolone, M. Marinelli, R. Paesani, e P. Di Felice, «Automatic Code Generation of MVC Web Applications», *Computers*, vol. 9, n. 3, p. 56, Jul. 2020, doi: 10.3390/computers9030056.

[27] H. Benouda, M. Azizi, M. Moussaoui, e R. Esbai, «Automatic code generation within MDA approach for cross-platform mobiles apps», em *2017 First International Conference on Embedded & Distributed Systems (EDiS)*, Oran, Dez. 2017, pp. 1–5. doi: 10.1109/EDIS.2017.8284045.

[28] S. E. Ev e P. Samuel, «Automatic Code Generation Using UML Activity and Sequence Models», *IET Software*, vol. 10, Jul. 2016, doi: 10.1049/iet-sen.2015.0138.

[29] K. Mizuoka e M. Koga, «MDA development of Manufacturing Execution System based on automatic code generation», em *Proceedings of SICE Annual Conference 2010*, Ago. 2010, pp. 3103–3106.

[30] F. Nascimento, M. Oliveira, M. Wehrmeister, C. Pereira, e F. Wagner, «MDA -based approach for embedded software generation from a UML/MOF repository», p. 6.

[31] J. M. Vara, V. de Castro, e E. Marcos, «WSDL Automatic Generation from UML Models in a MDA Framework», em *International Conference on Next Generation Web Services Practices (NWeSP'05)*, Seoul, Korea, 2005, pp. 319–324. doi: 10.1109/NWESP.2005.87.

[32] H. Abye, «Design and Implementation of Code Generator for Model-driven Software Development», p. 55.

[33] H. Benouda, M. Azizi, R. Esbai, e M. Moussaoui, «MDA Approach to Automate Code Generation for Mobile Applications», em *Mobile and Wireless Technologies 2016*, Singapore, 2016, pp. 241–250. doi: 10.1007/978-981-10-1409-3_27.

[34] A. Saurabh, D. Dahiya, e R. Mohana, «Maximizing Automatic Code Generation: Using XML Based MDA», em *Contemporary Computing*, vol. 306, M. Parashar, D. Kaushik, O. F. Rana, R. Samtaney, Y. Yang, e A. Zomaya, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 283–293. doi: 10.1007/978-3-642-32129-0_31.

[35] T. G. Moreira, M. A. Wehrmeister, C. E. Pereira, J.-F. Petin, e E. Levrat, «Automatic code generation for embedded systems: From UML specifications to VHDL code», em *2010 8th IEEE International Conference on Industrial Informatics*, Osaka, Japan, Jul. 2010, pp. 1085–1090. doi: 10.1109/INDIN.2010.5549590.

[36] A. G. Parada, E. Siegert, e L. B. de Brisolara, «Generating Java Code from UML Class and Sequence Diagrams», p. 3.

[37] S. E. V e P. Samuel, «Automatic Code Generation From UML State Chart Diagrams», *IEEE Access*, vol. 7, pp. 8591–8608, 2019, doi: 10.1109/ACCESS.2018.2890791.

[38] M. Usman e A. Nadeem, «Automatic Generation of Java Code from UML Diagrams using UJECTOR», *International Journal of Software Engineering and Its Applications*, p. 18, 2009.

[39] B. Vogel-Heuser, D. Witsch, e U. Katzke, «Automatic code generation from a UML model to IEC 61131-3 and system configuration tools», em *2005 International Conference on Control and Automation*, Jun. 2005, vol. 2, pp. 1034-1039 Vol. 2. doi: 10.1109/ICCA.2005.1528274.

[40] E. Umuhoza, H. Ed-douibi, M. Brambilla, J. Cabot, e A. Bongio, «Automatic Code Generation for Cross-Platform, Multi-device Mobile Apps: Some Reflections from an Industrial Experience», p. 8.

[41] A. Jakimi e M. Elkoutbi, «Automatic Code Generation FromUML Statechart». /paper/Automatic-Code-Generation-FromUML-Statechart-Jakimi-Elkoutbi/bda343b61d8279a5e01491307a6cfd7fdf272992 (acedido Mai. 20, 2021).

[42] I. Niaz, «Automatic Code Generation From UML Class and Statechart Diagrams», Jan. 2005.

[43] G. Pintér e I. Majzik, «Automatic Code Generation based on Formally Analyzed UML Statechart Models», Jan. 2003.

[44] J. Krizan, L. Ertl, M. Bradac, M. Jasansky, e A. Andreev, «Automatic code generation from Matlab/Simulink for critical applications», em *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, Toronto, ON, Canada, Mai. 2014, pp. 1–6. doi: 10.1109/CCECE.2014.6901058.

[45] D. Méry e N. K. Singh, «Automatic code generation from event-B models», em *Proceedings of the Second Symposium on Information and Communication Technology - SoICT '11*, Hanoi, Vietnam, 2011, p. 179. doi: 10.1145/2069216.2069252.

[46] M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, e S. Olivieri, «A Framework for Modeling, Simulation and Automatic Code Generation of Sensor Network Application», p. 8.

[47] S. D. Rathod, «Automatic code generation with business logic by capturing attributes from user interface via XML», em *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, Chennai, India, Mar. 2016, pp. 1480–1484. doi: 10.1109/ICEEOT.2016.7754929.

[48] P. Andersson e M. Höst, «UML and SystemC – A Comparison and Mapping Rules for Automatic Code Generation», em *Embedded Systems Specification and Design Languages: Selected contributions from FDL'07*, E. Villar, Ed. Dordrecht: Springer Netherlands, 2008, pp. 199–209. doi: 10.1007/978-1-4020-8297-9_14.

[49] A. A. Auer *et al.*, «Automatic code generation for many-body electronic structure methods: the tensor contraction engine», *Molecular Physics*, vol. 104, n. 2, pp. 211–228, Jan. 2006, doi: 10.1080/00268970500275780.

[50] M. M. Baskaran, J. Ramanujam, e P. Sadayappan, «Automatic C-to-CUDA Code Generation for Affine Programs», em *Compiler Construction*, Berlin, Heidelberg, 2010, pp. 244–263. doi: 10.1007/978-3-642-11970-5_14.

[51] L. B. Brisolara, M. F. S. Oliveira, R. Redin, L. C. Lamb, L. Carro, e F. Wagner, «Using UML as front-end for heterogeneous software code generation strategies», em *Proceedings of the conference on Design, automation and test in Europe*, New York, NY, USA, Mar. 2008, pp. 504–509. doi: 10.1145/1403375.1403494.

[52] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, e P. S. Yu, «Automatic code generation from design patterns», *IBM Systems Journal*, vol. 35, n. 2, pp. 151–171, 1996, doi: 10.1147/sj.352.0151.

[53] J. Cabot e E. Teniente, «Constraint Support in MDA Tools: A Survey», em *Model Driven Architecture – Foundations and Applications*, Berlin, Heidelberg, 2006, pp. 256–267. doi: 10.1007/11787044_20.

[54] M. Classen e M. Griebl, «Automatic code generation for distributed memory architectures in the polytope model», em *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, Abr. 2006, p. 7 pp.-. doi: 10.1109/IPDPS.2006.1639500.

[55] H. J. Ferreau, T. Kraus, M. Vukov, W. Saeys, e M. Diehl, «High-speed moving horizon estimation based on automatic code generation», em *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, Dez. 2012, pp. 687–692. doi: 10.1109/CDC.2012.6426428.

[56] V. García-Díaz, H. Fernández-Fernández, E. Palacios-González, B. C. Pelayo G-Bustelo, Ó. Sanjuan-Martínez, e J. M. Cueva Lovelle, «Automated code generation support for BI with MDA TALISMAN», Dez. 2009, Acedido: Mai. 19, 2021. [Em linha]. Disponível em: https://reunir.unir.net/handle/123456789/9357

[57] M. Gadaleta, M. Pellicciari, e G. Berselli, «Optimization of the energy consumption of industrial robots for automatic code generation», *Robotics and Computer-Integrated Manufacturing*, vol. 57, pp. 452–464, Jun. 2019, doi: 10.1016/j.rcim.2018.12.020.

[58] L. Geiger, C. Schneider, e C. Reckord, «Template- and modelbased code generation for MDA-Tools», em *PROCEEDINGS OF THE FUJABA DAYS 2005, VOLUME TR-RI-05-259 OF TECHNICAL REPORT.*, UNIVERSITY OF PADERBORN, 2005, pp. 57–62.

[59] F. Heidenreich, C. Wende, e B. Demuth, «A Framework for Generating Query Language Code from OCL Invariants», *Electronic Communications of the EASST*, vol. 9, n. 0, Art. n. 0, Nov. 2007, doi: 10.14279/tuj.eceasst.9.108.

[60] M. G. Hinchey, J. Rash, e C. A. Rouff, «Requirements to Design to Code: Towards a Fully Formal Approach to Automatic Code Generation», p. 26, 2005.

[61] R. Jardim-Goncalves, A. Grilo, e A. Steiger-Garcao, «Challenging the interoperability between computers in industry with MDA and SOA», *Computers in Industry*, vol. 57, n. 8, pp. 679–689, Dez. 2006, doi: 10.1016/j.compind.2006.04.013.

[62] «Automatic Code Generation for High-performance Discontinuous Galerkin Methods on Modern Architectures | ACM Transactions on Mathematical Software». https://dl.acm.org/doi/abs/10.1145/3424144 (acedido Mai. 19, 2021).

[63] Y. Khmelevsky, G. Hains, e C. Li, «Automatic code generation within student's software engineering projects», em *Proceedings of the Seventeenth Western Canadian Conference*

on Computing Education, New York, NY, USA, Mai. 2012, pp. 29–33. doi: 10.1145/2247569.2247578.

[64] D. Kundu, D. Samanta, e R. Mall, «Automatic code generation from unified modelling language sequence diagrams», *IET Software*, vol. 7, n. 1, pp. 12–28, Fev. 2013, doi: 10.1049/iet-sen.2011.0080.

[65] M. Lachgar e A. Abdali, «Generating Android graphical user interfaces using an MDA approach», em *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*, Out. 2014, pp. 80–85. doi: 10.1109/CIST.2014.7016598.

[66] G. Lasnier, B. Zalila, L. Pautet, e J. Hugues, «Ocarina : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications», em *Reliable Software Technologies – Ada-Europe 2009*, Berlin, Heidelberg, 2009, pp. 237–250. doi: 10.1007/978-3-642-01924-1_17.

[67] J. Mattingley e S. P. Boyd, «Automatic code generation for real-time convex optimization», 2010. doi: 10.1017/CBO9780511804458.002.

[68] M. I. Mukhtar e B. S. Galadanci, «Automatic code generation from UML diagrams: the state-of-the-art», *Science World Journal*, vol. 13, n. 4, Art. n. 4, 2018, doi: 10.4314/swj.v13i4.

[69] S.-A. Munoz, A. Steiner, e J. Farmer, «Processes of community-led social enterprise development: learning from the rural context», *Community Development Journal*, vol. 50, n. 3, pp. 478–493, Jul. 2015, doi: 10.1093/cdj/bsu055.

[70] M. Di Natale, F. Chirico, A. Sindico, e A. Sangiovanni-Vincentelli, «An MDA Approach for the Generation of Communication Adapters Integrating SW and FW Components from Simulink», em *Model-Driven Engineering Languages and Systems*, Cham, 2014, pp. 353–369. doi: 10.1007/978-3-319-11653-2_22.

[71] S. Philippi, «Automatic code generation from high-level Petri-Nets for model driven systems engineering», *Journal of Systems and Software*, vol. 79, n. 10, pp. 1444–1455, Out. 2006, doi: 10.1016/j.jss.2005.12.022.

[72] C. Pohl, C. Paiz, e M. Porrmann, «vMAGIC—Automatic Code Generation for VHDL», *International Journal of Reconfigurable Computing*, vol. 2009, p. e205149, Jul. 2009, doi: 10.1155/2009/205149.

[73] I. Reinhartz-Berger e D. Dori, «Object-Process Methodology (OPM) vs. UML: A Code Generation Perspective», p. 17, 2004.

[74] A. W. O. Rodrigues, F. Guyomarc'h, e J.-L. Dekeyser, «An MDE Approach for Automatic Code Generation from UML/MARTE to OpenCL», *Computing in Science Engineering*, vol. 15, n. 1, pp. 46–55, Jan. 2013, doi: 10.1109/MCSE.2012.35.

[75] A. Sabraoui, M. E. Koutbi, e I. Khriss, «GUI code generation for Android applications using a MDA approach», em *2012 IEEE International Conference on Complex Systems (ICCS)*, Nov. 2012, pp. 1–6. doi: 10.1109/ICoCS.2012.6458567.

[76] K. Sacha, «Automatic Code Generation for PLC Controllers», em *Computer Safety, Reliability, and Security*, Berlin, Heidelberg, 2005, pp. 303–316. doi: 10.1007/11563228_23.

[77] G. Sebastián, J. A. Gallud, e R. Tesoriero, «Code generation using model driven architecture: A systematic mapping study», *Journal of Computer Languages*, vol. 56, p. 100935, Fev. 2020, doi: 10.1016/j.cola.2019.100935.

[78] G. Sebastián, R. Tesoriero, e J. A. Gallud, «Automatic Code Generation for Language-Learning Applications», *IEEE Latin America Transactions*, vol. 18, n. 08, pp. 1433–1440, Ago. 2020, doi: 10.1109/TLA.2020.9111679.

[79] S. Teixeira, B. A. Agrizzi, J. G. P. Filho, S. Rossetto, e R. de L. Baldam, «Modeling and automatic code generation for wireless sensor network applications using model-driven or business process approaches: A systematic mapping study», *Journal of Systems and Software*, vol. 132, pp. 50–71, Out. 2017, doi: 10.1016/j.jss.2017.06.024.

[80] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, e P. Soulard, «IP reuse in an MDA MPSoPC co-design approach», em *2009 International Conference on Microelectronics - ICM*, Dez. 2009, pp. 256–259. doi: 10.1109/ICM.2009.5418638.

[81] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, e J.-P. Diguet, «A co-design approach for embedded system modeling and code generation with UML and MARTE», em *Automation Test in Europe Conference Exhibition 2009 Design*, Abr. 2009, pp. 226–231. doi: 10.1109/DATE.2009.5090662.

[82] M. Vukov, W. Van Loock, B. Houska, H. J. Ferreau, J. Swevers, e M. Diehl, «Experimental validation of nonlinear MPC on an overhead crane using automatic code generation», em *2012 American Control Conference (ACC)*, Jun. 2012, pp. 6264–6269. doi: 10.1109/ACC.2012.6315390.

[83] C. Xi, L. JianHua, Z. ZuCheng, e S. YaoHui, «Modeling SystemC design in UML and automatic code generation», em *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, New York, NY, USA, Jan. 2005, pp. 932–935. doi: 10.1145/1120725.1120760.

[84] Z. Liu, Y. Dou, J. Jiang, e J. Xu, «Automatic code generation of convolutional neural networks in FPGA implementation», em *2016 International Conference on Field-Programmable Technology (FPT)*, Dez. 2016, pp. 61–68. doi: 10.1109/FPT.2016.7929190.

[85] L. Zhu, N. B. Bui, Y. Liu, e I. Gorton, «MDABench: Customized benchmark generation using MDA», *Journal of Systems and Software*, vol. 80, n. 2, pp. 265–282, Fev. 2007, doi: 10.1016/j.jss.2006.10.052.

[86] M. Hummel, «State-of-the-Art: A Systematic Literature Review on Agile Information Systems Development», em *2014 47th Hawaii International Conference on System Sciences*, Jan. 2014, pp. 4712–4721. doi: 10.1109/HICSS.2014.579.

[87] A. Cockburn, «Writing Effective Use Cases», vol. 270., Jan. 2001.

[88] «Case Study Research in Software Engineering—It is a Case, and it is a Study, but is it a Case Study? | Elsevier Enhanced Reader». https://reader.elsevier.com/reader/sd/pii/S0950584921000033?token=713B89184BA05D113AC2DC7A933980867B35BF84DBCC241AB1C4A90CDCBD2091C7CE027AD9B0CA44B85F2C273060C985&originRegion=eu-west-1&originCreation=20210707081410 (acedido Jul. 07, 2021).