INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Recomendação de Pontos de Interesse para Grupos - Uma Abordagem baseada em Sistemas Multi-Agente e no Feedback dos Turistas

ANDRÉ BERNARDO MARTINS
Junho de 2023

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Recomendação de Pontos de Interesse para Grupos - Uma Abordagem baseada em Sistemas Multi-Agente e no Feedback dos Turistas

André Bernardo Martins

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Maria Goreti Carvalho Marreiros

Co-orientador: Patrícia Raquel de Jesus Araújo Alves

Júri:

Presidente:

Vogais:

Porto, 30 de junho de 2023

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 29 de junho de 2023

Dedicatória

À minha mãe.

Resumo

O protótipo de Sistema de Recomendação para Grupos no setor do turismo que se encontra em desenvolvimento pelo GECAD utiliza num dos seus microserviços um Sistema Multi-Agente. No entanto, a interação e partilha de conhecimento entre os agentes do sistema carece de melhorias que permitam a apresentação de recomendações cada vez mais coerentes, precisas e personalizadas.

Os dados sociodemográficos, personalidade e preferências turísticas de cada turista são modelados num agente inteligente que o representa, com o objetivo de tornar cada agente o mais similar possível ao turista que representa. Com isto, os agentes devem ter a capacidade de aprender com o conhecimento e ações dos outros agentes, assim como de partilhar dados de interação e perfil dos turistas, de forma a melhorar e a tornar mais precisas as recomendações enviadas pelo sistema de recomendações do GRS, melhorando a satisfação e experiência dos turistas.

Este documento apresenta uma análise de valor e inclui um estudo sobre o estado da arte em tecnologia relevante e sobre conceitos e trabalhos relacionados com o projeto em desenvolvimento. Apresenta ainda uma análise do domínio do problema, o desenho arquitetural e detalhes sobre a implementação e testagem do protótipo desenvolvido.

A solução final respondeu essencialmente a todas as necessidades que se tinham proposto resolver e possibilita o crescimento do GRS sem comprometer o trabalho já efetuado. Os utilizadores passaram a ser associados a clusters com base na sua personalidade e os respetivos agentes foram melhorados para utilizar *ratings* e *rules* que lhes diziam respeito de forma a priorizar e penalizar pontos de interesse turísticos nas recomendações obtidas.

Palavras-chave (Tema): Sistemas de Recomendação para Grupos, Turismo, Pontos de Interesse, Personalidade

Palavras-chave (Tecnologias): ActressMAS, C#, .NET, Sistemas Multi-Agente, Microserviços Multi-Agente

Abstract

The prototype of the Recommendation System for Groups (GRS) in the tourism sector that is being developed by GECAD uses a Multi-Agent System in one of its microservices. However, the interaction and knowledge sharing between the agents of the system needs improvements to allow the presentation of increasingly coherent, accurate and personalized recommendations.

The socio-demographic data, personality and tourist preferences of each tourist are modeled in an intelligent agent, with the goal of making each agent as similar as possible to the tourist it represents. With this, the agents should have the ability to learn from the knowledge and actions of the other agents, as well as to share interaction data and tourist profiles, in order to improve and make more accurate the recommendations sent by the GRS recommendation system, improving tourists' satisfaction and experience.

This paper presents a value analysis and includes a study of the state of the art in relevant technology and of concepts and work related to the project under development. It also presents an analysis of the problem domain, the architectural design, and details about the implementation and testing of the developed prototype.

The final solution essentially met all the needs that had been proposed to solve and allows for the growth of the GRS without compromising the work already done. Users are now associated to clusters based on their personality and the respective agents were improved to use ratings and rules related to them in order to prioritize and penalize tourist points of interest in the recommendations obtained.

Keywords (Topic): Recommendation Systems for Groups, Tourism, Points of Interest, Personality

Keywords (Technologies): ActressMAS, C#, .NET, Multi-Agent Systems, Multi-Agent Microservices

Agradecimentos

Gostaria de expressar minha profunda gratidão a todos aqueles que me apoiaram e contribuíram de maneira significativa para concluir o Mestrado. Em particular, gostaria de agradecer às professoras Maria Goreti Carvalho Marreiros e Patrícia Raquel de Jesus Araújo pelo acompanhamento ao longo do desenvolvimento do projeto, assim como pelo valioso *feedback* que melhorou o meu trabalho.

À minha família, especialmente à minha Mãe, pelos esforços incansáveis que fez e faz para me proporcionar a oportunidade de realizar este Mestrado e por sempre acreditar em mim.

Aos meus amigos, que estiveram ao meu lado nos bons e maus momentos ao longo da minha jornada académica.

Não posso esquecer de agradecer ao GECAD, pela oportunidade e experiência oferecida.

A todos os mencionados e a todos os que de alguma forma contribuíram para o meu crescimento e sucesso, quero expressar um sincero obrigado. Vocês foram essenciais na minha trajetória e sou imensamente grato por isso.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

ACL	Agent Communication Language
AHP	Analytic Hierarchy Process
APR	Reconhecimento automático de personalidade
BDI	Beliefs-Desires-Intentions
DI	Dependency Injection
DTO	Data Transfer Object
FFE	Fuzzy FrontEnd
FURPS+	*Functionality, Usability, Reliability, Performance, Supportability and more*
GECAD	Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e o Desenvolvimento
GRS	Sistemas de Recomendação para Grupos
IA	Inteligência Artificial
IC	Índice de Consistência
IO	Input/Output
IoC	Inversion-of-Control
ISEP	Instituto Superior de Engenharia do Porto
JS	JavaScript
JVM	Máquinas virtuais Java
MAMS	Microserviços Multi-Agente
MAS	Sistemas Multi-Agente
MEI-ES	Mestrado em Engenharia Informática – Ramo de Engenharia de Software

MS	Microserviço
MVC	Model-View-Controller
NCD	New Concept Development
NPD	New Product Development
ORM	*Object-relational mapper*
RC	Razão de Consistência
REST	REpresentional State Transfer
RL	*Reinforcement Learning*
RS	Sistemas de recomendação
SOA	Arquitetura orientada a serviços
SOAP	Single Object Access Protocol
TMDEI	Unidade curricular Tese/Dissertação/Estágio
n	Ordem da matriz
λ_{max}	Maior valor próprio da matriz de comparação

1 Introdução

O presente capítulo, após contextualizar o leitor acerca da origem do trabalho, disponibiliza um breve enquadramento do projeto, descrevendo o problema que se pretende resolver e os objetivos a atingir. Explica ainda a abordagem utilizada e descreve o planeamento realizado para a execução do projeto, assim como os seus contributos.

1.1 Contexto

O presente trabalho foi realizado no âmbito da unidade curricular Tese/Dissertação/Estágio (TMDEI) do Mestrado em Engenharia Informática – Ramo de Engenharia de Software (MEI-ES) do Instituto Superior de Engenharia do Porto (ISEP).

Este projeto foi escolhido por permitir aplicar não só novos conceitos de programação aprendidos no mestrado, aprimorando os conhecimentos na área da Inteligência Artificial (IA), nomeadamente Sistemas Multi-Agente (MAS), mas também para perscrutar novas tecnologias que complementassem as já conhecidas.

Este trabalho inseriu-se num projeto que se encontra em desenvolvimento pelo Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e o Desenvolvimento (GECAD), cuja missão culmina no desenvolvimento de inovação e pesquisa científica para a incorporação de inteligência na engenharia e em sistemas computacionais complexos, e alia à aplicação dos conhecimentos informáticos adquiridos, o desafio de enfrentar novas tecnologias e aplicabilidades de soluções de software inteligentes em sistemas de recomendação (RS), nomeadamente num protótipo *mobile* de um RS para grupos de turistas, denominado Grouplanner.

1.2 Descrição do Problema

Atualmente tem-se verificado um rápido crescimento de serviços e conteúdo online, o que dificulta cada vez mais aos utilizadores a escolha adequada de vários itens, desde filmes, séries e livros, a restaurantes e outros pontos de interesse (POI) turísticos. Os sistemas de recomendação procuram indicar aos utilizadores, sem os sobrecarregar, produtos específicos com base nos seus interesses individuais. Os RS tradicionais fazem recomendações baseadas no histórico de comportamentos dos utilizadores e esta abordagem tem sido muito bem-sucedida, porém, limitada particularmente para novos utilizadores do sistema face à escassez inicial de informação relevante no sistema (problema do arranque a frio, *cold-start*), o que pode dar origem a recomendações menos precisas e personalizadas [1].

Os RS são uma ferramenta cada vez mais importante para ajudar os utilizadores a fazer melhores escolhas numa variedade de domínios, mas quando grupos de pessoas estão envolvidas, surgem dissensões de preferências que muitas vezes se deve à sua heterogeneidade. Para ultrapassar este tipo de situações, em especial no turismo, estão a ser propostos Sistemas de Recomendação para Grupos (GRS).

No entanto, ainda podem ser encontradas muitas limitações, tanto nas configurações morosas, como na excessiva intrusividade necessária para criar o perfil dos turistas e propor recomendações de atrações a visitar numa determinada excursão/roteiro de viagem. Alguns autores têm proposto a utilização de MAS para responder a estas limitações, fazendo recomendações de forma proativa com base no perfil e contexto dos turistas [2].

Devido à sua autonomia, estado isolado, e distribuição, os MAS trazem grandes vantagens aos RS [3], permitindo o desenvolvimento de sistemas distribuídos [4], mais inteligentes e com a capacidade de aprender sobre as preferências dos turistas de forma não intrusiva. O protótipo de GRS para turismo que se encontra em desenvolvimento pelo GECAD utiliza num dos seus microserviços (MS) um MAS. Contudo, os agentes deste sistema necessitam de uma melhoria significativa a nível da interação e partilha de conhecimento entre eles que permita a apresentação de recomendações cada vez mais coerentes, precisas e personalizadas.

1.3 Objetivos

Com este projeto, pretendeu-se melhorar as capacidades e interação dos agentes do MAS existente no Microserviço Multi-Agente (MAMS) do protótipo de um Sistema de Recomendação para Grupos de Turismo, Grouplanner, que se encontra em desenvolvimento pelo GECAD [5, 6, 7].

Os dados sociodemográficos, personalidade e preferências turísticas de cada turista deviam ser modelados num agente inteligente que o representasse, com o objetivo de tornar cada agente o mais similar possível ao respetivo turista. Com isto, os agentes deveriam ter a capacidade de aprender com o conhecimento e ações dos outros agentes, assim como de partilhar dados de interação e perfil dos turistas, de forma a melhorar e a tornar mais precisas as recomendações enviadas pelo sistema de recomendações do GRS, melhorando a satisfação e experiência dos turistas. Deste modo, foram estabelecidos os seguintes objetivos:

1. Contextualização sobre o estado de arte de vários tópicos, nomeadamente: Sistemas de Recomendação para Grupos que consideram a personalidade e preferências do utilizador, Sistemas Multi-Agente e Microserviços Multi-Agente de forma a adquirir conhecimento suficiente para prosseguir ao estudo do MAMS já existente no protótipo do GRS, visando a melhoria das funcionalidades de acordo com o que foi descrito nesta secção.
2. Planeamento e Análise de Requisitos para perceber de que forma as várias componentes do GRS vão interagir. Estas interações devem ser devidamente representadas ao nível do Design da Solução.
3. Melhoria dos agentes inteligentes existentes e desenvolvimento das suas capacidades de aprendizagem, interação e de partilha do conhecimento paralelamente à atualização do tratamento de pedidos e respostas de recomendações fornecidas pelo motor de recomendações do GRS.
4. Testagem das funcionalidades recorrendo a testes de integração do MAMS com o protótipo do GRS de modo a verificar a boa comunicação entre os vários microserviços, testes de carga para aferir a capacidade de resposta da aplicação a pedidos de recomendação de vários utilizadores em simultâneo e testes de aceitação para identificar problemas e aspetos a melhorar com base na experiência de utilizadores reais durante a utilização da aplicação.

1.4 Abordagem

Decidiu-se adotar uma abordagem de desenvolvimento de *software* usando uma metodologia ágil recorrendo à *framework* Scrum.

Scrum é uma *framework* através da qual as pessoas podem solucionar problemas de adaptação complexos através do desenvolvimento incremental e iterativo de produtos, valorizando assim o que é entregue [8, 9].

O processo de desenvolvimento é baseado em iterações (*Sprints*) cujo objetivo passa por criar software pronto a utilizar. Inicialmente, cria-se um *Product Backlog* em que se listam todas as características de software a implementar com base na sua prioridade. Posto isto, as tarefas a desenvolver num dado *Sprint* são transferidas para o respetivo *Sprint Backlog* durante o *Sprint Planning*. É importante mencionar que não é possível adicionar tarefas ao *Sprint Backlog* se o *sprint* em questão já tiver iniciado. No fim de cada *Sprint* são realizados *Sprint Reviews* para avaliar o progresso e os resultados obtidos que podem fundamentar a adaptação do *Product Backlog* [8, 9].

Os *Sprints* utilizados possuíam a duração de 15 dias ao longo dos quais se realizaram reuniões frequentes de forma a esclarecer dúvidas que podiam atrasar de alguma forma a conclusão das tarefas definidas. Os *Sprint Reviews e Sprint Plannings* foram realizados no dia seguinte ao fecho de cada *Sprint* em reuniões com a equipa de desenvolvimento e *Scrum Master*.

A gestão do *Product Backlog* e do *Sprint Backlog* foi feita através da ferramenta Excel visto que a sua utilização foi imposta pelo GECAD. A definição dos requisitos iniciais do projeto já se encontrava previamente elicitada pelo GECAD, tendo este trabalho sido realizado de acordo com as *user stories* existentes.

O controlo de versões da solução foi efetuado com recurso ao sistema de versionamento Git usando um repositório Bitbucket [10].

1.5 Planeamento do trabalho

Desde o dia 1 de novembro até dia 30 de junho que o relatório foi escrito paralelamente à implementação das melhorias efetuadas no protótipo de GRS. A primeira etapa efetuada consistiu num estudo do estado da arte de tecnologias e conceitos relacionados com o tema do projeto. Esta tarefa durou desde o dia 1 de novembro até o dia 14 de novembro.

A segunda etapa focou-se na testagem das tecnologias utilizadas no desenvolvimento do MAMS e no estudo do funcionamento do protótipo de GRS. Esta tarefa prolongou-se por duas semanas e finalizou-se no dia 28 de novembro.

A identificação dos vários requisitos, a respetiva análise e o desenvolvimento do desenho da solução ocorreram na terceira etapa entre os dias 28 de novembro e 13 de dezembro. Por fim, a implementação desses requisitos prolongou-se até o dia 1 de junho.

1.6 Estrutura do relatório

À exceção do capítulo da Introdução, o presente relatório conta com mais seis capítulos. O segundo capítulo contextualiza o leitor acerca dos conceitos e trabalhos relacionados com o protótipo do Sistema de Recomendação para Grupos de Turismo e sobre as tecnologias selecionadas para desenvolver a interação e partilha de conhecimento entre os agentes que permita a apresentação de recomendações cada vez mais coerentes, bem como as suas concorrentes.

O terceiro capítulo (Análise de Valor) descreve um estudo dos métodos de análise do valor a utilizar num projeto, aplicando-os diretamente à solução proposta, obtendo-se uma proposta de valor final.

O quarto capítulo abarca o processo de compreensão do problema, onde é feita uma análise ao domínio do sistema a desenvolver, e disponibiliza uma secção que alude aos requisitos funcionais e não funcionais da solução. Por fim, apresenta o design estipulado com base em artefactos ilustrativos.

O quinto capítulo possui pormenores sobre a implementação da solução. Apresenta uma descrição inicial acerca do funcionamento geral do MAMS e, posteriormente, analisa detalhadamente os casos de uso implementados/melhorados. Os testes desenvolvidos são descritos no final do capítulo.

O sexto capítulo procura avaliar a qualidade do protótipo de sistema de recomendação de POI melhorado. Começa por definir as hipóteses de investigação a comprovar e descreve a metodologia de avaliação utilizada. Por fim, é feita a análise dos resultados.

Por fim, o último capítulo (Conclusão) realiza uma avaliação ao trabalho desenvolvido, mediante a apresentação dos objetivos atingidos. Descreve ainda as direções a seguir num desenvolvimento futuro em conjunto com algumas limitações encontradas. Por fim, apresenta uma apreciação pessoal acerca do trabalho realizado.

2 Estado da arte

Este capítulo apresenta e descreve o estado da arte em tecnologia relevante e os conceitos e trabalhos relacionados com o projeto em desenvolvimento.

2.1 Conceitos

Na presente secção, são apresentados os conceitos relacionados com o projeto.

2.1.1 Sistemas de Recomendação

Sistemas de recomendação são sistemas capazes de prever se um dado utilizador prefere ou não um dado item através da filtragem da informação gerada dinamicamente com base nas preferências, interesses e historial de comportamentos do utilizador. Estes sistemas procuram resolver o problema relativamente ao excesso de informação a que os utilizadores são expostos mediante recomendações personalizadas e exclusivas de serviços e conteúdo [11].

O processo de recomendação utilizado pela maioria dos RS utiliza uma sequência de passos semelhante. Começa com a criação de um perfil para o utilizador do sistema que resulta da análise da sua informação disponível. Este perfil armazena os dados necessários para a escolha e posterior apresentação dos itens que mais se adequam ao utilizador atual. Por fim, disponibiliza um mecanismo de *feedback* que permite aferir a satisfação dos utilizadores relativamente às recomendações apresentadas com o objetivo de ajustar os seus respetivos perfis, baseando-se nas suas opiniões [12].

A recolha do feedback pode ser feita através de métodos implícitos ou explícitos [12, 13]:

- Método Implícito: obtém o *feedback* dos utilizadores indiretamente, tendo por base as suas interações com os itens. Esta abordagem consegue obter *ratings* baseando-se, por exemplo, na quantidade de tempo que o utilizador passa a visualizar um determinado item e nos dados relativos aos "cliques". O ponto fraco desta abordagem provém da quantidade de tempo e dados necessários para obter feedback de forma eficiente e fidedigna;
- Método Explícito: obtém o feedback dos utilizadores perguntando diretamente aos utilizadores para escolherem um valor numa escala finita de pontos (e.g, "Avalie este restaurante de um a cinco") ou através de questionários onde os utilizadores podem expressar a sua opinião acerca de um determinado item. Esta abordagem faz com que as opiniões dos utilizadores obtidas, acerca das suas preferências, sejam mais fidedignas, contudo, pode reduzir a qualidade da experiência de utilização do sistema já que necessita de participação ativa.

A maioria dos RS optam por recolher o *feedback* através da utilização de ambos os métodos em paralelo [13].

Dependendo da técnica de filtragem de recomendações utilizada pelo RS, é possível categorizá-lo em três tipos de sistemas distintos: Sistemas baseados em Conteúdo, Sistemas Colaborativos e Sistemas Híbridos [11, 13]. A Figura 1 apresenta um diagrama das diferentes técnicas de filtragem de recomendações que um RS pode utilizar.

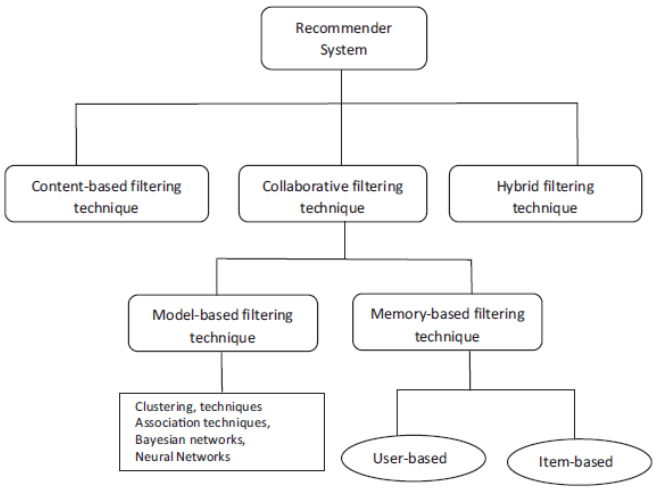

Figura 1 – Técnicas de Filtragem de Recomendações, adaptado de [11]

2.1.1.1 Sistemas Baseados em Conteúdo

Os Sistemas baseados em Conteúdo utilizam uma técnica dependente do domínio, isto é, o perfil do utilizador é construído através das características provenientes do conteúdo de itens que o utilizador avaliou anteriormente. Por exemplo, se o sistema recomenda filmes, o realizador e os atores que participam nas obras cinematográficas são características que se podem analisar. Os itens que estiverem mais relacionados com os itens avaliados positivamente são recomendados ao utilizador [11, 13,14].

Este tipo de técnica necessita de um conhecimento profundo das características dos itens para garantir recomendações fidedignas e esta informação nem sempre está disponível. Para além disso, os utilizadores podem ficar limitados a recomendações de itens semelhantes aos que estão definidos nos seus perfis. Contudo, também existem vantagens, nomeadamente a capacidade de conseguir recomendar novos itens mesmo que não possuam qualquer avaliação prévia. Basta possuir uma descrição completa o suficiente do item para comparar as suas características com as de outros itens já avaliados até encontrar semelhanças. Esta abordagem possibilita a rápida adaptação às mudanças de preferências dos utilizadores e, uma vez que os utilizadores não necessitam de partilhar os seus perfis para obter recomendações, a privacidade é garantida. Outro ponto forte destes sistemas é a capacidade

de apresentar explicações sobre como as recomendações foram geradas, algo que é valorizado pelos utilizadores que por norma gostam de sentir que compreendem como o sistema funciona [5, 11, 13, 14].

2.1.1.2 Sistemas Colaborativos

Os Sistemas Colaborativos utilizam uma técnica independente do domínio, utilizada maioritariamente nos casos em que o conteúdo dos itens é difícil de caracterizar ao contrário do que acontece, por exemplo, nos filmes. Esta abordagem começa por encontrar um grupo de utilizadores cujas preferências e gostos são semelhantes aos de um dado utilizador Y, apelidado de vizinhança. Os itens que a maioria dos membros da vizinhança avaliaram melhor, são recomendados ao utilizador Y [11, 13].

Este tipo de sistemas apresenta tradicionalmente problemas de privacidade devido à partilha de informação entre utilizadores e problemas relativos à insuficiência de informação disponível no sistema, por exemplo, se um novo filme fosse adicionado ao sistema, como esta técnica não se foca na comparação de características semelhantes entre itens, a inexistência de *feedback* relativo a esse novo filme tornaria a sua recomendação impossível (*cold-start*). Contudo, para casos em que exista bastante *feedback* no sistema, a recomendação de itens tem uma margem de diversidade muito maior comparativamente a técnicas baseadas no conteúdo. A eficiência desta técnica de filtragem de recomendações depende da qualidade do algoritmo em encontrar a vizinhança dos utilizadores do sistema [1, 11, 13, 14].

Esta técnica pode ser ainda classificada como baseada em memória ou baseada em modelos dependendo da sua abordagem. Os sistemas colaborativos baseados em memória utilizam os *ratings* atribuídos por utilizadores da vizinhança de um utilizador Y a um dado item X para definir o *rating* do utilizador Y relativamente ao item X (*user-based*), ou utiliza a média dos *ratings* atribuídos pelo utilizador Y a itens idênticos (*item-based*). Os sistemas colaborativos baseados em modelos fornecem recomendações através de modelos construídos com o *feedback* do utilizador. Estes modelos são obtidos através de técnicas de *machine learning* e *data mining* e ajudam a solucionar problemas relacionados com escalabilidade e esparsidade [11, 13, 15].

2.1.1.3 Sistemas Híbridos

Os Sistemas Híbridos combinam diferentes técnicas de filtragem de recomendações para conseguir preencher as lacunas resultantes da sua utilização individual. A Tabela 1 analisa algumas das abordagens híbridas existentes [13, 14].

Tabela 1 – Abordagens Híbridas de Filtragem de Recomendações, adaptado de [13, 14]

Abordagens Híbridas	Descrição
Weighted	Utiliza o *feedback* obtido por diferentes técnicas, ajustando o peso de cada uma ao longo do processo de classificação de um dado item.
Mixed	Agrega as recomendações de várias técnicas e recomenda o conjunto de recomendações resultante.
Switching	Utiliza uma única técnica que altera conforme o contexto.

Feature Combination	Combina características de uma dada técnica noutra.
Feature Augmentation	Utiliza informação de um dado item resultante da recomendação de uma técnica para ser utilizada como input numa outra técnica.
Meta-Level	Utiliza toda a informação conhecida de um dado item por uma dada técnica para ser utilizada como input numa outra técnica.
Cascade	Utiliza uma técnica com prioridade inferior a uma outra técnica de maior prioridade para melhorar as recomendações da hierarquicamente superior.

A eficácia dos Sistemas de Recomendação é uma das características que se tem procurado melhorar ao longo dos anos. O cálculo da personalidade enriqueceu estes sistemas quanto à quantidade de informação conhecida sobre os utilizadores, possibilitando uma nova forma de obter as suas preferências e contribuindo para a resolução de problemas como o *cold-start*. As técnicas de filtragem de recomendação dos RS que têm em conta a personalidade, apenas diferem das convencionais porque utilizam, adicionalmente, características psicológicas dos utilizadores [6, 16].

O cálculo correto da personalidade dos utilizadores é essencial para garantir a fidedignidade dos RS. Atualmente, os dois métodos basilares utilizados nesta medição, são as técnicas de reconhecimento automático de personalidade (APR) e os questionários de avaliação de personalidade. Os questionários costumam aferir o caracter e os comportamentos estimados dos utilizadores. Há uma tendência em optar pelos questionários que são menos longos, para evitar que os utilizadores forneçam respostas erradas devido à possível falta de concentração provocada por grandes quantidades de questões. Contudo, nem sempre é oportuno realizar estes questionários. A utilização de técnicas de APR, que visam associar dados relativos a um determinado assunto, a uma medição de personalidade de um utilizador, é o método mais indicado nestes contextos [16, 17].

Particularmente no domínio do turismo, a utilização da personalidade na recomendação de POI tem um impacto positivo, mesmo quando se utilizam apenas algumas das suas características [6].

2.1.2 Sistemas de Recomendação para Grupos

Quando as recomendações se dirigem a grupos de pessoas, surgem conflitos de preferências e heterogeneidade e alguns aspetos devem ser tidos em contra [5, 18, 19]:

- Os utilizadores dos RS apreciam a apresentação de justificações relativas às recomendações que lhes são apresentadas, porém, gerar uma explicação válida que seja compreendida por todos os membros de um grupo é um desafio muito mais complicado devido a possíveis conflitos de interesses existentes;
- Na secção 2.1.1 verificou-se que as preferências dos utilizadores podem ser obtidas através de métodos implícitos e explícitos, contudo, quando grupos estão envolvidos

no processo de recomendação, este *feedback* pode ser influenciado pelas motivações dos membros constituintes do grupo. Por exemplo, um utilizador pode preferir sair à noite numa discoteca no Porto se estiver a visitar a cidade com um grupo de amigos, mas já não possuir a mesma preferência se o grupo for constituído por desconhecidos;

- O processo de escolha após a apresentação das recomendações nos GRS é muito mais complexo relativamente aos RS, onde apenas um utilizador está encarregue da decisão final. É necessário chegar a um consenso entre todos os elementos do grupo o que nem sempre é possível e, em muitos dos casos, por exemplo no domínio do turismo, utilizadores aceitam visitar locais que não lhes agradam apenas para se conformarem com o resto do grupo e não se sentirem julgados.

Existem algumas estratégias para agregar as opiniões individuais de modo a alcançar recomendações satisfatórias para todos os membros do grupo. A Tabela 2 apresenta as estratégias mais utilizadas [19, 20].

Tabela 2 – Estratégias de agregação de opiniões individuais em grupos, adaptado de [19, 20]

Estratégia	**Descrição**
Most Pleasure	Utiliza o maior *rating* atribuído por um elemento do grupo como o *rating* preferencial de todo o grupo.
Least Misery	Utiliza o menor *rating* atribuído por um elemento do grupo como *rating* preferencial de todo o grupo.
Dictatorship	Utiliza sempre o *rating* atribuído por um elemento específico do grupo como *rating* preferencial de todo o grupo.
Average	Utiliza a média dos *ratings* individuais como *rating* preferencial de todo o grupo.
Approval Voting	Utiliza o valor resultante da contagem dos *ratings* individuais que possuem valor superior a um dado valor mínimo como *rating* preferencial de todo o grupo.
Borda Count	Atribuem-se pontos aos *ratings* individuais dos utilizadores do grupo. Quanto melhor classificado um item estiver relativamente a outros itens do mesmo utilizador, maior será a sua pontuação. Por fim, somam-se as pontuações dos vários elementos do grupo e o item com mais pontos será o preferencial.
Copeland Rule	O item com o maior resultado da subtração do número de vezes que possui melhor rating que outras opções com o número de vezes que possui rating inferior a outras opções, será escolhido.
Utilitarian Strategy	Junta os *ratings* atribuídos pelos membros do grupo a cada item e soma-os (*Additive*) ou multiplica-os (*Multiplicative*). O item com maior valor resultante será escolhido.
Plurality Voting	Cada utilizador vota no seu item favorito e aquele que obtiver a maioria dos votos é escolhido.

Apesar de não existirem muitas diferenças entre nas performances de cada estratégia, existe quem defenda que a *Multiplicative Utilitarian Strategy* apresenta um melhor desempenho [19, 20].

Para responder às limitações apresentadas na presente secção, foi proposta a utilização de Sistemas Multi-Agente [2, 5].

2.1.3 Sistemas Multi-Agente

A utilização de agentes no desenvolvimento de software é considerado um dos mais importantes paradigmas desde a aparição da programação orientada a objetos. Um agente pode ser definido como uma entidade capaz de ajudar humanos na realização de determinadas tarefas, cujas propriedades mais importantes são a inteligência, cooperação, autonomia e a capacidade de aprender e reagir atempadamente às mudanças do ambiente em que atuam [21, 22, 91].

O ambiente em que os agentes se localizam fornece-lhes informação para tomadas de decisão e possui várias características que afetam a dinâmica dos sistemas baseados em agentes [23]:

- Determinismo: Um ambiente pode ser classificado como determinístico se possibilitar a previsão dos resultados de uma dada ação, ou não determinístico se não possibilitar essa previsão;
- Acessibilidade: Este parâmetro refere-se à precisão com que os agentes conseguem obter informação do ambiente. Quanto mais acessível um ambiente é, mais precisa e completa é a informação que os agentes obtêm;
- Dinamismo: Um ambiente pode ser considerado estático se apenas sofrer modificações como consequência das ações dos agentes, mas caso um ambiente altere por mais algum motivo, este pode ser classificado como dinâmico. De notar que, nos ambientes dinâmicos, os agentes necessitam de ter um cuidado extra em manter a informação que obtêm atualizada, já que as mudanças no ambiente não dependem exclusivamente das suas ações.

Um único agente não possui informação suficiente para realizar por si só uma tarefa por completo. Deste modo, deve-se combinar agentes para resolver o problema [21].

Os agentes podem atingir o seu potencial máximo se colaborarem entre si para atingir um objetivo comum. Os sistemas que utilizam esta abordagem designam-se MAS e apresentam algumas vantagens relativamente à utilização isolada de um único agente. Os MAS permitem diminuir erros e o custo necessário para resolver uma dada tarefa, alcançar a estabilidade e escalabilidade necessária, implementar processamento paralelo e distribuir as tomadas de decisão [24, 89]. Este tipo de sistemas pode ser classificado quanto à heterogeneidade dos seus agentes. Se todos os agentes possuírem as mesmas

funcionalidades e características, o MAS classifica-se como homogéneo, caso contrário, classifica-se como heterogéneo [23].

Apesar das características positivas que os MAS oferecem, existem alguns aspetos que se devem ter em atenção. Nos MAS, a comunicação entre os agentes pode variar no seu grau de complexidade. Esta consegue ser feita através de sinais com interpretações pré-definidas ou através de *blackboards*. Uma *blackboard* consiste num recurso partilhado onde os agentes colocam ou obtêm informação das suas ações. A comunicação pode ainda ser efetuada através de troca direta de mensagens entre os agentes [25].

A coordenação é outro fator relevante dos MAS, uma vez que é indispensável para que os agentes consigam atingir os seus objetivos. De modo a garantir que o sistema funcione de acordo com o que é espectável, os agentes necessitam de ter em conta as ações de outros agentes do sistema antes de planear e executar as suas ações. O termo *consensus* representa a iteração entre grupos de agentes necessária para alcançar um acordo relativamente a um valor comum. Quanto maior for a heterogeneidade entre os agentes, mais difícil será a sua sincronização e respetivo *consensus*. A realidade é que a coordenação pode implicar a competição por um dado recurso entre agentes com objetivos distintos, ou pode gerar cooperação entre agentes com objetivos comuns. As características do ambiente do MAS influenciam a complexidade da coordenação dos agentes [23, 24, 25, 90].

A aprendizagem nos MAS é um fator que dificulta as tarefas a nível da coordenação e *consensus* entre agentes devido às mudanças de comportamentos derivadas da constante aquisição de conhecimento. Contudo, o potencial da aprendizagem é notório visto que permite que o sistema se adapte dinamicamente a vários problemas em que não é possível conhecer o comportamento de outros agentes no ambiente [26].

Os agentes podem possuir conflitos de interesses, e as alterações nos comportamentos resultantes da aprendizagem de um agente podem interferir com as políticas já estabelecidas de outros, ou seja, o processo de aprendizagem possui um maior nível de complexidade nos MAS comparativamente à utilização de um único agente. Um dos métodos mais comumente utilizados na aprendizagem dos agentes nos MAS é o *Reinforcement Learning* (RL). O RL é um método de tentativa e erro onde o agente interage continuamente com o ambiente em que atua, efetuando ações e analisando as suas consequências. As ações que originarem efeitos positivos vão ter maior probabilidade de serem repetidas no futuro e vice-versa [27].

A alocação de tarefas entre agentes deve ser devidamente efetuada de acordo com o tempo e custos associados. A atribuição das tarefas pode ser feita de acordo com o número total de recursos do agente, isto é, deve-se ter em conta o balanço atual de tarefas alocadas ao agente antes de lhe atribuir uma nova. Quando mais ocupado um agente estiver, mais demoradas serão as conclusões das tarefas. Outra abordagem de alocação considera o posicionamento dos agentes, ou seja, se existir conhecimento prévio de que uma dada tarefa necessita de utilizar recursos de alguns agentes, então será

preferível que se atribua a tarefa a um agente que se encontre o mais perto possível desses agentes [23].

Os MAS trazem grandes vantagens aos RS graças à sua autonomia, estado isolado e distribuição, propriedades que coincidem com as características dos microserviços. Face a estas semelhanças, propôs-se a integração dos dois conceitos, resultando nos Microserviços Multi-Agente [3].

2.1.4 Microserviços Multi-Agente

2.1.4.1 Microserviços

As aplicações desenvolvidas segundo uma arquitetura monolítica acabam por trazer vários problemas sempre que atingem grandes proporções. A manutenção da aplicação fica muito mais dispendiosa, tornando-se praticamente impossível fazer qualquer alteração funcional sem que a mínima mudança num componente afete negativamente o normal funcionamento de outros [28, 29].

A arquitetura em microserviços consiste em estruturar uma aplicação em vários serviços independentes entre si, com responsabilidades únicas e distintas. Estes serviços independentes trazem várias vantagens já que contribuem para uma maior manutenibilidade, testabilidade e escalabilidade e para um baixo acoplamento. O conceito de escalabilidade refere-se à escalabilidade do sistema em *runtime,* em que cada microserviço pode ser escalado de forma distinta conforme os seus requisitos, mas também se refere à escalabilidade de cada microserviço como unidade independente de desenvolvimento e *deployment*. Desta forma, cada microserviço pode estar ao comando de equipas de desenvolvimento diferentes, possibilitando a adição paralela de novas características à aplicação, fruto do seu estado isolado. Estas particularidades dos microserviços facilitam a entrega rápida e frequente de aplicações grandes e complexas. Outra vantagem importante é o facto de permitir a utilização de diferentes tecnologias e bases de dados independentes nos vários microserviços e, desta forma, aumenta *stack* tecnológica das organizações [30,31].

A arquitetura orientada a serviços (SOA), que por norma utiliza o protocolo de comunicação Single Object Access Protocol (SOAP), apresenta características semelhantes à arquitetura orientada a microserviços, no entanto, a arquitetura em microserviços utiliza REpresentional State Transfer (REST), um protocolo de comunicação mais simples e leve, nativo para desenvolvimento web. Apesar do aumento da popularidade dos microserviços, nem sempre a adoção desta arquitetura é a melhor opção, pois os custos podem ultrapassar os benefícios. Se o sistema a desenvolver não justificar, deve-se optar por uma arquitetura monolítica. Normalmente os sistemas pequenos não necessitam da complexidade extra de roteamento, monitorização, implementação e *deployment* que advêm da utilização de microserviços [29,31], o que não é o caso do protótipo de GRS utilizado, que utiliza microserviços e se encontra melhor descrito nas secções 4.3 e 5.1.

2.1.4.2 Microserviços e MAS

As semelhanças existentes entre os microserviços e os agentes que atuam nos MAS são notórias e podem ser verificadas através da comparação de propriedades apresentada na Tabela 3.

Tabela 3 – Comparação de propriedades entre microserviços e MAS, adaptado de [3, 32]

Propriedade	Microserviços	Agentes no MAS
Autonomia	Conseguem realizar as suas tarefas sem necessidade de intervenção humana.	Conseguem realizar as suas tarefas sem necessidade de intervenção humana.
Reatividade	Respondem a pedidos HTTP que lhes são efetuados.	Reagem às mudanças no ambiente em que atuam.
Proatividade	Não têm iniciativa, limitam-se a reagir.	Para além de reagirem, também têm iniciativa.
Sociabilidade	Interagem, colaboram e coordenam-se entre si através de RESTful API e HTTP.	Interagem, colaboram e coordenam-se entre si através das abordagens de comunicação mencionadas na secção 2.1.3.
Acoplamento	Os sistemas que seguem a arquitetura de microserviços são decompostos em serviços com baixo acoplamento.	Resolvem problemas relacionados com baixo acoplamento.
Manutenção Automatizada	Operações de manutenção são automáticas (p.e. tratamento de erros e escalonamento)	Operações de manutenção não são o foco dos agentes apesar de poderem ser consideradas.
Elasticidade	Permite a adição e remoção de recursos em *runtime*.	Permite a adição e remoção de agentes em *runtime*.
Distribuição	O código é distribuído pelos serviços.	O código é distribuído em conjunto com conhecimento e inteligência.
Tamanho	Devem ser pequenos o suficiente para facilitar a manutenibilidade e escalabilidade.	Define-se por norma com base no domínio do problema, sem grandes restrições.
Bounded Context	Representa uma parte única do negócio.	Pode efetuar várias tarefas a nível do negócio.
Aprendizagem e Adaptação	Não são capazes de aprender nem de se adaptar ao contexto.	São capazes de aprender e de se adaptar ao contexto.

Na secção 2.1.3 constatou-se que os agentes atingem o seu potencial máximo quando trabalham em conjunto, através da troca de mensagens. Os microserviços, por outro lado, procuram minimizar o número de comunicações existentes. Para isso, os microserviços normalmente disponibilizam vários recursos, distribuídos entre os serviços com base numa boa divisão de responsabilidades. Apesar dos agentes serem muito mais que meros recursos, podem funcionar como recursos na sua coexistência com microservicos, isto é, alocando-os a

um único microserviço. Um agente pode ser exposto como um recurso ou pode implementar recursos. Por exemplo, um agente pode funcionar como uma API de um dado recurso num microserviço enquanto os restantes agentes apenas se encarregam dos detalhes da implementação [3].

O aperfeiçoamento das recomendações individuais e para grupos na área do turismo através da utilização de MAMS tem sido investigado pelo GECAD [7]. A exposição de agentes modelados com o perfil de turistas num microserviço, através de um único *endpoint* REST e a utilização da capacidade de comunicação característica dos agentes, facilita e aumenta a rapidez na troca de informação dos turistas. Estas características vantajosas revelam que a utilização de MAMS é uma solução ideal para o GRS em desenvolvimento [7].

2.2 Trabalhos Relacionados

De forma a melhorar a experiência de grupos de turistas quanto ao planeamento de viagens e excursões, nos últimos anos têm sido apresentadas várias propostas de GRS no domínio do turismo. Neste domínio, existem muitos RS que utilizam MAS para melhorar as recomendações apresentadas aos utilizadores, mas existe pouca literatura acerca da integração de MAS em GRS [2, 6].

O *e-Tourism* é um MAS que fornece recomendações individuais e para grupos em excursão na cidade de Valencia com base nas preferências de visita, gostos e classificações demográficas dos utilizadores. O sistema considera também as classificações atribuídas pelos utilizadores em visitas prévias e, nas recomendações de grupo, procura satisfazer da forma mais semelhante possível as preferências de todos os membros. As recomendações têm também em conta as distâncias entre os locais e as respetivas horas de abertura. As preferências dos grupos são calculadas por um processo de utiliza métodos como Agregação, Interseção e Interseção Incremental para combinar as preferências individuais dos membros constituintes [56].

O MABLRF consiste numa *framework* de recomendação de POI baseada num MAS. Possui a capacidade de manter os interesses a curto e longo prazo do utilizador atualizados graças ao trabalho de seis agentes com responsabilidades distintas que, em conjunto, utilizam as alterações de comportamento do utilizador nas suas redes sociais que foram obtidas pela *framework*, para atualizar o seu perfil sempre que necessário. O MABLRF utiliza uma técnica de filtragem de recomendações colaborativa multinível que, em conjunto com os interesses do utilizador, gera recomendações mais otimizadas [57].

O ITRS é uma aplicação web que utiliza um MAS para garantir comunicação em tempo real e disponibilizar recomendações com base numa técnica de filtragem híbrida composta por abordagens colaborativas e baseadas no conteúdo. Utiliza informação obtida em tempo-real de vários setores do turismo para efetuar recomendações de excursões a um dado utilizador [58].

O PersonalTour é um MAS que efetua recomendações de serviços relacionados com viagens (voos, estadia e atrações) com base nas preferências dos utilizadores. Os agentes utilizam o feedback prévio de utilizadores para aumentarem o seu "grau de confiança" relativamente a voos, estadias ou atrações e comunicam entre si para gerar uma recomendação final com base no conhecimento dos agentes envolvidos [59].

2.3 Tecnologias existentes

A presente secção aborda as tecnologias atuais relevantes na área em estudo do projeto em desenvolvimento. Independentemente de as tecnologias terem sido ou não impostas pela organização, devido à integração num protótipo de GRS que já se encontrava em desenvolvimento, adotou-se uma postura crítica e efetuou-se uma comparação com outras tecnologias alternativas. Ou seja, os microserviços do protótipo de GRS a aprimorar estavam desenvolvidos em C# e .NET Core, ficando assim decidido que seriam a respetiva linguagem e ambiente de desenvolvimento a utilizar durante o projeto. A secção termina com uma comparação entre tecnologias alternativas existentes para entender quais são as suas principais semelhanças ou diferenças, com uma descrição do *object-relational mapper* (ORM) e sistema de base de dados utilizado pelos microserviços do protótipo de GRS.

2.3.1 C#, .NET, .NET Core e ASP.NET Core

C# é uma linguagem de programação, desenvolvida pela Microsoft, *type-safe,* moderna e orientada a objetos. O surgimento de novas práticas de design de software fez com que a linguagem acompanhasse o movimento de inovação através da integração de novas características. Desta forma o C# mantem-se atualizado e apto a criar aplicações robustas e de grande durabilidade [33].

Esta linguagem não é distribuída como um produto *standalone,* mas sim como parte da plataforma .NET Framework da Microsoft. A plataforma .NET é o ambiente construído para o desenvolvimento e execução de programas implementados em C# ou noutras linguagens de programação compatíveis, tais como VisualBasic ou C++ [34].

O .NET Core abriu horizontes ao desenvolvimento de diferentes tipos de aplicações. É uma plataforma de programação grátis e cross-platform (ao contrário do .NET) [35].

ASP.NET Core é uma *framework web open-source* criada pela Microsoft para o desenvolvimento de aplicações e serviços web através da extensão da plataforma .NET, com recurso a novas ferramentas e bibliotecas [36]. A ASP.NET Core suporta a criação de serviços RESTful (Web APIs) utilizando a linguagem C#. O tratamento dos pedidos HTTP por parte das Web APIs efetua-se com recurso a *controllers.* Os *controllers* numa Web API são classes que derivam da classe *ControllerBase,* responsável por disponibilizar propriedades e métodos que são utilizados no processo de resposta aos vários pedidos HTTP recebidos [37].

2.3.2 JavaScript, Node.js e Express.js

JavaScript (JS) é uma linguagem de programação dinâmica utilizada quer em *client-side*, como em *server-side* (apesar de ser mais comum em *server-side*). Os programas desenvolvidos em JS não necessitam de compilação prévia antes da sua execução. É uma linguagem *prototype-based*, com suporte para paradigmas de programação orientada a objetos [38, 39].

Node.js é uma plataforma de software que foi desenvolvida no Chrome's V8 JavaScript *runtime* para auxiliar na criação de aplicações de rede escaláveis. Utiliza um modelo *Input/Output* (I/O) que é *event-driven* e *non-blocking*, tornando-a leve e eficiente, perfeita para *data-intensive real-time applications* [40].

Express.js é uma *framework* flexível de aplicações web para Node.js que disponibiliza um conjunto robusto de recursos (p.e., RESTful API) que facilitam o desenvolvimento de *web* ou *mobile applications* [41].

2.3.3 Java, Spring e Spring Boot

Java é uma linguagem orientada a objetos cujas aplicações necessitam de ser interpretadas antes de serem executadas. Independentemente do sistema operativo ou do *hardware* a utilizar, as máquinas virtuais Java (JVM) encarregam-se de manter a compatibilidade nos programas desenvolvidos nesta linguagem. Características como a sua portabilidade, segurança e facilidade de aprender tornaram-na numa linguagem popular entre os programadores [42].

Spring é uma *framework open-source* que auxilia a programação em Java através da disponibilização de diversos módulos que podem ser utilizados em separado, impactando positivamente a performance das aplicações. Os módulos *spring-core* e *spring-beans* correspondem aos módulos mais importantes, uma vez que fornecem funcionalidades relacionadas com Dependency Injection (DI) e Inversion-of-Control (IoC) [43].

O grande número de configurações XML necessárias para o correto funcionamento dos programas desenvolvidos com recurso ao Spring, inclusive aplicações de pequena dimensão, levou ao aparecimento de uma espécie de componente intermediário chamado Spring Boot. O Spring Boot encarrega-se das configurações que normalmente seriam tarefa do programador, melhorando e acelerando a sua experiência de desenvolvimento com Spring. Para além disso, força mecanismos que auxiliam a integração com tecnologias externas [43, 44].

2.3.4 Análise comparativa

Apesar dos microserviços a melhorar se encontrarem desenvolvidos em C# e .NET Core, é importante perceber de que forma outras tecnologias, apresentadas nas secções 2.3.2 e 2.3.3, poderiam desempenhar o mesmo papel e quais seriam as suas diferenças ou parecenças.

Todas as tecnologias em questão são *open-source,* e suportam múltiplas plataformas (o .NET Core trouxe a compatibilidade com Linux e macOS) mas a nível de segurança o .NET Core e o Spring Boot estão melhor preparados comparativamente ao Node.js que possui uma quantidade reduzida de packages para lidar com este problema. A integração dos microserviços com as suas bases de dados possui um grande suporte em qualquer uma das tecnologias, contudo, o Node.js não é compatível com um ORM ao nível dos que o Spring e o .NET Core oferecem (p.e. Hibernate e Entity Framework). Todas as plataformas possuem uma comunidade ativa, algo que acelera o percurso até à solução de um problema, contudo, a comunidade do Node.js é maior. Quanto à performance das tecnologias, existem estudos que defendem o melhor desempenho do Node.js comparativamente ao ASP .NET a nível da utilização de memória, utilização de CPU, rapidez e percentagem de erros. Também há estudos que defendem que em ambientes Windows o ASP .NET Core apresenta um melhor desempenho que o Spring Boot e até mesmo o Node.js [45, 46, 47, 48].

É possível verificar que as tecnologias não apresentam grandes diferenças entre si e que, caso fosse necessário migrar os microserviços a melhorar, o Spring Boot e o Node.js seriam boas alternativas a considerar.

O MAS que foi necessário melhorar em diversas vertentes neste projeto foi desenvolvido com recurso à *framework* ActressMAS, estabelecendo-se assim como a tecnologia a utilizar no MAMS. À semelhança do que foi feito na secção 2.3.4 para os microserviços, pretendeu-se perceber de que forma outras tecnologias utilizadas no desenvolvimento de MASs seriam viáveis para substituir o ActressMAS.

2.3.5 ActressMAS

O ActressMAS é uma *framework open-source* utilizada no desenvolvimento de MAS que visa reduzir todas as complexidades inerentes às configurações e à aprendizagem das linguagens utilizadas pelos agentes que variam de tecnologia para tecnologia. Para além disso, a *framework* foi desenvolvida para ser compatível com .NET devido à escassez de *frameworks* compatíveis com este ambiente de desenvolvimento. A simplicidade característica do ActressMAS facilita a aprendizagem de conceitos de MAS para iniciantes [49].

Esta *framework* possibilita a comunicação sequencial e paralela entre os agentes. A comunicação estabelece-se através de troca de mensagens direta entre os agentes, identificando o nome único do agente destinatário, ou através da propagação das mensagens para todos os agentes presentes no ambiente em que atuam. As mensagens trocadas pelos agentes possuem campos que permitem identificar o destinatário da mensagem, quem a enviou, o seu conteúdo e o respetivo id da conversa. O ambiente em que os agentes atuam também disponibiliza memória partilhada de modo a reduzir o *overhead* que possa surgir do processo de comunicação por mensagem [49].

2.3.6 JADE e JADE-LEAP

O JADE (Java Agent Development Framework) é uma das *frameworks open-source* utilizadas no desenvolvimento de MAS mais populares, onde os agentes são desenvolvidos em Java. Os agentes do JADE (entidades de software *single-threaded*) interagem entre si de forma concorrente e distribuída através de mensagens Agent Communication Language (ACL) que identificam os agentes. Este processo de comunicação é transparente e independente dos *hosts* da rede onde os agentes executam. O ambiente onde os agentes atuam é constituído por plataformas e contentores onde cada contentor corre num *host* da rede e, como as plataformas são constituídas por pelo menos um contentor, acabam por ser distribuídas por *hosts* distintos. Cada plataforma é gerida por um contentor principal e, no âmbito de integrarem uma dada plataforma, os contentores periféricos registam-se em conjunto com a plataforma no contentor principal [50].

A escolha do Java como única linguagem de programação apta a desenvolver agentes JADE retira valor à framework. O JADE-LEAP é uma versão modificada do JADE que não possui modificações relevantes para além da possibilidade de desenvolver os agentes JADE noutras tecnologias como é o caso de C# e .NET [51].

2.3.7 Jason

O Jason é uma framework *open-source* que possui um interpretador Java AgentSpeak para o desenvolvimento de MAS. Esta framework segue uma arquitetura BDI (Beliefs-Desires-Intentions). Os *Beliefs* constituem informação que os agentes assumem como verdadeira que pode ser obtida através da análise do ambiente em que atuam, ou através da iteração com outros agentes via troca de mensagens. Os *Desires* representam a motivação que um agente possui para realizar uma determinada tarefa e as *Intentions* são as ações que o agente se propôs a realizar. Aliando a arquitetura BDI à linguagem orientada ao desenvolvimento de agentes AgentSpeak, possibilita que o agente desenvolva um sistema de raciocínio em tempo real para a realização de tarefas complexas através do *Procedural Reasoning System*. Com base no conhecimento adquirido através da abordagem dos agentes Jason, eventos podem desencadear a execução de um conjunto sequencial de ações que por norma estão persistidos numa biblioteca especifica [52, 53].

2.3.8 Análise Comparativa das frameworks

O MAMS a melhorar encontra-se desenvolvido em ActressMAS, contudo, é importante compreender como o JADE-LEAP e o Jason se posicionam relativamente à framework escolhida. A Tabela 4 efetua uma análise comparativa das características das três tecnologias.

Tabela 4 – Análise Comparativa entre ActressMAS, Jason e JADE-LEAP [49, 50, 54, 55]

	JADE-LEAP	Jason	ActressMAS

Possui Integração com .NET	✓	✗	✓
É Open-Source	✓	✓	✓
Disponibiliza Interface Gráfica	✓	✗	✓
Oferece Mecanismos de Segurança	✓	✗	✗*
Ausência de Complexidade Inerente a Especificações FIPA	✗	✓	✓**

* Não existe qualquer informação disponível relativamente aos mecanismos de segurança que o ActressMAS oferece;

** A estrutura simples das mensagens é baseada em conceitos FIPA ACL (Agent Communication Language proposed by the Foundation for Intelligent Physical Agents) mas não utiliza as suas especificações.

Através da análise da Tabela 7 é possível verificar que todas as tecnologias são *open-source*. Esta característica é fundamental face o caracter de investigação do projeto. Outro aspeto a ter em conta é falta de compatibilidade da *framework* Jason com .NET visto que o MAMS se encontra desenvolvido com base nesta tecnologia. O JADE-LEAP destaca-se pela disponibilização de mecanismos de segurança. A segurança é uma característica importante a considerar num futuro em que se pretenda escalar o protótipo do GRS, contudo, a utilização de especificações FIPA, apesar de oferecer interoperabilidade e definir *standards* a utilizar no MAS, faz com que a compreensão inicial destes conceitos possa ser complexa [50].

É possível verificar que o ActressMAS acaba por ser uma *framework* indicada ao MAMS a melhorar pela sua simplicidade e compatibilidade com .NET, no entanto, percebe-se que, apesar do ActressMAS ser a melhor opção, o JADE-LEAP pode ser uma alternativa interessante face aos mecanismos vantajosos que oferece. Por sua vez, caso o microserviço em que o MAS se integra estivesse desenvolvido em Java, o Jason também poderia ser uma alternativa a considerar para o desenvolvimento do MAS do protótipo do GRS.

2.3.9 Entity Framework Core

O Entity Framework Core é um ORM que utiliza objetos desenvolvidos em .NET para manipular uma base de dados. Esta tecnologia proporciona um maior nível de abstração relativamente à gestão de dados, nomeadamente durante a criação e manutenção de aplicações *data-oriented* [69].

Existem três tipos de abordagens nesta ORM: *Database-First, Code-First* e *Model-First*. Assim como o nome o sugere, a abordagem *Database-First* gera o contexto e as classes de domínio com base numa base de dados pré-existente. Já na abordagem *Code-First*, que por sua vez é a abordagem utilizada nos microserviços do protótipo do GRS, as classes de domínio e o contexto codificam-se inicialmente para que seja gerada uma base de dados com base nessas classes. Por fim, na abordagem *Model-First,* a classe de contexto, as classes de domínio e o

script da base de dados são gerados através de classes de domínio, relações e hierarquias de herança desenhadas no *visual designer* integrado no Visual Studio [69].

A Figura 2 apresenta o normal posicionamento da Entity Framework dentro de uma aplicação.

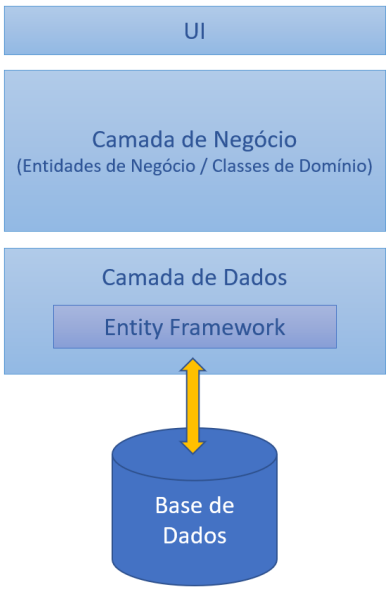

Figura 2 – Posicionamento da Entity Framework dentro de uma aplicação, adaptado de [69]

2.3.10 PostgreSQL e ElephantSQL

PostgreSQL é um sistema de base de dados *open source*, *object-relational* e *cross-platform* que se encontra em constante desenvolvimento há mais de 35 anos. Por sua vez, o ElephantSQL é um *hosting service* de PostgreSQL que se encarrega de tarefas administrativas, desde instalações e atualizações a tratamento de *backups* [70, 71].

3 Análise de valor

Este capítulo abarca o estudo do produto relativamente à análise de valor, onde se identificam e analisam as oportunidades de negócio. A análise de valor procura valorizar um determinado produto ou serviço da forma mais eficiente possível, minimizando custos sem prejudicar a qualidade [60].

3.1 Processo de Inovação

O processo de inovação é constituído por três fases que podem ser descritas como FuzzyFront End (FFE), New Product Development (NPD) e Comercialização [61]. Este mesmo processo é representado através da Figura 3 onde se verifica que o processo tem um início caótico, caracteristicamente muito experimental, e acaba por adotar uma natureza mais precisa e disciplinada ao longo do tempo.

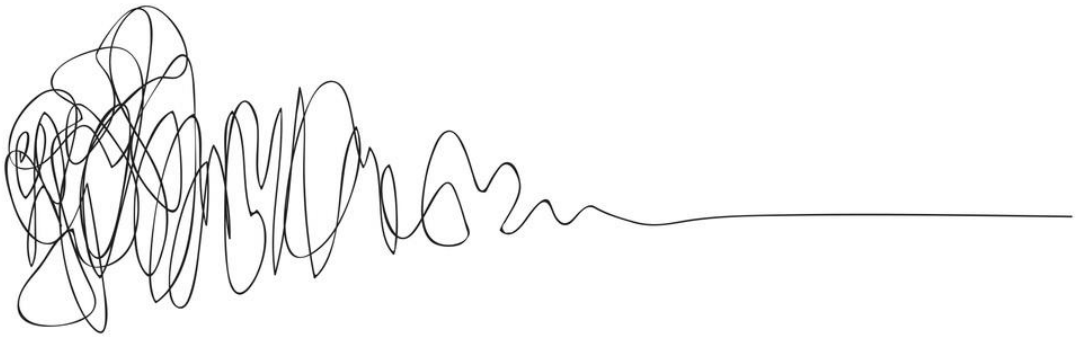

Figura 3 - The Processo of Design Squiggle, adaptado de [62]

O FFE especifica os primeiros passos no desenvolvimento de novos produtos ou serviços. Articula um aglomerado de ideias e conceitos de forma a proporcionar um mecanismo de abstração e experimentação para definir o que pode ser desenhado. Este é um processo crucial e decisivo no sucesso do produto ou serviço, face à sua importância na determinação dos passos seguintes. A escassez de planeamento faz com que os limites entre este processo e os próximos sejam confusos, sem uma definição concreta [63].

Para lidar com a falta de estandardização do FFE, criou-se o modelo New Concept Development (NCD) que fornece uma terminologia comum para o FFE. A Figura 4 apresenta as três secções principais do NCD [61]:

- *Engine* – Representa os atributos que catalisam o desenvolvimento como é o caso da estratégia e cultura da organização;
- As cinco áreas internas – Correspondem aos cinco elementos controláveis: Identificação de Oportunidade, Análise de Oportunidade, Geração e Enriquecimento de Ideias, Seleção de Ideias e Definição de Conceito;
- Fatores Externos – Corresponde aos fatores não controláveis que influenciam as cinco áreas internas e o *Engine* do NCD tal como regulações, leis ou clientes.

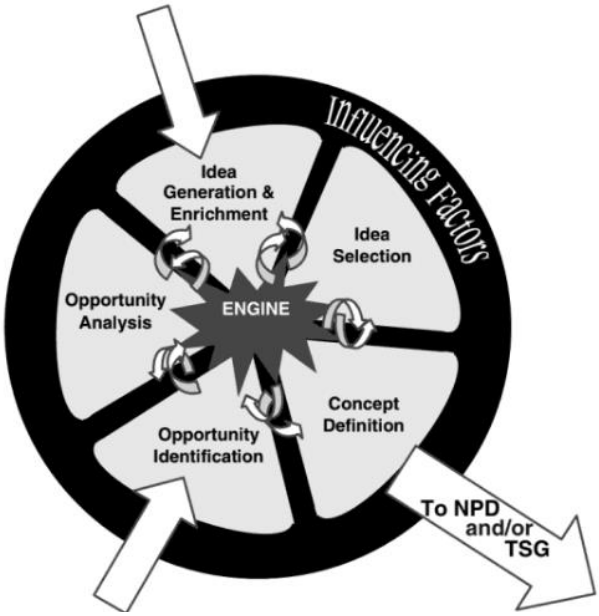

Figura 4 – Modelo NCD, adaptado de [61]

3.1.1 Identificação de Oportunidade

A Identificação de Oportunidade permite dar a conhecer à organização as oportunidades relevantes que pode seguir. Conforme o que foi detalhado na secção 1.2, é possível verificar que existe uma escassez de GRS capazes de ultrapassar limitações comuns neste tipo de sistemas, tanto nas configurações morosas, como na excessiva intrusividade necessária para

criar o perfil dos turistas e propor recomendações de atrações a visitar numa determinada excursão/roteiro de viagem.

3.1.2 Análise de Oportunidade

Na Análise de Oportunidade, efetua-se uma análise à oportunidade identificada na fase da Identificação de Oportunidade de forma a perceber se a sua exploração é vantajosa ou não. A pandemia provocada pela COVID-19 causou uma enorme crise no setor do turismo, contudo, tem-se verificado uma recuperação global significativa do setor e espera-se quem em 2023 as chegadas de turistas internacionais já tenham recuperado os níveis pré-pandémicos. Em 2022 a Europa alcançou cerca de 80% dos seus níveis pré-pandémicos com a chegada de 585 milhões de turistas internacionais. O gráfico apresentado na Figura 5 analisa os níveis de chegadas de turistas internacionais em comparação com o ano de 2019 [64].

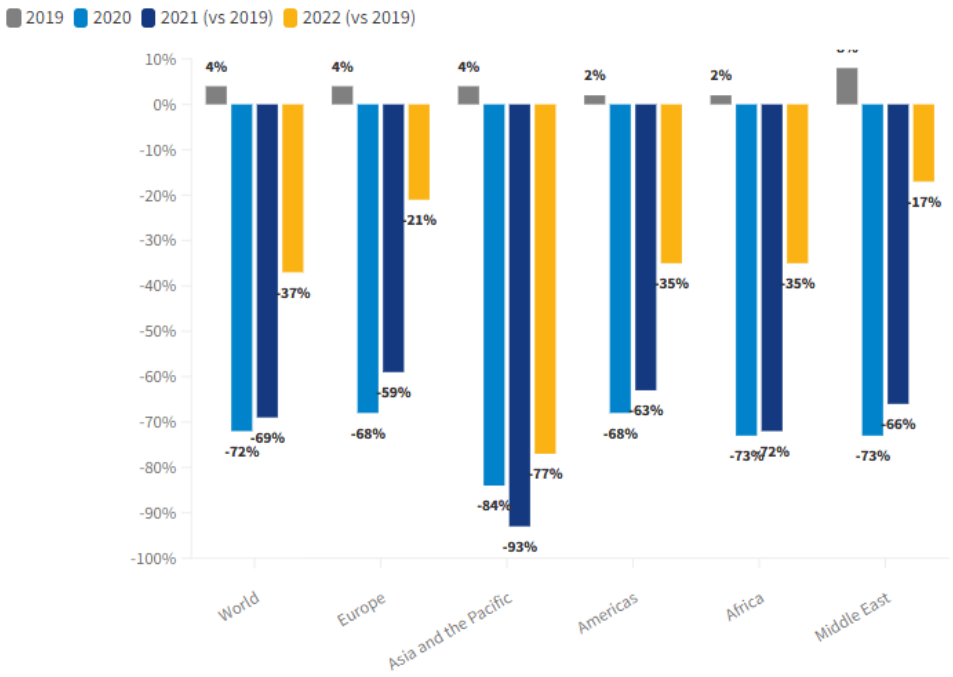

Figura 5 – Chegadas de Turistas Internacionais relativamente aos níveis de 2019, adaptado de [64]

Assim como já foi referido na secção 1.2, a utilização de MAS está a ser proposta para responder às limitações comuns dos GRS. O capítulo 2 apresenta um estudo detalhado sobre de que forma as soluções que estão a ser propostas funcionam e que tipo de vantagens e desvantagens as tecnologias utilizadas no desenvolvimento destas soluções oferecem.

3.1.3 Geração e Enriquecimento de Ideias

As ideias maturam e tornam-se concretas durante a Geração e Enriquecimento de Ideias. As ideias geradas fornecem respostas às seguintes perguntas:

- Como se consegue ultrapassar as limitações características dos GRS atuais mencionadas na secção 1.2?
- De que forma se consegue tirar proveito das similaridades entre os MAS e os microserviços e facilitar a iteração entre os agentes e o utilizador?
- De que forma a adoção de métodos de desenvolvimento ágeis para a implementação deste tipo de soluções pode ser simplificada?

As ideias geradas são:

- Adoção de uma arquitetura MAMS que utiliza uma técnica de filtragem de recomendações colaborativa;
- Adoção de uma arquitetura MAMS que utiliza uma técnica de filtragem de recomendações baseada em conteúdo;
- Adoção de uma arquitetura MAMS que utiliza uma técnica de filtragem de recomendações híbrida.

3.1.4 Seleção de Ideias

Geradas as ideias, torna-se necessário decidir qual é a mais viável para prosseguir com o processo. Escolher a técnica de filtragem de recomendações mais indicada a utilizar no MAMS do GRS em desenvolvimento é essencial uma vez que a lógica por trás das recomendações de POI aos turistas depende da técnica utilizada. O Analytic Hierarchy Process (AHP) é um método utilizado para fundamentar tomadas de decisão complexas através da sintetização de vários fatores de forma hierárquica [65]. O processo divide-se em sete fases [66]:

- Construção da árvore hierárquica de decisão;
- Comparação das alternativas e dos critérios;
- Prioridade relativa a cada critério;
- Avaliação da consistência das prioridades relativas;
- Construção da matriz de comparação para cada critério;
- Cálculo da prioridade composta para as alternativas;
- Escolha da alternativa.

As alternativas selecionadas das técnicas de filtragem de recomendações a utilizar no MAMS são:

- Filtragem colaborativa;
- Filtragem baseada em conteúdo;
- Filtragem híbrida.

As alternativas foram comparadas entre si com base nos seguintes critérios:

- Escalabilidade de utilizadores;
- Escalabilidade de POI;
- Complexidade.

A Figura 6 apresenta a árvore hierárquica de decisão resultante.

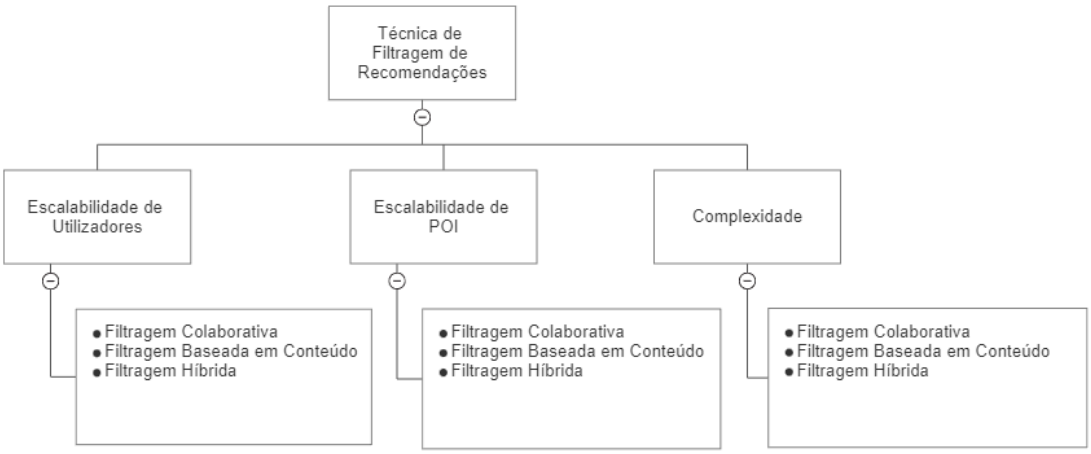

Figura 6 – Árvore Hierárquica de Decisão

O próximo passo consiste em definir prioridades para os elementos de cada nível da hierarquia recorrendo a uma matriz de comparação. As prioridades seguem a escala proposta por Saaty representada na Figura 7 e com base nos três critérios da base hierárquica, desenvolveu-se a matriz de comparação representada na Tabela 5.

Nivel de importância	Definição	Explicação
1	Igual importânica	As duas atividades contribuem igualmente para o objetivo
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Figura 7 – Escala fundamental de Saaty, adaptado de [66]

Tabela 5 – Matriz de comparação de critérios

	Escalabilidade de Utilizadores	Escalabilidade de POI	Complexidade
Escalabilidade de Utilizadores	1	5	3
Escalabilidade de POI	1/5	1	1/3
Complexidade	1/3	3	1

A matriz de comparação de critérios é normalizada de modo a igualar todos os critérios numa unidade única para obter a prioridade relativa de cada critério fazendo a média das linhas da matriz. Para normalizar, divide-se cada valor da matriz pelo valor da soma da sua respetiva coluna. A matriz de comparação de critérios normalizada é representada na Tabela 6.

Tabela 6 – Matriz de comparação de critérios normalizada

	Escalabilidade de Utilizadores	Escalabilidade de POI	Complexidade	Prioridade Relativa
Escalabilidade de Utilizadores	15/23	5/9	9/13	0,63
Escalabilidade de POI	3/23	1/9	1/13	0,11
Complexidade	5/23	1/3	3/13	0,26

A próxima etapa consiste em calcular a Razão de Consistência (RC) para medir a consistência dos julgamentos relativamente a grandes amostras de juízos aleatórios. Se o RC for superior a 0,1 significa que os julgamentos não possuem grande fiabilidade uma vez que se encontram demasiado perto para o conforto de aleatoriedade. Para se calcular o RC necessita-se de obter o valor de λ_{max} que representa o maior valor próprio da matriz de comparação e n que representa a ordem da matriz. Desta forma é possível obter o Índice de Consistência (IC) através da seguinte fórmula:

$$IC = \frac{\lambda \max - \mathrm{n}}{\mathrm{n} - 1} \quad (1)$$

O IR é um índice aleatório calculado para matrizes quadradas de ordem n pelo Laboratório Nacional de Oak Ridge, nos Estados Unidos. A Tabela 7 apresenta os valores de IR estipulados para cada quantidade de critérios.

Tabela 7 – Valores de IR para matrizes quadradas de ordem n

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
IR	0,00	0,00	0,58	0,90	1,12	1,24	1,32	1,41	1,45	1,49	1,51	1,48	1,56	1,57	1,59

É possível verificar que o valor de IR a utilizar é 0,58. Pode-se assim prosseguir ao cálculo do CR através da seguinte fórmula:

$$\mathrm{RC} = \frac{\mathrm{IC}}{\mathrm{IR}} \Leftrightarrow \mathrm{RC} = \frac{\frac{3{,}04-3}{3-1}}{0{,}58} \Leftrightarrow \mathrm{RC} = 0{,}034 \quad (2)$$

Uma vez que $\mathrm{RC} = 0{,}034$ é menor que 0,1, pode-se concluir que os julgamentos são consistentes e é possível prosseguir para a criação de matrizes de comparação por critério.

A Tabela 8 e 9 apresentam respetivamente as matrizes de comparação relativamente ao critério de escalabilidade de utilizadores antes e depois da normalização.

Tabela 8 – Matriz de comparação com base na escalabilidade de utilizadores

Escalabilidade de Utilizadores	Filtragem Colaborativa	Filtragem Baseada em Conteúdo	Filtragem Híbrida
Filtragem Colaborativa	1	7	3
Filtragem Baseada em Conteúdo	1/7	1	1/5
Filtragem Híbrida	1/3	5	1

Tabela 9 - Matriz de comparação normalizada com base na escalabilidade de utilizadores

Escalabilidade de Utilizadores	Filtragem Colaborativa	Filtragem Baseada em Conteúdo	Filtragem Híbrida	Prioridade
Filtragem Colaborativa	21/31	7/13	5/7	0,64
Filtragem Baseada em Conteúdo	3/31	1/13	1/21	0,07
Filtragem Híbrida	7/31	5/13	5/21	0,28

A Tabela 10 e 11 apresentam respetivamente as matrizes de comparação relativamente ao critério de escalabilidade de POI antes e depois da normalização.

Tabela 10 - Matriz de comparação com base na escalabilidade de POI

Escalabilidade de POI	Filtragem Colaborativa	Filtragem Baseada em Conteúdo	Filtragem Híbrida
Filtragem Colaborativa	1	1/5	1/3
Filtragem Baseada em Conteúdo	5	1	2
Filtragem Híbrida	3	1/2	1

Tabela 11 - Matriz de comparação normalizada com base na escalabilidade de POI

Escalabilidade de POI	Filtragem Colaborativa	Filtragem Baseada em Conteúdo	Filtragem Híbrida	Prioridade
Filtragem Colaborativa	1/9	2/17	1/10	0,11
Filtragem Baseada em Conteúdo	5/9	10/17	3/5	0,58
Filtragem Híbrida	1/3	5/17	3/10	0,31

A Tabela 12 e 13 apresentam respetivamente as matrizes de comparação relativamente ao critério de complexidade antes e depois da normalização.

Tabela 12 - Matriz de comparação com base na complexidade

Complexidade	Filtragem Colaborativa	Filtragem Baseada em Conteúdo	Filtragem Híbrida
Filtragem Colaborativa	1	4	9
Filtragem Baseada em Conteúdo	1/4	1	7
Filtragem Híbrida	1/9	1/7	1

Tabela 13 - Matriz de comparação normalizada com base na complexidade

Complexidade	Filtragem	Filtragem	Filtragem	Prioridade

	Colaborativa	Baseada em Conteúdo	Híbrida	
Filtragem Colaborativa	36/49	7/9	9/17	0,68
Filtragem Baseada em Conteúdo	9/49	7/36	7/17	0,26
Filtragem Híbrida	4/49	1/36	1/17	0,06

Por fim é possível perceber qual é a melhor alternativa através da multiplicação da matriz de prioridade relativa de cada critério com a matriz das prioridades de cada alternativa:

$$\begin{bmatrix} 0{,}64 & 0{,}11 & 0{,}68 \\ 0{,}07 & 0{,}58 & 0{,}26 \\ 0{,}28 & 0{,}31 & 0{,}06 \end{bmatrix} \begin{bmatrix} 0{,}63 \\ 0{,}11 \\ 0{,}26 \end{bmatrix} = \begin{bmatrix} 0{,}59 \\ 0{,}18 \\ 0{,}23 \end{bmatrix} \quad (3)$$

De acordo com os critérios definidos, os resultados permitem concluir que a primeira linha da matriz, correspondente à filtragem colaborativa, deve ser a técnica de filtragem de recomendações a utilizar no MAMS visto que pontuou mais que as outras alternativas (0,59).

3.2 Valor

O valor para o cliente pode-se resumir à relação entre os benefício e sacrifícios que um produto ou serviço oferece. O valor percecionado corresponde à avaliação geral de um cliente relativamente à utilidade de um dado produto com base nas suas convicções sobre o que é dado e recebido [67].

De forma a avaliar o valor para o cliente criou-se a Tabela 14 que apresenta os benefícios e sacrifícios associados ao projeto.

Tabela 14 - Benefícios e sacrifícios do projeto

	Produto	**Relacionamento**
Benefícios	• Utilização continua de nova informação existente no sistema para gerar recomendações de POI mais personalizadas e rápidas; • Facilita a interação entre utilizador e agentes; • Facilita a adoção de metodologias ágeis de desenvolvimento;	• Compreensão da razão das recomendações atribuídas.

	• Escalabilidade.	
Sacrifícios	• Custos associados à implantação do GRS; • Custos associados à implementação do GRS; • Custos associados à alteração do *workflow* existente.	

3.3 Proposta de Valor

A proposta de valor consiste numa visão geral do aglomerado de produtos e serviços de uma organização que possuem valor para o cliente. A Figura 8 apresenta o modelo de Osterwalder composto por duas partes, o perfil do cliente e a proposta de valor da organização. As secções deste modelo podem ser descritas da seguinte forma [68]:

- *Gains* – Descreve os benefícios desejados pelo cliente;
- *Pains* – Descreve os riscos e obstáculos relativos aos principais objetivos do cliente;
- *Customer Jobs* – Descreve os problemas que os clientes querem ver resolvidos;
- *Gain Creators* – Descreve como a solução a ser implementada pela organização adiciona valor para os clientes;
- *Pain Relievers* – Descreve como os produtos da organização alivam os *pains* dos clientes;
- *Products and Services* – Descreve os produtos e serviços que a organização oferece para responder aos pontos anteriores.

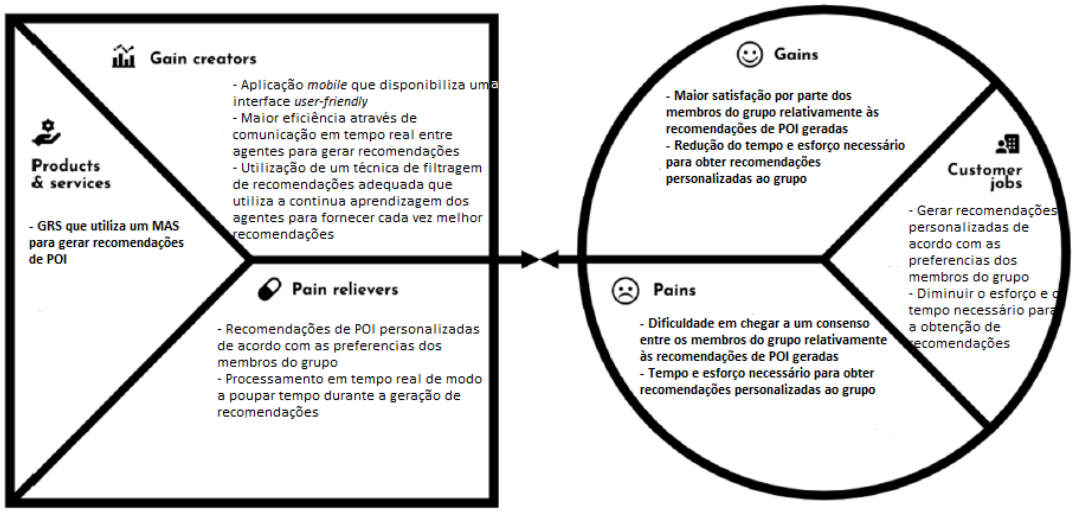

Figura 8 – Proposta de valor segundo o modelo de Osterwalder

4 Análise e desenho da solução

Este capítulo aborda o processo de compreensão do problema, onde é feita uma análise ao domínio do sistema a desenvolver. Segue-se a secção da Engenharia de Requisitos que alude aos requisitos funcionais e não funcionais da solução. Por último, apresenta-se o design estipulado, partindo de uma perspetiva geral da arquitetura global da solução que é aprofundada ao pormenor.

4.1 Domínio do problema

Várias reuniões com a coorientadora do GECAD permitiram compreender os principais conceitos de negócio necessários à concretização dos objetivos propostos. Estes conceitos encontram-se representados na Figura 9 através de um modelo de domínio do MAMS que foi melhorado.

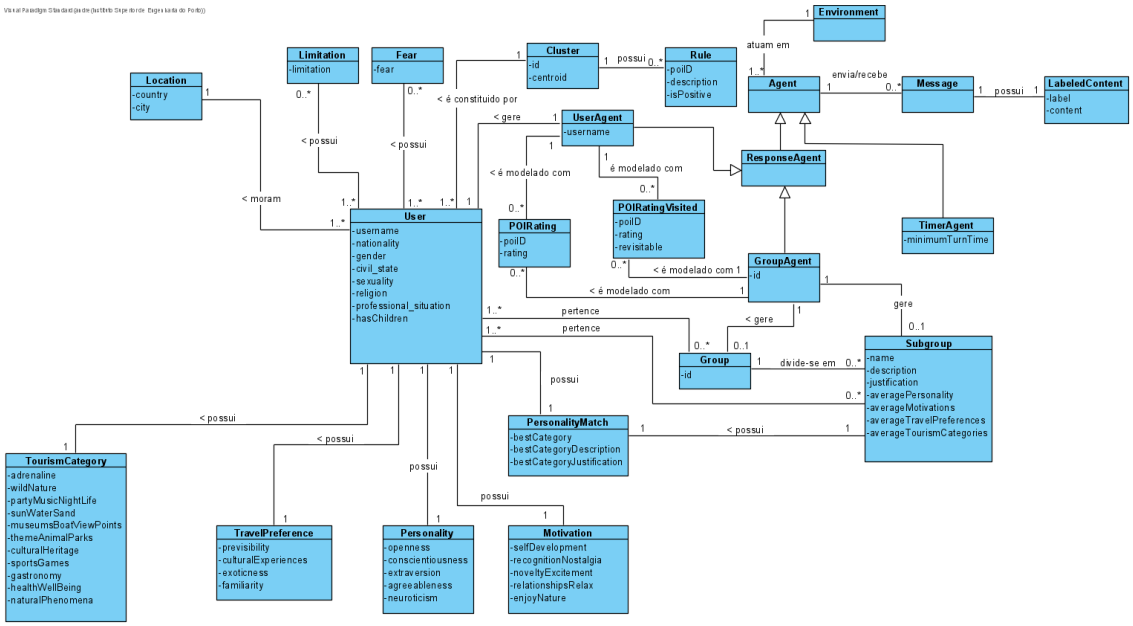

Figura 9 – Modelo de Domínio do MAMS

Um **User** representa um utilizador (turista) da aplicação que é modelado no MAMS com todos os atributos necessários para o correto funcionamento dos pedidos de recomendação de POI. Um **User** está sempre associado a um **Cluster** com base na sua **Personality**. Este processo de associação é explicado na secção 5.1.2.

A **Location** representa o país e a cidade em que o utilizador reside e as **Limitations** e **Fears** representam respetivamente as limitações e medos do utilizador que são tidas em conta na geração das recomendações de POI.

Os restantes atributos dos **Users** contribuem para a criação de **Rules** nos **Clusters** que são posteriormente usadas para tornar os pedidos de recomendação de POI mais personalizados a par das **Motivations**, das **TravelPreferences** e das **TourismCategories** que representam respetivamente as motivações, as preferências de viagem e as categorias de turismo preferidas do utilizador.

O **Group** representa um grupo de excursão de **Users** que podem ser distribuídos por **Subgroups** de modo a agrupar os utilizadores com base na sua personalidade. Este processo é explicado em pormenor na secção 5.1.4. O **PersonalityMatch** fornece informação relativa às características dos **Users** ou **Subgroups**. Esta informação é obtida com base na média dos valores da **Personality**, **Motivations**, **TravelPreferences** e **TourismCategories** dos membros do **Subgroup** em questão.

A *framework* utilizada no desenvolvimento do MAMS trouxe consigo alguns conceitos de domínio, nomeadamente o **Environment** que representa o ambiente onde os **Agents** atuam. O **ResponseAgent** representa agentes capazes de comunicar para fora do seu **Environment**. A comunicação interna e externa dos agentes é feita através de **Messages** (mensagens) que utilizam um padrão estabelecido através de **LabeledContents**. O **UserAgent** e o **GroupAgent** são dois tipos de **ResponseAgents** que se encarregam respetivamente pela gestão de informação e pelas ações a executar por parte do **User** e do **Group** (ou **Subgroup**). Quando um **UserAgent** ou um **GroupAgent** é adicionado ao **Environment**, são modelados com **POIRatings** e **POIRatingsVisited** que corresponde respetivamente ao feedback (*rating* de 1 a 5 estrelas) existente acerca de POI não visitados e POI já visitados. Mais informação sobre a obtenção e utilização destes ratings está disponível nas secções 5.1.3, 5.1.4 e 5.1.5.

A Tabela 15 corresponde ao glossário do domínio do problema.

Tabela 15 – Glossário do domínio do problema

Termo	Descrição
Agent	Entidade responsável pela gestão de informação e execução de tarefas que lhe são atribuídas.
Cluster	Aglomerado de utilizadores que possuem personalidades semelhantes em pelo menos 80%, usando a distância Euclidiana normalizada.
Environment	Ambiente onde os Agents atuam.
Fear	Medo/fobia associada a utilizadores.
Group	Grupo de excursão de utilizadores.
GroupAgent	Tipo de ResponseAgent que é responsável pela gestão de informação e execução de tarefas de um Group ou Subgroup.
LabeledContent	Rótulo associado a Messages durante o

	processo de comunicação.
Limitation	Limitação física associada a utilizadores (e.g, surdez, cegueira, limitação motora, problemas cardíacos).
Location	Residência associada a utilizadores.
Message	Mensagem utilizada no processo de comunicação interna e externa dos Agents.
Motivation	Informação relativa às motivações que levam um dado User a viajar.
Personality	Personalidade associada a um utilizador de acordo com 5 dimensões do modelo Big Five (Abertura à experiência, Conscienciosidade, Extraversão, Agradabilidade, Neuroticismo).
PersonalityMatch	Informação relativa às características de um dado User ou Subgroup. Por exemplo, a melhor categoria de turismo para um dado utilizador com base nos seus dados.
POIRating	*Feedback* relativo a um POI que ainda não foi visitado por um dado User.
POIRatingVisited	*Feedback* relativo a um POI que já foi visitado por um dado User.
ResponseAgent	Tipo de Agent que é capaz de comunicar para fora do seu Environment.
Rule	Indicação de um Cluster para priorizar ou dificultar a recomendação de um determinado POI. As regras são geradas com base no *feedback* fornecido pelos utilizadores de um dado Cluster e pelas suas respetivas características pessoais.
Subgroup	Subgrupo de um determinado Group que agrupou Users com uma Personalidade semelhante.
TimerAgent	Agent adicionado ao Environment no momento da sua criação para prevenir que este termine a sua execução.
TourismCategory	Informação relativa às categorias de turismo preferenciais de um dado User.
TravelPreference	Informação relativa às preferências de viagem

	de um dado User.
User	Utilizador da aplicação.
UserAgent	Tipo de ResponseAgent que é responsável pela gestão de informação e execução de tarefas de um User.

4.2 Engenharia de Requisitos

Durante a análise detalhada do problema, efetuada com o intuito de aferir o propósito do sistema, documentam-se os resultados da observação em diversos formatos de representação e valida-se o conhecimento adquirido. Este processo de obtenção de requisitos tem o nome de Engenharia de Requisitos e é a linha de partida para se prosseguir às restantes etapas do processo de desenvolvimento de software. Não se deve menosprezar a importância da Engenharia de Requisitos até porque os problemas relacionados com este processo são a principal causa do fracasso de projetos de software [72, 73].

4.2.1 Requisitos funcionais

O comportamento do sistema que se pretende reproduzir através de algum tipo de funcionalidade pode se definir como requisito funcional [74]. Identificaram-se seis funcionalidades distintas após analisar o que se pretendia para a aplicação (Figura 10).

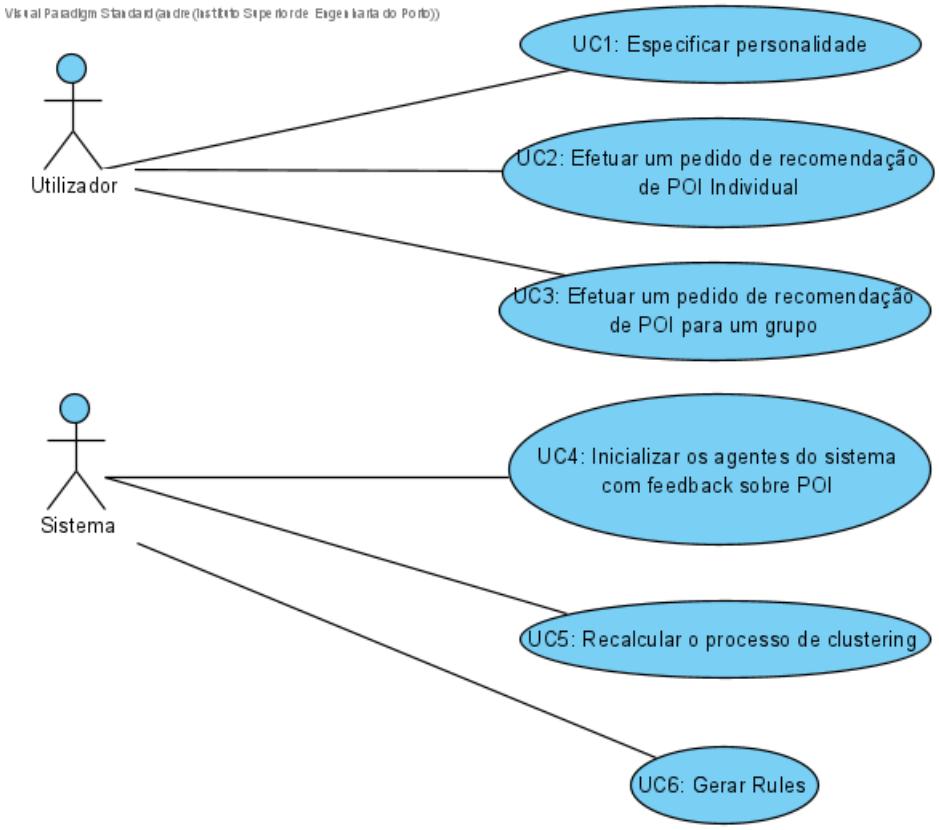

Figura 10 – Diagrama de casos de uso

É possível distinguir dois tipos de atores na aplicação. O Sistema, cujos casos de uso associados são automatizados, isto é, não necessitam da intervenção por parte de um utilizador, e o Utilizador, cujos casos de uso necessitam dessa intervenção e ao mesmo tempo não carecem de distinção entre tipos de utilizadores.

Pode-se obter mais detalhes sobre cada caso de uso na Tabela 16.

Tabela 16 – Descrição dos casos de uso

Casos de uso	Descrição
UC1 – Especificar personalidade	O Utilizador deve preencher um questionário sobre a sua personalidade. Esta ação desencadeia a atualização da personalidade do UserAgent que lhe corresponde no MAMS e associa-o a um Cluster.
UC2 – Efetuar um pedido de recomendação de POI individual	O Utilizador deve efetuar um pedido de recomendação de POI individual e obter uma lista de POI personalizada de acordo com as suas características.
UC3 – Efetuar um pedido de recomendação de POI para um grupo	O Utilizador (*group owner*) deve efetuar um pedido recomendação de POI para um grupo

	e obter uma lista de POI personalizada de acordo com as características do grupo. É possível que os membros do grupo inicial sejam distribuídos por subgrupos com base no grau de similaridade entre as suas personalidades.
UC4 – Inicializar os agentes do sistema com feedback sobre POI	O Sistema deve adicionar os UserAgents que gerem os utilizadores ao Enviroment do MAMS com os POIRatings, POIRatingsVisited e Rules que lhes correspondem devidamente modelados.
UC5 – Recalcular o processo de *clustering*	O Sistema deve redistribuir os utilizadores do MAMS em novos clusters de 24 em 24 horas caso essa necessidade se verifique.
UC6 – Gerar Rules	O Sistema deve gerar Rules através de um algoritmo *apriori* com base numa análise efetuada ao feedback e ao perfil dos membros de cada Cluster

4.2.2 Requisitos não funcionais

Enquanto os requisitos funcionais procuram especificar o que é que o sistema deve fazer, os requisitos não funcionais procuram aferir como é que o sistema o deve fazer, geralmente conhecidos como requisitos de qualidade ou restrições. É importante salientar que os requisitos não funcionais são tão importantes quantos os funcionais e devem ser especificados com o mesmo nível de detalhe. Estes requisitos especificam normas que o produto deve apresentar [75, 76]. Para categorizar os requisitos não funcionais recorreu-se ao modelo de classificação de características qualitativas de software FURPS+ *(Functionality, Usability, Reliability, Performance, Supportability and more)* [77].

O "+" alude a outras categorias utilizadas na classificação dos requisitos, nomeadamente os de design, que especificam ou restringem o design do sistema, os de implementação, que especificam ou restringem o desenvolvimento de um sistema de software, os físicos, que especificam restrições físicas exigidas pelo hardware utilizado na implantação do sistema, ou os de interface, que especificam ou restringem funcionalidades imanentes às interfaces de diferentes componentes [77].

Funcionalidade (*Functionality*)

- Já foram especificados na secção 4.2.1.

Usabilidade (*Usability*)

- Não se aplica (n/a).

Confiabilidade (*Reliability*)

- n/a.

Desempenho (*Performance*)

- Os pedidos de recomendação de POI devem obter resposta em menos de 10 segundos.

Suportabilidade (*Supportability*)

- O sistema deve estar apto a receber vários pedidos de recomendação de POI em simultâneo;
- Os microserviços do sistema devem disponibilizar uma REST API.

Restrições de design

- n/a.

Restrições de implementação

- O MAMS deve ser desenvolvido em C# e ASP.NET Core;
- O modelo de dados a utilizar deve ser gerado através da Entity Framework Core;

Restrições físicas

- Os microserviços devem estar hospedados na plataforma Microsoft Azure;
- A base de dados utilizada por cada microserviço deve estar hospedada em ElephantSQL.

Restrições de interface

- n/a.

4.3 Desenho

A junção de dois modelos de representação arquitetural (C4 e 4+1) permitiu especificar de uma forma geral, a arquitetura do protótipo do GRS.

O design e implementação da estrutura de software são responsabilidades da arquitetura de software. Algumas arquiteturas podem atingir grandes proporções e uma má documentação do sistema pode dificultar a obtenção de informação por parte dos *stakeholders*. O modelo de Vistas 4+1 procura minimizar situações semelhantes através da descrição do sistema em cinco vistas principais [78]:

- Vista lógica – Especifica os serviços que o sistema fornece aos utilizadores;
- Vista de processos – Detalha os fluxos de processos existentes no sistema;
- Vista de implementação – Demonstra como o software se organiza estaticamente no seu ambiente de desenvolvimento;
- Vista física – Representa a distribuição do software em hardware;
- Vista de cenários – Associa processos de negócio a atores que os possam iniciar.

A vista de cenários não é apresentada juntamente com as restantes uma vez que já foi abordada na secção 4.2.

Segundo o modelo C4, a descrição do software divide-se em quatro níveis distintos de abstração e à medida que o nível aumenta, menor é o grau de granularidade, isto é, detalha o sistema em "pedaços" cada vez mais reduzidos. Os níveis de abstração são os seguintes [79]:

- Nível 1 – Enquadra o sistema a nível das pessoas que o utilizam e dos sistemas de software com que interage;
- Nível 2 – Efetua uma ampliação ao sistema e mostra os seus contentores independentes;
- Nível 3 – Detalha os componentes dos contentores do sistema;
- Nível 4 – Descreve elementos de código utilizados na implementação de componentes.

Os dois modelos coexistem através da apresentação das vistas apropriadas em cada nível para que se compreenda a solução.

4.3.1 Nível 1 - Contexto

A Figura 11 recorre a um diagrama de componentes para demonstrar o contexto em que o sistema se insere.

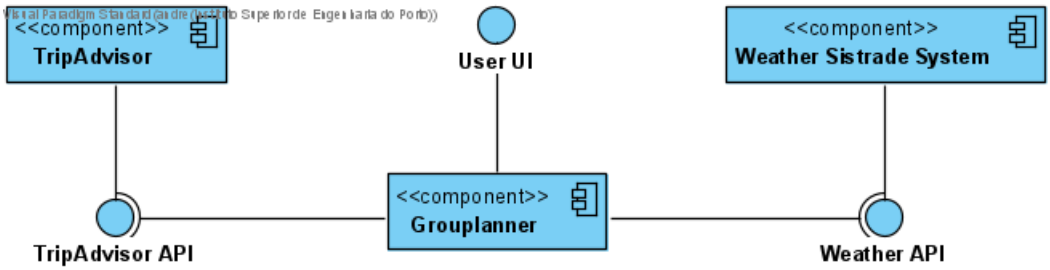

Figura 11 – Diagrama de Contexto

A componente Grouplanner representa o protótipo do GRS em desenvolvimento. Este protótipo disponibiliza uma interface gráfica mobile (em Android) para os utilizadores da aplicação e consome informação de duas REST API distintas. Uma das REST API é disponibilizada pela TripAdvisor e fornece informação relativa aos POI a recomendar. A segunda REST API pertence à Sistrade e dispõe dados relativos ao clima previsto em diversas cidades num intervalo de cinco dias.

4.3.2 Nível 2 – Contentores

Esta secção aborda as vistas do modelo de Vistas 4+1 consideradas apropriadas para o nível dois do modelo C4.

4.3.2.1 Vista lógica

O protótipo do GRS adota uma arquitetura em microserviços e fornece uma interface gráfica mobile. A API Gateway é o único ponto de entrada para os clientes dos microserviços e é responsável por rotear a informação entre os vários microserviços. A Figura 12 representa a vista lógica de nível 2 do Grouplanner.

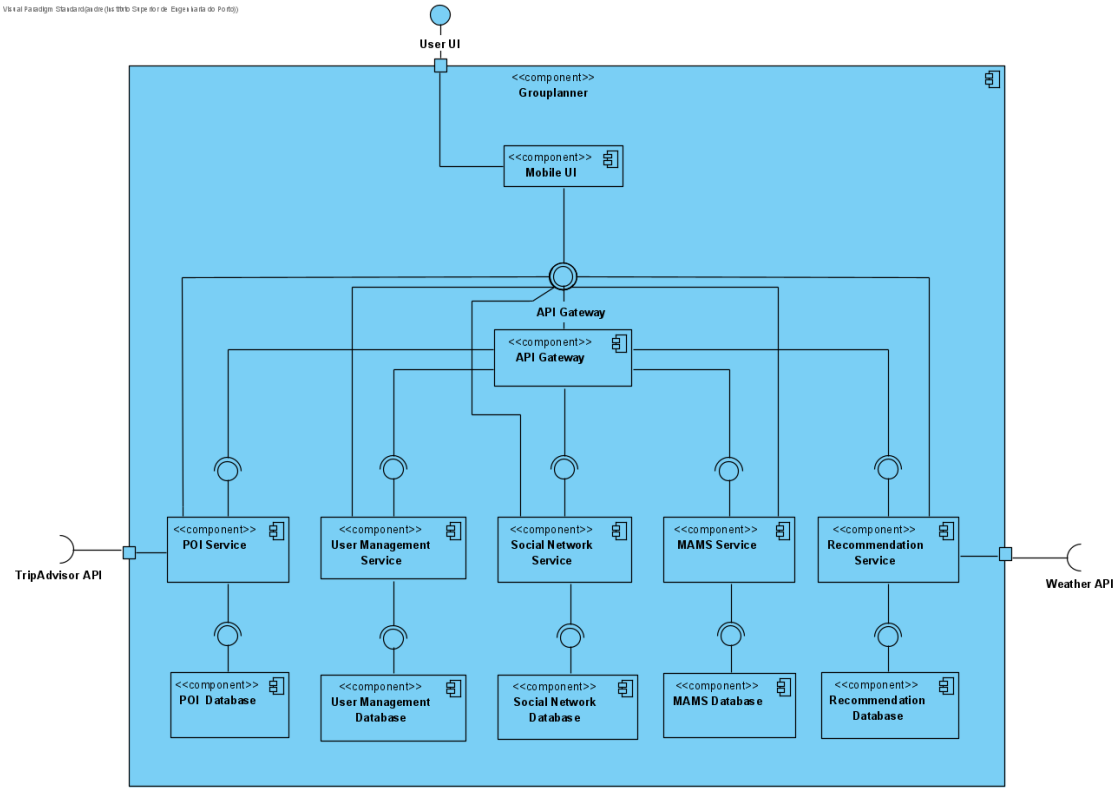

Figura 12 – Diagrama de vista lógica nível 2

4.3.2.2 Vista de processos

A Figura 13 retrata a vista de processos de nível 2 com base num diagrama de sequência que reflete o fluxo do caso de uso UC3 – Efetuar um pedido de recomendação de POI para um grupo.

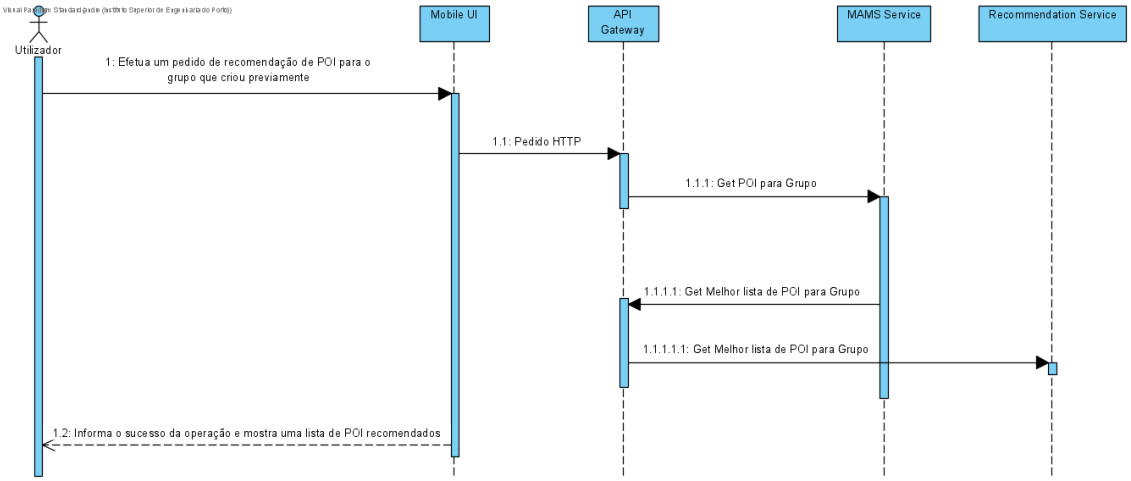

Figura 13 – Diagrama de vista de processos nível 2

4.3.2.3 Vista de implementação

A Figura 14 representa o diagrama de implementação de nível 2.

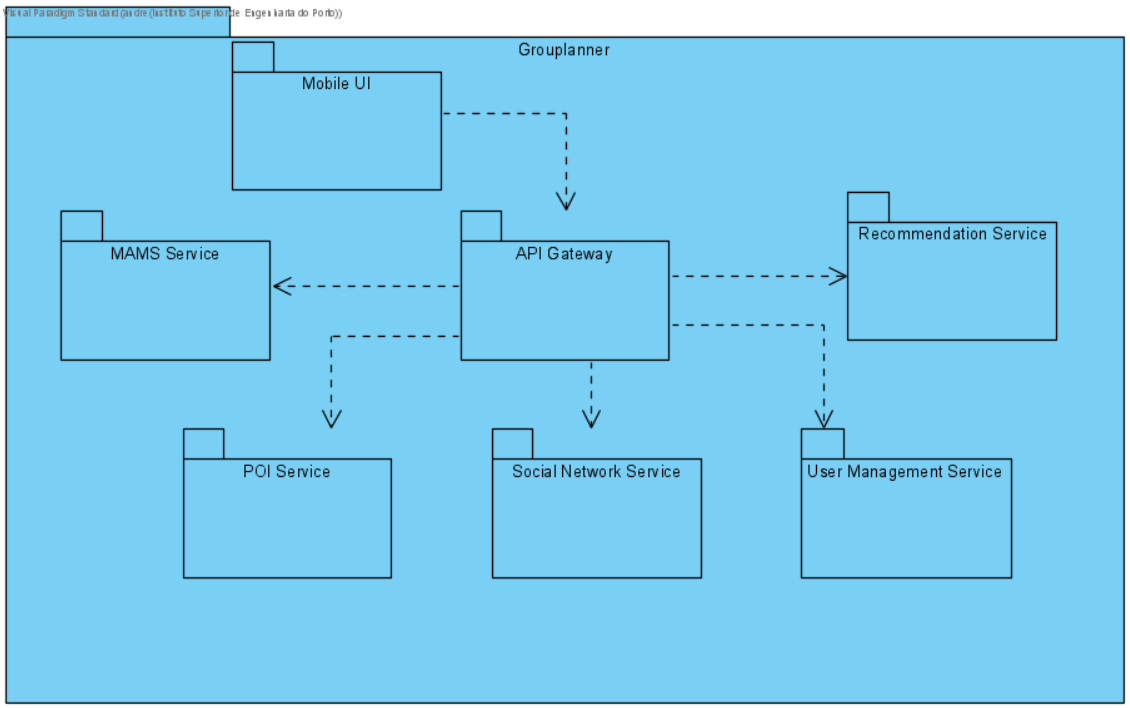

Figura 14 – Diagrama de vista de implementação nível 2

O módulo Mobile UI depende da API Gateway uma vez que manipula e apresenta dados provenientes de pedidos HTTP a utilizadores que interagem com o protótipo do GRS. Por sua vez, o módulo da API Gateway depende dos módulos dos vários microserviços do protótipo visto que roteia dados entre os microserviços através de pedidos HTTP para posteriormente serem consumidos pela Mobile UI.

Todos os módulos foram desenvolvidos com tecnologias impostas pela organização. O módulo da Mobile UI foi desenvolvido em Android e os restantes em C# e .NET.

4.3.2.4 Vista física

A Figura 15 mostra como é que os diferentes componentes interagem entre si e como é que estão hospedados. O componente da Mobile UI está normalmente inserido nos dispositivos mobile individuais de cada utilizador. Este componente comunica com a API Gateway recorrendo ao protocolo HTTPS.

A organização impos que os microserviços fossem hospedados na plataforma Microsoft Azure por questões de compatibilidade com C# e .NET. As respetivas bases de dados de cada microserviço estão hospedadas na plataforma ElephantSQL. A comunicação com as bases de dados é feita via TCP/IP.

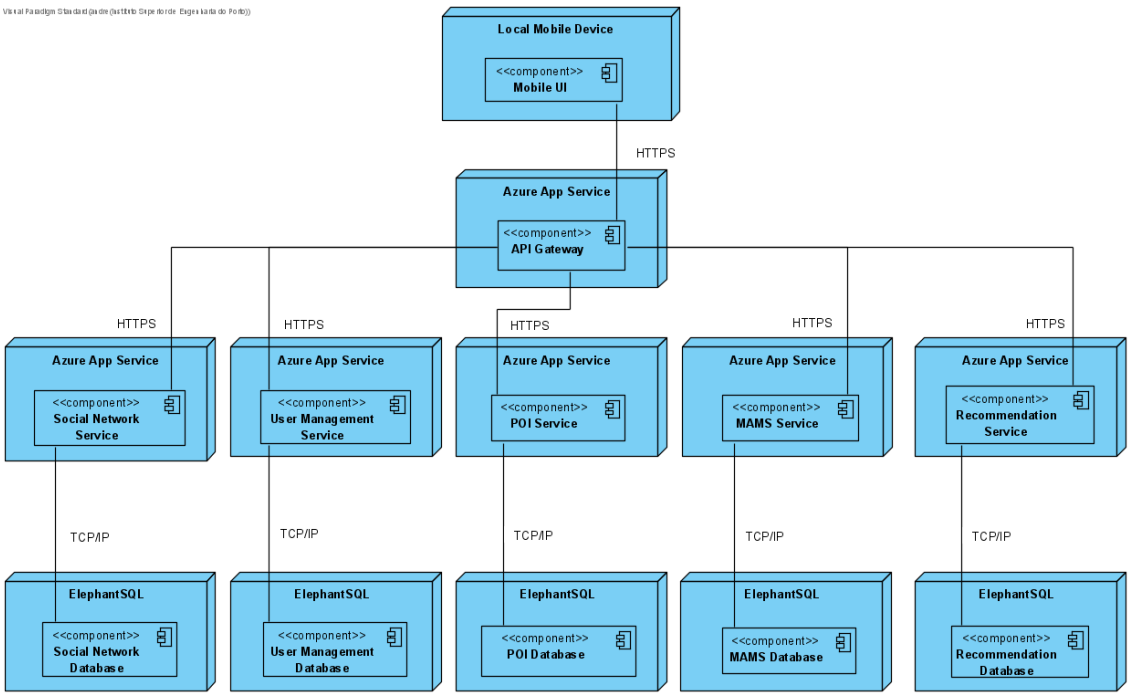

Figura 15 – Diagrama de vista física de nível 2

4.3.3 Nível 3 – Componentes

Esta secção aborda as vistas do modelo de Vistas 4+1 consideradas apropriadas para o nível três do modelo C4.

4.3.3.1 Vista lógica

A Figura 16 representa a arquitetura desenvolvida para o contentor MAMS Service.

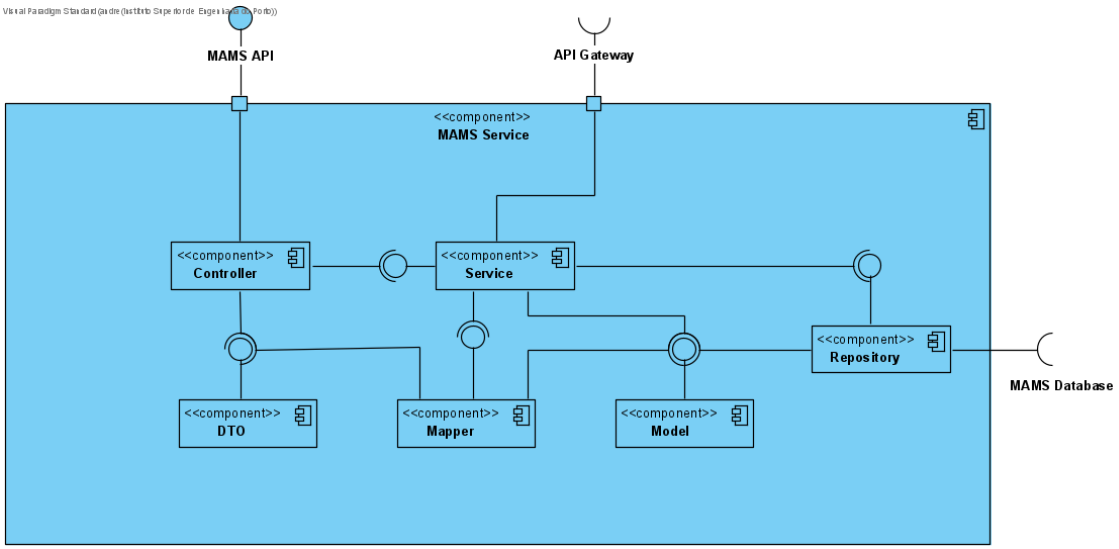

Figura 16 – Diagrama de vista lógica de nível 3 para o contentor MAMS Service

Os objetos do tipo DTO (Data Transfer Object) são utilizados na transferência de informação entre as camadas que possuem e não possuem conhecimento do negócio. A camada Controller é responsável por responder aos pedidos que recebe através da REST API que providencia. Durante este processo, os pedidos são redirecionados para os serviços apropriados. Esta camada é desprovida de qualquer conhecimento de domínio, por isso apenas utiliza objetos do tipo DTO.

As operações necessárias à realização de uma determinada funcionalidade são orientadas pela camada Service. Esta camada possui permissões de acesso à camada Repository e detém conhecimento dos objetos de domínio (Model) e das respetivas regras de negócio.

A camada Mapper disponibiliza métodos para converter objetos do tipo Model para DTO e vice-versa.

A camada Repository tem a responsabilidade de aceder à base de dados e executar todas as operações necessárias para alcançar objetivos relacionados com as regras de negócio. Por exemplo, a salvaguarda e obtenção de objetos.

4.3.3.2 Vista de processos

O caso de uso “UC3 – Efetuar um pedido de recomendação de POI para um grupo” é provavelmente o caso de uso de maior importância, posto isto, foi escolhido para explorar a vista de processos de nível 3 através de diagramas de sequência ilustrados nas Figuras 17, 18, 19, 20 e 21.

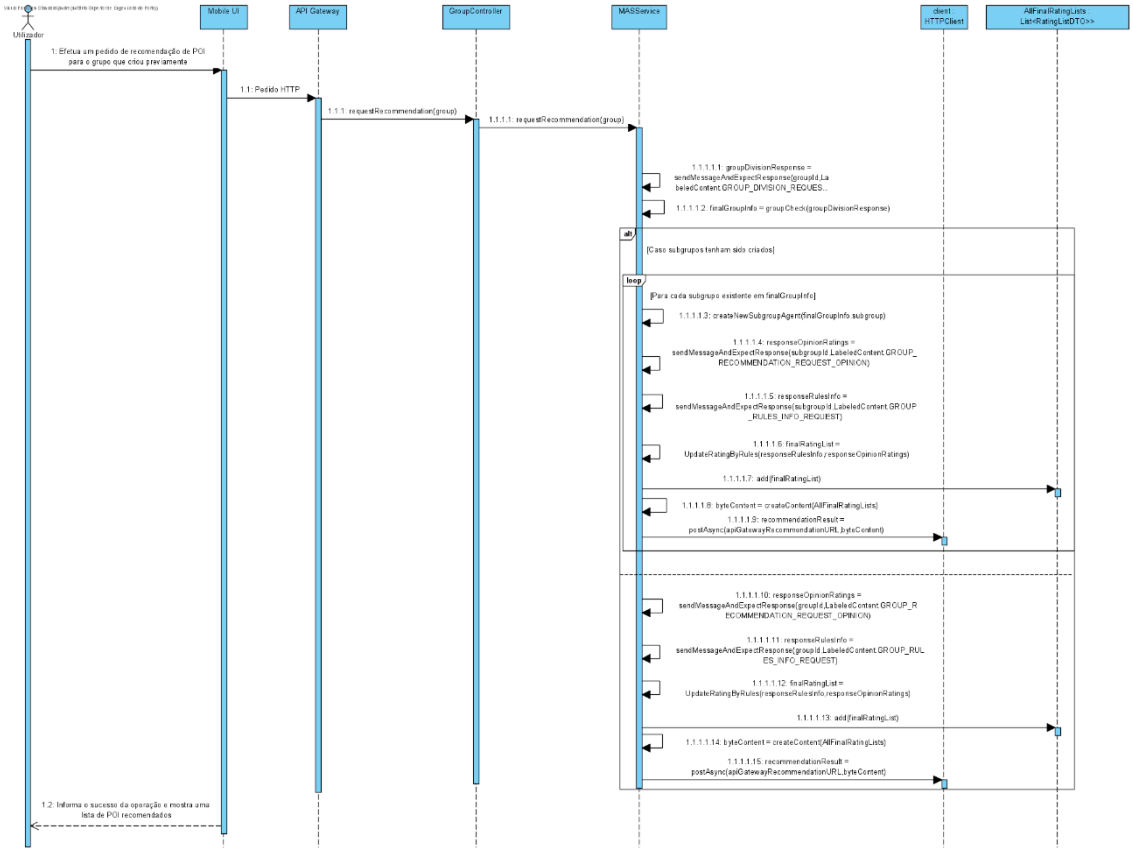

Figura 17 – Vista de processos nível 3 (UC3)

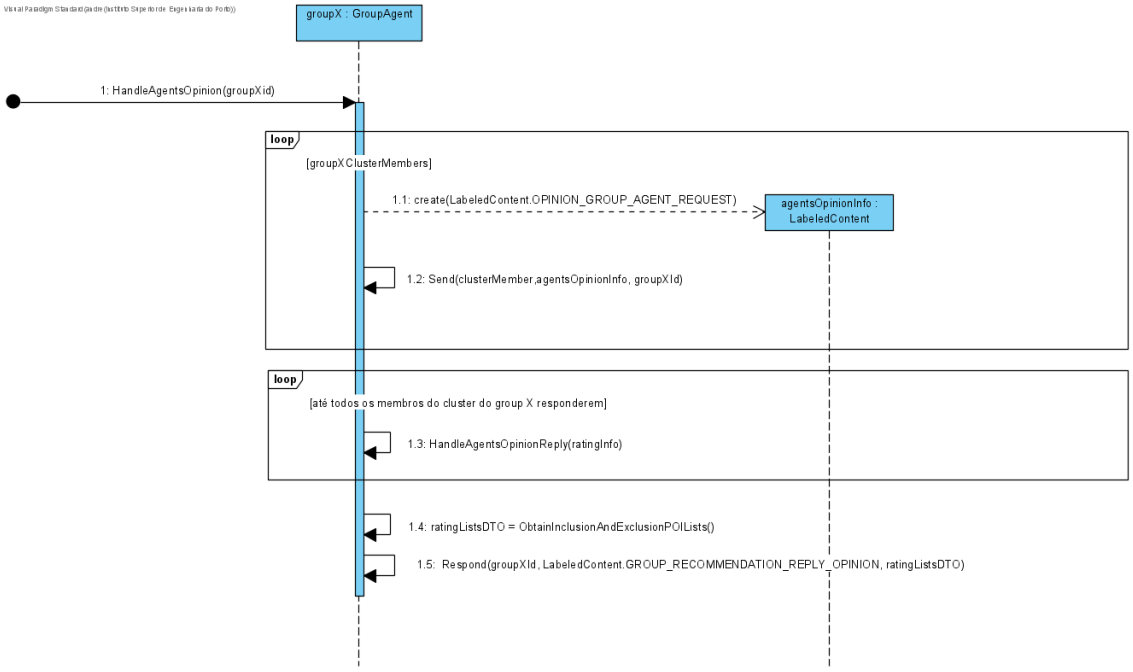

Figura 18 – Vista de processos nível 3 (UC3) – Fluxo da obtenção de ratings no GroupAgent

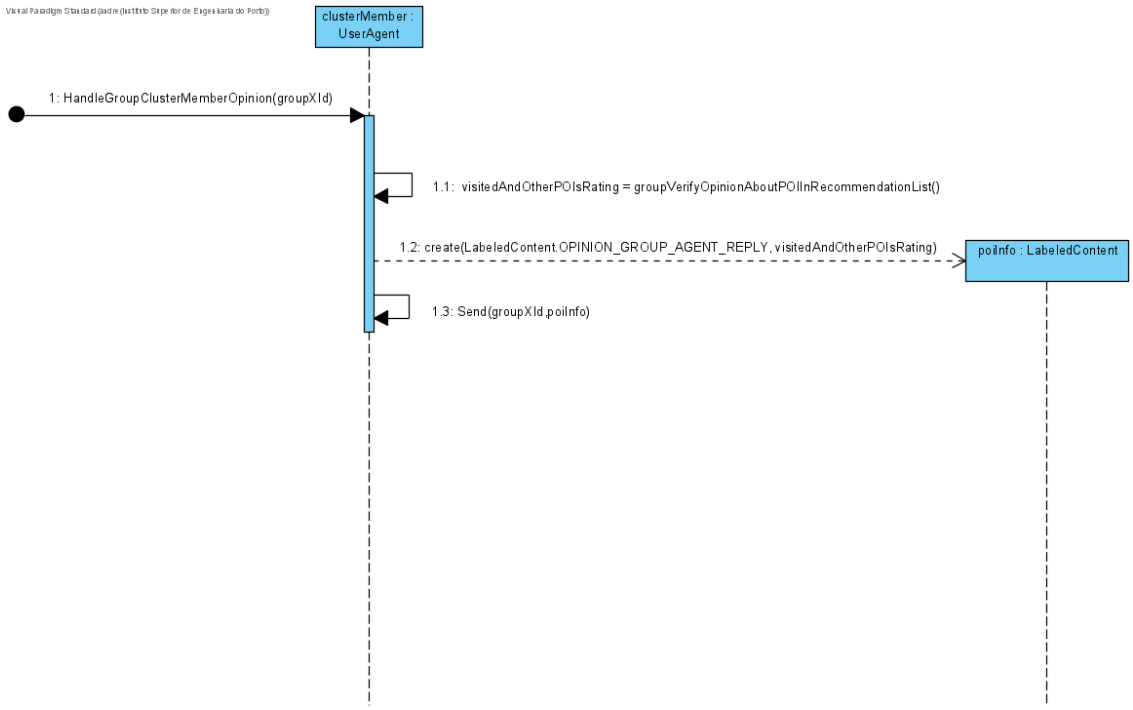

Figura 19 - Vista de processos nível 3 (UC3) – Fluxo da obtenção de ratings no UserAgent

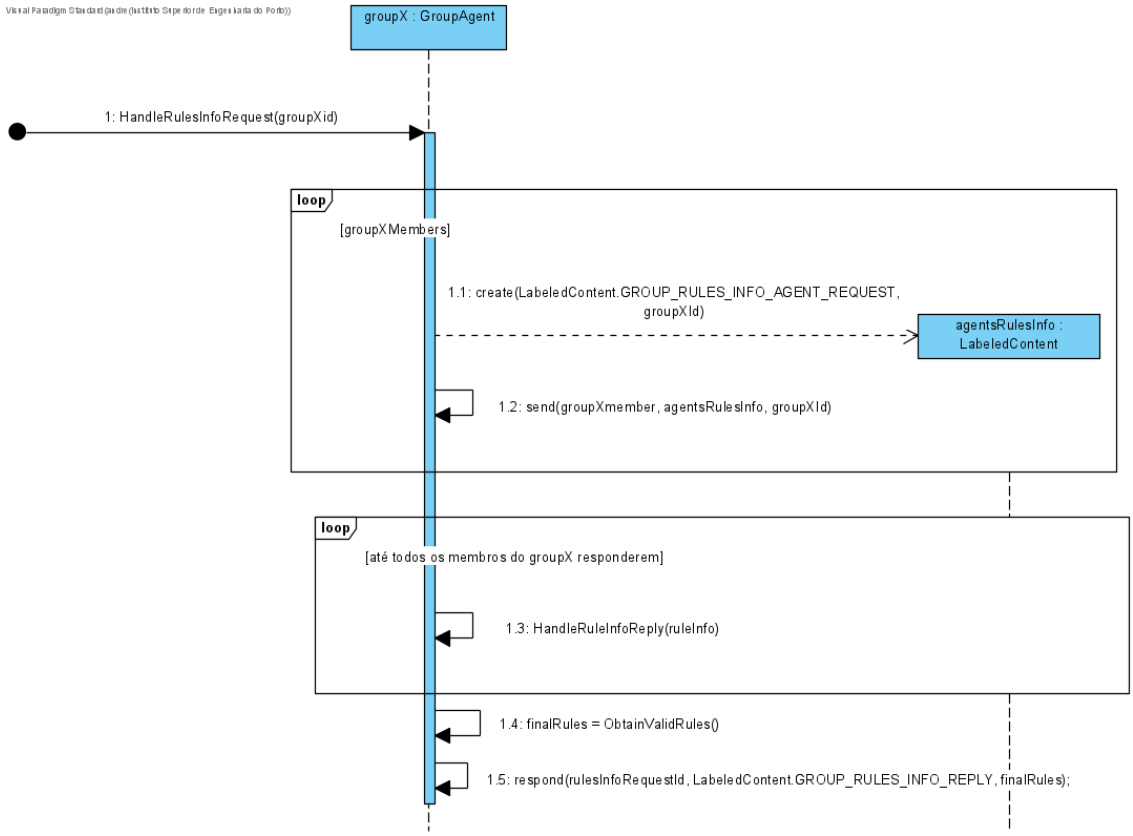

Figura 20 - Vista de processos nível 3 (UC3) – Fluxo da obtenção de rules no GroupAgent

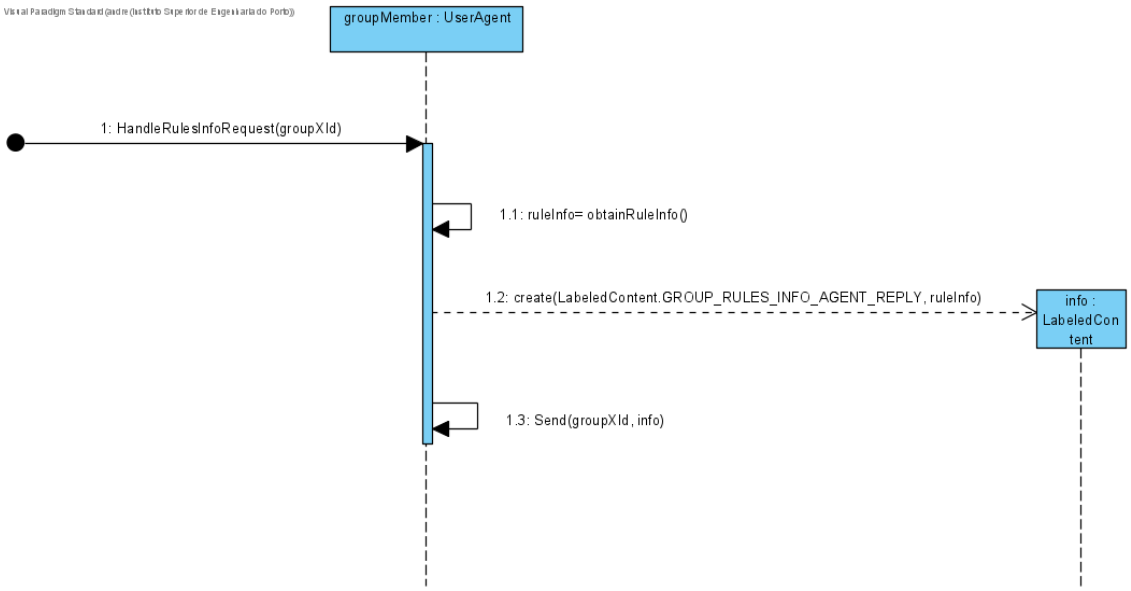

Figura 21 - Fluxo da obtenção de rules no UserAgent

Mais informação sobre a sequência de operações existentes no caso de uso UC3 é disponibilizada na secção 5.1.4.

4.3.3.3 Vista de implementação

A vista de implementação de nível 3 do módulo MAS Service está ilustrada na Figura 22, onde é possível observar as dependências entre os seus componentes.

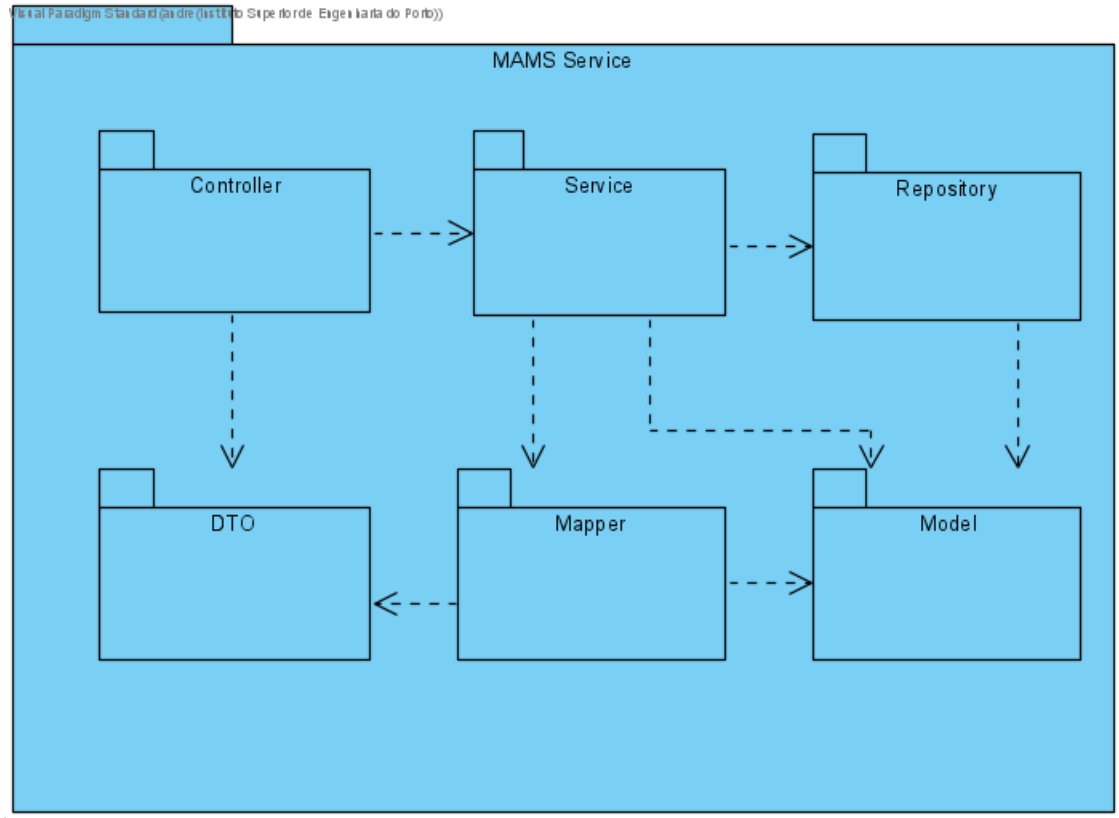

Figura 22 – Vista de implementação nível 3 do módulo MAMS Service

4.3.4 Nível 4 – Codificação

O nível onde granularidade é mais fina, ou seja, o nível de abstração é menor, procura mostrar de que forma os componentes individuais são implementados e de que forma essas classes de código interagem entre si. Estas interações são explicadas ao pormenor no capítulo 5.

4.3.5 Padrões de design adotados

Padrões de design de software procuram identificar soluções para problemas recorrentes com base na experiência de estruturas de software bem-sucedidas. É difícil definir um padrão como bom ou mau porque depende sempre do contexto em que é utilizado. No fundo, os padrões de software são uma recomendação do que precisa de ser feito face a uma determinada situação e podem e devem ser adaptados ao sistema em que vão ser utilizados [80, 81].

Padrão Model-View-Controller (MVC)

O padrão MVC procura dividir a estrutura da aplicação em três componentes distintas, cada uma com as suas próprias responsabilidades. A camada Model, que é responsável pela gestão da informação da aplicação, isto é, o domínio, recebe o input dos utilizadores através da camada Controller que coordena esses dados. A camada View está encarregue da

apresentação dos dados do Model ao utilizador num formato específico. A adoção deste padrão é vantajosa já que possibilita o baixo acoplamento, melhoria na escabilidade e a reutilização de componentes [82].

Padrão DTO

O Padrão DTO encapsula informação que é requisitada durante um dado pedido à aplicação em questão, no caso do Grouplanner, são os dados que chegam à interface mobile. A sua utilização é vantajosa porque permite disponibilizar num só pedido, informação que por norma necessitaria de vários pedidos para se obter. Possibilita ainda filtrar a informação de modo que dados que não sejam requisitados sejam enviados. O DTO desconhece lógica associada aos objetos de domínio [83].

Padrão Repository

O Padrão Repository é utilizado para criar uma abstração entre a lógica de negócio e o acesso à base de dados que no contexto deste projeto é feito através de uma ORM. Alguns dos seus benefícios são a redução do acoplamento, da redundância de código e também reduz a ocorrência de possíveis erros [84].

Padrão Service Layer

É normal que funcionalidades distintas possuam etapas ao longo do seu fluxo comuns, seja na manipulação ou no acesso aos dados. A adoção do Padrão Service Layer surge com o intuito de diminuir a duplicação de código que poderia surgir na resposta a múltiplas funcionalidades da aplicação através da centralização da lógica de negócio numa só camada [85].

Padrão API Gateway

O Padrão API Gateway oferece um único ponto de entrada para todos os clientes dos microserviços. Os clientes desconhecem desta forma que a aplicação está estruturada em microserviços, nem necessitam de localizar os microserviços com quem precisam de comunicar. Desta forma, diminui-se o número de pedidos que os clientes necessitam de fazer e melhora a experiência do utilizador [86].

Padrão Database per service

O Padrão Database per service diz que cada microserviço deve manter a sua informação privada e só a deve disponibilizar por meio da sua API. Deste Modo, os serviços são apenas responsáveis pelas transações que envolvem a sua base de dados. Existem várias possibilidades para manter a informação privada [86]:

- Private-tables-per-service – Cada serviço detém um conjunto de tabelas cujo acesso é exclusivo a esse serviço;
- Schema-per-service – Cada serviço possui um *database schema* que é privado a esse serviço;
- Database-server-per-service – Cada serviço tem o seu próprio servidor de base de dados (possibilidade escolhida).

Padrão Single Service Instance per Host

O Padrao Single Service Instance per Host recomenda hospedar cada microserviço nos seus próprios *hosts*. Deste modo, os microserviços estão isolados uns dos outros, facilitando a gestão e monitorização dos mesmos [86].

5 Implementação da solução

Pormenores sobre a implementação da solução são apresentados no presente capítulo através de uma descrição inicial acerca do funcionamento geral do MAMS e de uma posterior análise detalhada aos casos de uso implementados/melhorados. Os testes desenvolvidos são descritos no final do capítulo.

5.1 Descrição da implementação

As funcionalidades a implementar e a melhorar no MAMS do protótipo do GRS originaram alterações em outros microserviços existentes de modo que os requisitos fossem concretizados com sucesso. A descrição dos detalhes sobre as ações efetuadas nesses microserviços limita-se à informação necessária para a contextualização das funcionalidades do MAMS.

As tecnologias utilizadas ao longo do desenvolvimento das funcionalidades são descritas na secção 2.3.

5.1.1 Contextualização do MAMS

Dentro do MAMS existe um serviço responsável pela gestão e processamento dos pedidos que necessitam de interagir com o MAS cujo nome é *MASService*. Este serviço é instanciado como *Singleton* na classe *Startup* do microserviço. Desta forma, o MAS permanece ativo durante o tempo de execução do MAMS, permitindo a normal comunicação entre os agentes, e garante que o MAS é o mesmo para qualquer pedido que procure interagir com os agentes.

O construtor do *MASService* inicializa um dicionário do tipo *Dictionary,* posteriormente utilizado para receber mensagens de resposta dos agentes que habitam o ambiente do MAS que, por sua vez, também é inicializado no construtor e colocado em execução através do método *Start* de uma *thread* (*envThread*). Um *TimerAgent* é adicionado ao ambiente para prevenir que este termine a sua execução, uma vez que está programado por *default* para a terminar na inexistência de agentes (Figura 23).

```
public MASService()
{
    //Initializing the response dictionary dictionary
    agentResponseDictionary = new Dictionary<String, ConcurrentDictionary<Guid, Message>>();
    //Initializing the environment
    environment = new EnvironmentMas();
    //Adding the timer agent
    //This also ensures that the environment doesn't stop due to lacking any agents
    environment.Add(new TimerAgent(), TIMER_AGENT_NAME);
    //Creating the thread for the environment
    envThread = new Thread(new ParameterizedThreadStart(StartEnv));
    //Starting the environment thread
    envThread.Start(environment);

    if (client.BaseAddress == null)
    {
        client.BaseAddress = new Uri("https://grouplanner-rems.azurewebsites.net");
        client.DefaultRequestHeaders.Accept.Clear();
        client.DefaultRequestHeaders.Accept.Add(
            new MediaTypeWithQualityHeaderValue("application/json"));
    }
}
```

Figura 23 – Construtor do serviço *MASService*

Os métodos do *MASService* que interagem com o seu ambiente são referenciados pelos *Controllers* cujos *endpoints* permitem a receção de pedidos externos ao MAMS. A Figura 24 mostra um dos *endpoints* disponibilizado pelo *UserController* que permite a interação externa com o MAS para aferir o estado de atividade de um agente de um dado utilizador.

```
[HttpPost("{username}/testMASConnection")]
0 references
public async Task<IActionResult> TestMASConnection(string username)
{
    String resp = await masService.TestMASConnection(username);
    return Ok(resp);
}
```

Figura 24 – *Endpoint* disponibilizado pelo *UserController*

Quando um agente que precisa de interagir fora do seu ambiente, no momento da sua criação é associado a um *ConcurrentDictionary* que é simultaneamente guardado no *MASService*. Isto acontece porque nativamente a *framework ActressMAS* permite que os agentes recebam mensagem externas ao seu ambiente, mas não possibilita o contrário. De forma a conectar mensagens numa conversa, o *MASService* procura nos dicionários dos agentes por um *globally unique identifier* (GUID). A Figura 25 mostra o método utilizado pelo *MASService* na comunicação bilateral com os agentes do seu ambiente.

```
//Method that sends a message and waits for a response from the agent
11 references
private LabeledContent SendMessageAndExpectResponse(string receiver, string label, dynamic content)
{
    //Retrieve the dictionary that the agent will use to respond
    ConcurrentDictionary<Guid, Message> responseDictionary;
    if (!agentResponseDictionary.TryGetValue(receiver, out responseDictionary))
    {
        //If the dictionary can't be found, return an error message
        return new LabeledContent("failed", "failed");
    }

    //Create an unique id for the message
    Guid messageId = Guid.NewGuid();

    //Build and send the message to the agent
    SendMessage(receiver, label, content, messageId);

    //Look into the response dictionary until a response with the correct id is found
    Message response;
    while (!responseDictionary.TryRemove(messageId, out response))
    {
        //Sanity sleep to prevent CPU overusage
        Thread.Sleep(1);
    }

    //Return the response's contents
    return response.ContentObj;
}
```

Figura 25 – Método utilizado pelo *MASService* para comunicar com agentes e aguardar por respostas

Os *ResponseAgents* (Figura 26) foram criados com o intuito de responder às limitações de resposta externas nativas da classe *Agent* do ActressMAS. Enquanto o *GUID* era responsável por rotear as *Messages*, um LabeledContent, devidamente rotulado com o propósito da mensagem, era associado à mensagem para encapsular o conteúdo enviado.

```
public abstract class ResponseAgent : Agent
{
    //Dictionary used to send responses to outside the MAS
    public ConcurrentDictionary<Guid,Message> responseDictionary;

    public string logSignature;

    //Places a response in the response dictionary
    9 references
    public void Respond(String conversationId, String label, dynamic content){
        //Parse the id into Guid
        Guid responseId = Guid.Parse(conversationId);
        //Construct the labeled content response
        LabeledContent responseContent = new LabeledContent(label,content);
        //Construct the response message
        Message response = new Message(this.Name,"",responseContent);
        //Add the response to the response dictionary
        responseDictionary.TryAdd(responseId,response);
    }
```

Figura 26 – Classe *ResponseAgent*

5.1.2 Especificar Personalidade

Para que um utilizador possa efetuar pedidos de recomendação de POI, sejam eles individuais ou de grupo, necessita de preencher previamente um questionário sobre a sua personalidade com cerca de 44 perguntas. Após o preenchimento do questionário, o perfil do utilizador atualizado com os dados da personalidade é enviado do *User Management Microservice*

(UMMS) para o MAMS em formato JSON para um dos *endpoints* disponibilizados pelo MAMS via HTTP PUT.

O método *UpdateUser* do *UserController* é responsável pela gestão do *endpoint* que atualiza a personalidade dos utilizadores. No URL do *endpoint* existe uma parcela dinâmica ({username}) que armazena o *username* do utilizador cuja informação será atualizada. Esta prática é comum em vários *endpoints*. O JSON com o perfil atualizado é convertido num DTO do tipo *UserProfileMAMSDTO* e dois métodos de *Services* distintos são chamados para gerir a persistência das alterações na base de dados do MAMS e para atualizar a informação dos agentes (Figura 27).

```
[HttpPut("{username}/updateUser")]
0 references
public async Task<IActionResult> UpdateUser(string username, UserProfileMAMSDTO dto)
{
    try
    {
        //Update user in the database
        CreateUserDTO dtoWithCluster = await updateUserService.updateUserInfo(username, dto.user);
        //Update user in MAS
        await masService.UpdateUserAgent(username, dtoWithCluster);

        return Ok("User info updated");
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 27 – Método responsável por gerir o *endpoint* que atualiza a personalidade dos utilizadores do MAMS

O método *updateUserInfo* do *MASService*, responsável pela atualização da informação do utilizador na base de dados, começa por obter o objeto da classe User que referencia o utilizador na base de dados através do método *GetUserByUsername* providenciado pelo *UserRepository*. Os dados do utilizador relativos à sua personalidade são validados e atualizados em memória através do método *updateUser* e o utilizador é associado a um cluster com base nesses dados por via do método *associateClusterByPersonality*. Por fim, utiliza-se o método *UpdateObject* do *UserRepository* para persistir as alterações na base de dados e um DTO com a informação do utilizador atualizada, obtido com o auxílio do *UserMapper*, é retornado (Figura 28).

```
public async Task<CreateUserDTO> updateUserInfo(string username, CreateUserDTO dto)
{
    User user = await userRepository.GetUserByUsername(username);

    user.updateUser(dto);

    await associateClusterByPersonality(user);

    await userRepository.UpdateObject(user);

    //Return DTO already with the clusters associated because it is used in the user agents creation
    return (CreateUserDTO)map.toCreateDTO(user);
}
```

Figura 28 – Atualização a nível da base de dados da informação do utilizador no MAMS

Dentro do método *associateClusterByPersonality*, começa-se por obter a lista de todos os *Clusters* existentes na base de dados do MAMS através do método *GetAllCluster* da classe *ClusterRepository* (Figura 29).

```
List<Cluster> clusters = await this.clusterRepository.GetAllCluster();
```

Figura 29 – Obtenção dos Clusters existentes na base de dados do MAMS

Para cada *Cluster* existente na base de dados MAMS, é calculada uma média dos valores da personalidade dos utilizadores que pertencem ao *Cluster* para comparar com os valores da personalidade do utilizador que se pretende associar a um *Cluster*. A comparação é efetuada através do método *NormalizePersonalityEuclideanDistance* da classe *Distance* que utiliza a fórmula da distância euclidiana para obter a similaridade entre duas *Personalitys*. A Figura 30 e 31 mostram respetivamente a obtenção da média dos valores de personalidade dos utilizadores dos *Clusters* e a obtenção da similaridade entre a *Personality* resultante dessa média e a *Personality* do utilizador a associar a um *Cluster*.

```
if (clusters.Count() != 0)
{
    foreach (Cluster c in clusters)
    {
        averageOpenness = 0;
        averageConsc = 0;
        averageExtra = 0;
        averageAgree = 0;
        averageNeuro = 0;

        foreach (User t in c.users)
        {
            //Here we only have the total personality dimensions, not the average
            averageConsc = averageConsc + t.conscientiousness;
            averageOpenness = averageOpenness + t.openness;
            averageExtra = averageExtra + t.extraversion;
            averageAgree = averageAgree + t.agreeableness;
            averageNeuro = averageNeuro + t.neuroticism;
        }

        //Calculate Average
        averageConsc = (averageConsc / (c.users.Count()));
        averageOpenness = (averageOpenness / (c.users.Count()));
        averageExtra = (averageExtra / (c.users.Count()));
        averageAgree = (averageAgree / (c.users.Count()));
        averageNeuro = (averageNeuro / (c.users.Count()));
```

Figura 30 – Obtenção dos valores médios de personalidade dos *Users* dos *Clusters*

```
Personality averagePersonality = new Personality
{
    Openness = averageOpenness,
    Conscientiousness = averageConsc,
    Extraversion = averageExtra,
    Agreeableness = averageAgree,
    Neuroticism = averageNeuro
};

euclideanSimilarity = Distance.NormalizePersonalityEuclideanDistance(personalityUser, averagePersonality);
```

Figura 31 – Obtenção da similaridade entre duas *Personalitys*

Caso o valor da similaridade, guardado na variável *euclideanSimilarity*, seja simultaneamente superior ao valor máximo de similaridade encontrado até ao momento, guardado na variável

maxSimilarity, e maior ou igual ao valor de similaridade mínimo da variável *MIN_SIMILARITY*, que está definido em 0.8 numa escala de 0 a 1, atualiza-se o valor da variável maxSimilarity com o da euclideanSimilarity e associa-se o *Cluster* atual à variável *bestCluster* que preserva o melhor *Cluster* encontrado para o utilizador em análise (Figura 32). A variável *maxSimilarity* é inicialmente inicializada com um valor negativo para não bloquear o processo.

```
if (maxSimilarity < euclideanSimilarity && euclideanSimilarity >= MIN_SIMILARITY)
{
    maxSimilarity = euclideanSimilarity;
    bestCluster = c;
}
```

Figura 32 – Atualização da melhor Similaridade e do melhor *Cluster*

Este processo é repetido até todos os *Clusters* da base de dados do MAMS serem analisados e, no final, o melhor *Cluster*, que consta na variável *bestCluster*, é associado ao *User*. De notar que, caso nenhum *Cluster* cumpra os requisitos para ser considerado um *Cluster* adequado para o utilizador ou caso não exista nenhum *Cluster* na base de dados, um novo *Cluster* é criado e associado ao *User* (Figura 33). Esta condição verifica-se quando a variável *bestCluster* mantem o valor com a qual foi inicializada, ou seja, null.

```
if (bestCluster != null)
{
    user.cluster = bestCluster;
}
else
{
    Cluster newCluster = new Cluster
    {
        clusterId = Guid.NewGuid().ToString(),
        rules = new List<Rule>()
    };

    user.cluster = newCluster;
}
```

Figura 33 – Associação do User a um Cluster

A atualização da informação do *UserAgent* ocorre no método *UpdateUserAgent* do *MASService*. Uma vez que o agente do utilizador já se encontra inicializado no ambiente do MAS, a atualização da sua informação é feita através do método *SendMessage* que recebe como parâmetro o *username* do *User*, para identificar o *UserAgent* destinatário, um *LabeledContent*, para identificar o propósito da mensagem, e um *CreateUserDTO* com as informações atualizadas (Figura 34).

```
//Method used to update an already existing agent
3 references
public async Task UpdateUserAgent(string username, CreateUserDTO dto)
{
    SendMessage(username, LabeledContent.UPDATE_USER_REQUEST, dto);
}
```

Figura 34 – Envio da mensagem de atualização ao *UserAgent*

Quando um *UserAgent* recebe uma *Message*, o método *Act* é executado. Dentro deste método existe um *switch-case* cujas opções variam conforme os propósitos da *Message* que são especificados no *LabeledContent*. Para a etiqueta "UPDATE_USER_REQUEST", o método HandleUpdate é executado de modo a atualizar o *UserAgent* com as novas informações que constam no *CreateUserDTO* enviado na *Message* (Figura 35).

```
public override void Act(Message message)
{
    //Parse the message content into LabeledContent format
    LabeledContent receivedContent = LabeledContent.ParseMessage(message);

    switch(receivedContent.label){
        //Received an update request
        case LabeledContent.UPDATE_USER_REQUEST:
            //Convert the contents into UpdateUserDTO format
            CreateUserDTO updateDTO = (CreateUserDTO) receivedContent.content;
            //Update this agent's attributes
            HandleUpdate(updateDTO);
            break;
```

Figura 35 – Processamento do pedido de atualização do UserAgent

Após os dados atualizados serem persistidos na base de dados e o *UserAgent* ser modelado com a sua personalidade e o cluster associado, o MAMS informa o sucesso da operação.

5.1.3 Pedido de recomendação de POI individual

O método *requestRecommendation* do *UserController* é responsável pela gestão do *endpoint* que lida com os pedidos de recomendação de POI individuais. O método *RequestIndividualRecommendation* do *MASService* é chamado para processar o pedido de recomendação do utilizador cujo *username* consta no DTO do tipo *AskIndividualRecommendationsDTO* passado por parâmetro (Figura 36). Este DTO possui informação que é utilizada no processamento do pedido de recomendação.

```
[HttpPost("{username}/requestRecommendation")]
public async Task<IActionResult> requestRecommendation(AskIndividualRecommendationsDTO askIndividualRecommendationsDTO)
{
    try
    {
        //Request the recommendatiosn from the MAS
        var result = await masService.RequestIndividualRecommendation(askIndividualRecommendationsDTO);
        return Ok(result);
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 36 - Método responsável por gerir o *endpoint* que processa os pedidos de recomendação de POI individuais

O *username* é utilizado para verificar se o *UserAgent* do utilizador que efetuou o pedido existe. Feita a verificação, uma mensagem com a etiqueta

"INDIVIDUAL_RECOMMENDATION_REQUEST" é enviada ao agente através do método *SendMessageAndExpectResponse*.

```
public async Task<ResponseIndividualRecommendationsMAMSDTO> RequestIndividualRecommendation(AskIndividualRecommendationsDTO askIndividualRecommendationsDTO)
{
    //Verify if user exists
    if (TestMASConnection(askIndividualRecommendationsDTO.username).Result.Equals("failed"))
    {
        throw new NullReferenceException("User agent " + askIndividualRecommendationsDTO.username + " could not be found");
    }

    LabeledContent response = SendMessageAndExpectResponse(askIndividualRecommendationsDTO.username, LabeledContent.INDIVIDUAL_RECOMMENDATION_REQUEST);
```

Figura 37 – Verificação da existência do *UserAgent* para o envio da mensagem de etiqueta "INDIVIDUAL_RECOMMENDATION_REQUEST"

O *UserAgent* executa o método *HandleIndividualRecommendationRequest* após verificar o propósito da mensagem. O *ConversationId* da mensagem é passado por parâmetro para que mais tarde se possa enviar uma resposta ao *MASService*.

```
case LabeledContent.INDIVIDUAL_RECOMMENDATION_REQUEST:
    //Handle the individual recommendation request
    HandleIndividualRecommendationRequest(message.ConversationId);
    break;
```

Figura 38 – Processamento de uma mensagem de etiqueta "INDIVIDUAL_RECOMMENDATION_REQUEST" no *UserAgent*

Dentro do método *HandleIndividualRecommendationRequest*, um objeto to tipo *PersonalityMatch* é criado com base na informação do utilizador gerida pelo seu *UserAgent* e posteriormente enviado numa mensagem de resposta ao *MASService*.

```
private void HandleIndividualRecommendationRequest(String messageId){
    //Calculate the personality match
    PersonalityMatch personalityMatch = new PersonalityMatch(this.username,this.personality,
        this.travel_preferences,this.motivations);

    //Respond with the personality match
    Respond(messageId,LabeledContent.INDIVIDUAL_RECOMMENDATION_REPLY,personalityMatch);

}
```

Figura 39 – Resposta ao *MASService* com um *PersonalityMatch*

O *MASService* aguarda assincronamente pela resposta do *UserAgent* e, quando a obtém, envia uma mensagem ao *UserAgent* a requisitar informação sobre *ratings* dos POI.

```
//Ask for agents opinions before send any recommendation list to user
LabeledContent responseOpinion = SendMessageAndExpectResponse(askIndividualRecommendationsDTO.username,
    LabeledContent.INDIVIDUAL_RECOMMENDATION_REQUEST_OPINION);
```

Figura 40 – Envio de mensagem de etiqueta "INDIVIDUAL_RECOMMENDATION_REQUEST_OPINION" ao *UserAgent*

Quando o *UserAgent* recebe uma mensagem etiquetada com "INDIVIDUAL_RECOMMENDATION_REQUEST_OPINION", utiliza a variável booleana *opinionOngoing* para sinalizar que o agente está a processar um pedido de retribuição de *feedback* e executa o método *HandleAgentsOpinion* (Figura 41). De notar que a variável *individualRecommendationDTO* do tipo *ResponseIndividualRecommendationsMAMSDTO*, criada para responder a necessidades do processamento antigo, deve ser ignorada visto que não está a ser utilizada durante a implementação atual.

```
case LabeledContent.INDIVIDUAL_RECOMMENDATION_REQUEST_OPINION:
    //Make sure that no recommendation request opinions are already ongoing
    if (opinionOngoing)
    {
        //If opinion is ongoing, ignore this request
        break;
    }

    //Opinions are not ongoing, update the current status to reflect that one has just started
    opinionOngoing = true;

    //Convert the contents into ResponseIndividualRecommendationsMAMSDTO format
    ResponseIndividualRecommendationsMAMSDTO individualRecommendationDTO = (ResponseIndividualRecommendationsMAMSDTO)receivedContent.content;
    this.opinionRequestId = message.ConversationId;
    //Save last recommendation
    this.individualRecommendationDTO = individualRecommendationDTO;
    //Handle the individual reccomendation request opinions
    HandleAgentsOpinion(this.username,individualRecommendationDTO);
    break;
```

Figura 41 – Processamento de uma mensagem de etiqueta "INDIVIDUAL_RECOMMENDATION_REQUEST_OPINION" no *UserAgent*

O método *HandleAgentsOpinion* efetua um pedido de ratings de POI a todos os membros do cluster a que está associado, ou seja, o próprio *UserAgent* está incluído. Para isso, uma mensagem de etiqueta "OPINION_AGENT_REQUEST" é enviada através do método *Send* do *UserAgent* (Figura 42). O *id* do utilizador que efetuou o pedido de recomendação de POI é enviado no *ConversationId* da mensagem para que os vários *UserAgents* saibam a quem têm de enviar a informação.

```
private void HandleAgentsOpinion(String userXnickName,ResponseIndividualRecommendationsMAMSDTO recommendationsMAMSDTO)
{

    foreach (string member in clusterMembers)
    {
        //Construct the labeled content for the response
        LabeledContent agentsOpinionInfo = new LabeledContent(
                                            LabeledContent.OPINION_AGENT_REQUEST,
                                            recommendationsMAMSDTO);

        Send(member, agentsOpinionInfo, userXnickName);
    }
}
```

Figura 42 – Pedido de feedback a todos os membros do cluster do *UserAgent*

Cada *UserAgent* dos membros do cluster executa o método *HandleClusterMemberOpinion* ao receber as mensagens a requisitar *feedback* (Figura 43). Por sua vez, é executada uma chamada ao método *verifyOpinionAboutPOI* que cria um objeto do tipo VisitedAndOtherPOIsRatingDTO que guarda o id do utilizador que forneceu o *feedback* e as listas dos DTOs de *POIRating* e *POIRatingVisited que* lhe correspondem (Figura 44). Criado o objeto com a informação, este é enviado para o *UserAgent* cujo *username* consta na variável *messageId* numa mensagem de etiqueta "OPINION_AGENT_REPLY" (Figura 45).

```
case LabeledContent.OPINION_AGENT_REQUEST:
    //Convert the contents into ResponseIndividualRecommendationsMAMSDTO format
    ResponseIndividualRecommendationsMAMSDTO individualreq = (ResponseIndividualRecommendationsMAMSDTO)receivedContent.content;

    HandleClusterMemberOpinion(message.ConversationId, individualreq);
    break;
```

Figura 43 - Processamento de uma mensagem de etiqueta “OPINION_AGENT_REQUEST” no *UserAgent*

```
public VisitedAndOtherPOIsRatingDTO verifyOpinionAboutPOI()
{
    VisitedAndOtherPOIsRatingDTO visitedAndOtherPOIsRating = new VisitedAndOtherPOIsRatingDTO();
    visitedAndOtherPOIsRating.visitedPOIs = this.visitedPOIs;
    visitedAndOtherPOIsRating.otherPOIs = this.otherPOIs;
    visitedAndOtherPOIsRating.id = this.username;

    return visitedAndOtherPOIsRating;
}
```

Figura 44 – Criação do objeto do tipo *VisitedAndOtherPOIsRatingDTO*

```
private void HandleClusterMemberOpinion(String messageId, ResponseIndividualRecommendationsMAMSSDTO recommendationsMAMSDTO)
{
    VisitedAndOtherPOIsRatingDTO visitedAndOtherPOIsRating = verifyOpinionAboutPOI();

    LabeledContent poiInfo = new LabeledContent(LabeledContent.OPINION_AGENT_REPLY, visitedAndOtherPOIsRating);

    Send(messageId, poiInfo);
}
```

Figura 45 –Envio do *feedback* para o *UserAgent* que o requisitou

Quando o *UserAgent* do utilizador que efetuou o pedido de recomendação de POI recebe as mensagens com o *feedback* de um dado utilizador do cluster efetua uma chamada ao método *HandleAgentsOpinionReply* (Figura 46).

```
case LabeledContent.OPINION_AGENT_REPLY:
    VisitedAndOtherPOIsRatingDTO poiInfo = (VisitedAndOtherPOIsRatingDTO)receivedContent.content;
    HandleAgentsOpinionReply(poiInfo);
    break;
```

Figura 46 - Processamento de uma mensagem de etiqueta “OPINION_AGENT_REPLY” no *UserAgent*

Dentro do método *HandleAgentsOpinionReply,* o *UserAgent* do utilizador que efetuou o pedido de recomendação de POI sabe que um dado *UserAgent* já enviou o seu *feedback* através da remoção do id que lhe corresponde de uma lista auxiliar ao processamento que contém o id de todos os membros do cluster em análise. Por sua vez, o DTO do tipo *VisitedAndOtherPOIsRatingDTO* é adicionado a uma lista que é posteriormente explorada (Figura 47).

```
private void HandleAgentsOpinionReply(VisitedAndOtherPOIsRatingDTO info)
{
    //Remove the user from the opinion cluster members list
    clusterMembers.Remove(info.id);

    //Remember the Pois rating opinion
    poiInfos.Add(info);
}
```

Figura 47 – Receção de um *VisitedAndOtherPOIsRatingDTO* de um dado *UserAgent*

O método *ActDefault* de um *ResponseAgent* possui o comportamento *default* que o agente efetua quando não possui mensagens para tratar. Se a variável *opinionOngoing informar* que se encontra a decorrer uma obtenção de *feedback* (o seu valor é *true*) e caso não existam ids na lista clusterMembers, significa que todos os membros do cluster responderam. Quando isto acontece, o método *obtainPOILists* é executado, a variável de controlo opinionOngoing assume o valor *false* e a lista *clusterMembers* é preenchida com todos os ids dos membros do cluster do UserAgent (Figura 48).

```
public override void ActDefault(){
    if (opinionOngoing & clusterMembers.Count == 0)
    {

        obtainPOILists();
        opinionOngoing = false;
        this.clusterMembers = this.clusterMembersConst;
    }
}
```

Figura 48 – Método *ActDefault* do *UserAgent*

Dentro do método *obtainPOILists*, começa-se por guardar na lista *existingPois* todos os ids de POI que constam nos *ratings* recebidos (Figura 49). De seguida, para cada POI existente em *existingPois*, guarda-se o número total de avaliações de POI já visitados feitas pelos utilizadores do cluster do *UserAgent* atual para o POI em análise na variável *totalVisited,* assim como a soma de todas as avaliações na variável *totalVisitedRating*. Efetua-se o mesmo para as avaliações de POI ainda não visitados (variáveis *totalOther* e *totalOtherRating*). Caso o utilizador do *UserAgent* que efetuou o pedido de recomendação de POI tenha especificado que não pretende revisitar o POI em análise, a variável *dontWantToVisitAgain* assume o valor *true* para sinalizar a sua intenção (Figura 50).

```
//Saves the IDs of all the POIs that have ratings
List<string> existingPois = new List<string>();

//Obtain all ids of POIs that have ratings
foreach (VisitedAndOtherPOIsRatingDTO rating in this.poiInfos)
{
    foreach (POIRatingVisitedDTO poivi in rating.visitedPOIs)
    {
        //Add id if its is a new found id
        if (!existingPois.Exists(p => p == poivi.poi.poiId))
        {
            existingPois.Add(poivi.poi.poiId);
        }
    }
    foreach (POIRatingDTO poiot in rating.otherPOIs)
    {
        //Add id if its is a new found id
        if (!existingPois.Exists(p => p == poiot.poi.poiId))
        {
            existingPois.Add(poiot.poi.poiId);
        }
    }
}
```

Figura 49 – Obtenção dos POI que foram avaliados

```
//Analyse the global rating for each Poi with ratings
foreach (string existingId in existingPois)
{
    dontWantToVisitAgain = false;

    foreach(VisitedAndOtherPOIsRatingDTO rating in this.poiInfos)
    {
        //Verify if the the current POI has any visited ranking in the current cluster member opinion in poiInfos
        if(rating.visitedPOIs.Exists(p => p.poi.poiId == existingId))
        {
            POIRatingVisitedDTO poiCheck = rating.visitedPOIs.Find(p => p.poi.poiId == existingId);
            totalVisitedRating = totalVisitedRating + poiCheck.rating;
            totalVisited++;
            //Verify if the current rating is from the current user and if he want do revist the current POI
            if (!poiCheck.revisitable && this.username == rating.id)
            {
                dontWantToVisitAgain = true;
            }
            //Verify if the the current POI has any other ranking in the current cluster member opinion in poiInfos
        }
        else if (rating.otherPOIs.Exists(p => p.poi.poiId == existingId))
        {
            POIRatingDTO poiCheck = rating.otherPOIs.Find(p => p.poi.poiId == existingId);
            totalOtherRating = totalOtherRating + poiCheck.rating;
            totalOther++;
        }
    }
```

Figura 50 – Análise dos *ratings* associados a um determinado POI

Caso a variável *totalVisited* possua um valor diferente de zero, significa que o POI atual possui *ratings* associados. A média dos *ratings* atribuídos ao POI é calculada e guardada na variável *visitedRating.* Caso a média seja superior a três e o valor da variável *dontWantToVisitAgain* seja false (pretende voltar a visitar o POI atual), um objeto do tipo *POINumberOfRatesDTO* é criado e adicionado à lista *wantedPOIs,* que guarda os POI que devem ter prioridade no pedido de recomendação. Caso esta condição não se verifique, um objeto do tipo *POINumberOfRatesDTO* também é criado, mas desta vez, adicionado à lista *unwantedPOIs,* que guarda os POI que devem ser penalizados no pedido de recomendação (Figura 51). Os objetos do tipo *POINumberOfRatesDTO*, guardam o id do POI, o *rating* médio obtido e os valores que constam nas variáveis *totalVisited* e *totalOther* para o POI em questão. A lógica para as avaliações de POI que ainda não foram visitados é idêntica à explicada.

```
//Verify if this POI had any rating
if(totalVisited != 0)
{
    double visitedRating = totalVisitedRating / totalVisited;
    //Add POI to unwanted POIs because the average visited rating is low
    if (visitedRating < 3)
    {
        POINumberOfRatesDTO unwantedPoi = new POINumberOfRatesDTO
        {
            poiId = existingId,
            rating = visitedRating,
            totalVisitedRating = totalVisited,
            totalOtherRating = totalOther
        };

        unwantedPOIs.Add(unwantedPoi);
    }
    //Only add to the wantedPOIs list if he wants to visit again
    else if(!dontWantToVisitAgain)
    {
        POINumberOfRatesDTO wantedPoi = new POINumberOfRatesDTO
        {
            poiId = existingId,
            rating = visitedRating,
            totalVisitedRating = totalVisited,
            totalOtherRating = totalOther
        };

        wantedPOIs.Add(wantedPoi);
    //It can have positive rating but still don't want to revisit
    }else if (dontWantToVisitAgain)
    {
        POINumberOfRatesDTO unwantedPoi = new POINumberOfRatesDTO
        {
            poiId = existingId,
            rating = visitedRating,
            totalVisitedRating = totalVisited,
            totalOtherRating = totalOther
        };

        unwantedPOIs.Add(unwantedPoi);
    }
```

Figura 51 – Adição de *POINumberOfRatesDTOs* às listas *unwantedPOIs* e *wantedPOIs*

Quando todos os POI são analisados, um objeto do tipo *RatingListsDTO*, que guarda as listas *unwantedPOIs* e *wantedPOIs*, é criado e enviado através do método *Respond* para o *MASService numa mensagem de etiqueta "*INDIVIDUAL_RECOMMENDATION_REPLY_OPINION*"* (Figura 52).

```
RatingListsDTO ratingListsDTO = new RatingListsDTO()
{
    unwantedPOIs = unwantedPOIs,
    wantedPOIs = wantedPOIs
};

Respond(opinionRequestId, LabeledContent.INDIVIDUAL_RECOMMENDATION_REPLY_OPINION, ratingListsDTO);
```

Figura 52 – Resposta ao *MASService* com os POI a dar prioridade e a penalizar

Quando o *MASService* recebe a mensagem de resposta do *UserAgent* com o *RatingListsDTO, um* JSON formatado é criado e enviado para o microserviço responsável pelas recomendações

que irá priorizar os POI constantes na lista *wantedPOIs* e penalizar os POI constantes na lista *unwantedPOIs*, tendo sempre em conta as categorias mais adequadas ao perfil do utilizador (Figura 53). As recomendações de POI resultantes do pedido POST são finalmente enviadas como resposta ao pedido de recomendação de POI.

```
myContent = JsonConvert.SerializeObject(opinionDTO);
buffer = Encoding.UTF8.GetBytes(myContent);
byteContent = new ByteArrayContent(buffer);
byteContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

result = await client.PostAsync(URL_REMS, byteContent);
result.EnsureSuccessStatusCode();
json = await result.Content.ReadAsStringAsync();
firstList = JsonConvert.DeserializeObject<ResponseIndividualRecommendationsMAMSDTO>(json);

return firstList;
```

Figura 53 – Pedido de recomendações individuais ao microserviço responsável pelas recomendações (REMS)

5.1.4 Pedido de recomendação de POI para grupo

O método *RequestRecommendation* do *GroupController* é responsável pela gestão do *endpoint* que lida com os pedidos de recomendação de POI para grupo. O método *RequestRecommendation* do *MASService* é chamado para processar o pedido de recomendação do grupo cujos membros constam no DTO do tipo *GroupDTO* passado por parâmetro (Figura 54). Este DTO possui informação que é utilizada no processamento do pedido de recomendação para grupo.

```
[HttpPost("{group}/requestRecommendation")]

public async Task<IActionResult> RequestRecommendation(string group, GroupDTO dto)
{
    try
    {
        //Request the recommendatiosn from the MAS
        var result = await masService.RequestRecommendation(dto);
        return Ok(result);
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 54 - Método responsável por gerir o *endpoint* que processa os pedidos de recomendação de POI para grupo

Dentro do método *RequestRecommendation* do *MASService*, verifica-se a existência de um *GroupAgent* encarregue do grupo, através do id do grupo, e verifica-se também se todos os membros do grupo possuem um *UserAgent* associado. O id e os membros do grupo são obtidos através do *GroupDTO* passado como argumento. Caso o grupo ainda não possua um *GroupAgent* associado, um novo agente é criado através do método *CreateNewGroupAgent*.

Feita a verificação, uma mensagem de etiqueta “GROUP_DIVISION_REQUEST” e com o *GroupDTO* associado é enviada ao *GroupAgent* através do método *SendMessageAndExpectResponse* (Figura 55).

```
public async Task<GroupDivisionPOIMAMSDTO> RequestRecommendation(GroupDTO recRequest)
{
    //Verify if all group members exist
    foreach (GroupMembersDTO member in recRequest.mainGroup.members)
    {
        if (TestMASConnection(member.nickname).Result.Equals("failed"))
        {
            Console.WriteLine("User agent " + member.nickname + " could not be found");
            return null;
        }
    }

    //Verify if the group exists
    if (TestMASConnection(recRequest.mainGroup.id.ToString()).Result.Equals("failed"))
    {
        //Create the group agent if it doesn't already exist
        await CreateNewGroupAgent(recRequest);

        Console.WriteLine("Created Travel Agent " + recRequest.mainGroup.id + " for group " + recRequest.mainGroup.name);
    }

    LabeledContent response = SendMessageAndExpectResponse(recRequest.mainGroup.id.ToString(), LabeledContent.GROUP_DIVISION_REQUEST, recRequest);
```

Figura 55 – Pedido de divisão de grupo para o GroupAgent

O *GroupAgent*, ao receber uma mensagem etiquetada com “GROUP_DIVISION_REQUEST”, utiliza a variável *divisionOngoing* para sinalizar que o agente já se encontra a verificar uma possível divisão do grupo original e efetua uma chamada ao método HandleRecommendationRequest. O *conversationId* da mensagem é passado por parâmetro para que o *GroupAgent* possa responder ao *MASService* no fim do processamento.

O método *HandleRecommendationRequest* efetua uma verificação inicial à informação presente no *GroupDTO* recebido como argumento e inicializa uma lista de subgrupos (*subGroups*) para guardar os subgrupos que possam ser criados, uma lista com todos os membros do grupo original (*divisionMembers*), e uma lista para armazenar as repostas dos membros do grupo original quando lhes é questionado o cluster a que pertencem (*memberReplies*). Por fim, utiliza o método *Send* para enviar mensagens de etiqueta “GROUP_DIVISION_REQUEST” a todos os *UserAgents* dos membros do grupo original com o intuito de obter o cluster a que cada um pertence (Figura 56).

```
private void HandleRecommendationRequest(GroupDTO recReq,String groupId){
    //Update this group's info
    UpdateGroupInfo(recReq);
    //Clear any previous list of subgroups
    subGroups = new List<SubGroup>();
    divisionMembers = new List<string>();
    divisionMembers.AddRange(members);
    memberReplies = new List<GroupDivisionReply>();

    //Send the members a request to negotiate the subgroups
    foreach (string member in members){
        //Create the group division request content for use in the message to be sent to the users
        LabeledContent groupDivisionRequestInfo = new LabeledContent(
                                                LabeledContent.GROUP_DIVISION_REQUEST,
                                                "");

        Send(member,groupDivisionRequestInfo,groupId);
    }
}
```

Figura 56 – Envio de mensagens de etiqueta "GROUP_DIVISION_REQUEST" aos *UserAgents* dos membros do grupo original

Quando os *UserAgents* recebem as mensagens de etiqueta "GROUP_DIVISION_REQUEST", executam o método HandleGroupDivisionRequest, que cria um objeto do tipo *GroupDivisionReply* onde consta informação relevante para o processamento do pedido de recomendação, nomeadamente, o id do cluster a que o utilizador pertence e os membros do cluster. O *GroupDivisionReply* é enviado para o *GroupAgent* numa mensagem de etiqueta "GROUP_DIVISION_NEGOTIATION_REPLY" através do método *Send* (Figura 57).

```
private void HandleGroupDivisionRequest(String messageId){
    //Calculate the personality match
    PersonalityMatch personalityMatch = new PersonalityMatch(this.username,this.personality
        ,this.travel_preferences,this.motivations);

    //Create the Division Reply
    GroupDivisionReply reply = new GroupDivisionReply(personalityMatch, this.clusterMembersConst,
        this.clusterId,this.fears,this.limitations,this.rules);

    //Construct the labeled content for the response
    LabeledContent response = new LabeledContent(
                                        LabeledContent.GROUP_DIVISION_NEGOTIATION_REPLY,
                                        reply);

    Send(messageId,response);
}
```

Figura 57 – Envio da *GroupDivisionReply* ao *GroupAgent*

Quando o *GroupAgent* recebe uma resposta dos *UserAgents*, utiliza o método *HandleGroupDivisionNegotiationReply* para remover o id do membro autor da resposta da lista *divisionMembers* e adiciona a *GroupDivisionReply* à lista *memberReplies* (Figura 58).

```
private void HandleGroupDivisionNegotiationReply(GroupDivisionReply reply){
    //Remove the user from the division list
    divisionMembers.Remove(reply.personalityMatch.tourist);

    //Remember the personality match
    memberReplies.Add(reply);
}
```

Figura 58 – Receção da *GroupDivisionReply* no *GroupAgent*

Dentro do método *ActDefault* do *GroupAgent*, caso a variável *divisionOngoing* sinalizar que uma divisão de grupo está em curso e a lista *divisionMembers* estiver vazia, significa que todos os *UserAgents* dos membros do grupo original responderam com a sua *GroupDivisionReply* e o método *HandleGroupDivision* é executado. No fim da sua execução, o valor da variável *divisionOngoing* é alterado para sinalizar que o *GroupAgent* já não se encontra a processar a divisão dos membros do grupo (Figura 59).

```
public override void ActDefault(){
    if (divisionOngoing & divisionMembers.Count == 0){
        //The user agents have finished sending their info
        //Construct the subgroups
        HandleGroupDivision();
        divisionOngoing = false;
    }
```

Figura 59 – Tratamento da divisão de grupos no *GroupAgent* após todos os *UserAgents* responderem

Dentro do método *HandleGroupDivision*, para cada *GroupDivisionReply* da lista *memberReplies*, verifica-se a existência de um subgrupo na lista *subGroups* cujo id do cluster que o originou (corresponde ao seu nome) é idêntico ao id do cluster presente na *GroupDivisionReply*. Caso esse subgrupo exista, o membro autor da *GroupDivisionReply* é adicionado ao subgrupo, caso contrário, um novo subgrupo é criado e o membro autor da *GroupDivisionReply* é adicionado ao novo subgrupo. Por fim, uma mensagem de etiqueta "GROUP_DIVISION_REPLY", com os subgrupos presentes num objeto do tipo *RecommendationDTO*, é enviada ao *MASService* através do método *Respond* (Figura 60).

```
foreach (GroupDivisionReply reply in memberReplies){
    //If the subgroup has already been created
    if (addedSubGroups.Contains(reply.clusterId)){
        //Find the existing subgroup
        foreach (SubGroup psg in subGroups){
            if (psg.name.Equals(reply.clusterId)){

                if(this.clusterId == null)
                {
                    this.clusterId = reply.clusterId;
                    this.clusterMembersConst = reply.clusterMembers;
                    this.clusterMembers = reply.clusterMembers;
                }
                //Add the tourist to the subgroup
                psg.AddMember(reply.personalityMatch.tourist, reply.personalityMatch.personality,
                    reply.personalityMatch.travelPreferences,reply.personalityMatch.motivations,reply.personalityMatch.categories,
                    reply.fears,reply.limitations);
                break;
            }
        }
    } else {
        //Create a new subgroup with the match's category and add the user to it
        SubGroup newSG = new SubGroup(reply.clusterId,reply.personalityMatch.bestCategoryDescription,
            reply.personalityMatch.bestCategoryJustification,reply.clusterMembers,reply.rules);
        newSG.AddMember(reply.personalityMatch.tourist,reply.personalityMatch.personality,reply.personalityMatch.travelPreferences,
            reply.personalityMatch.motivations,reply.personalityMatch.categories, reply.fears, reply.limitations);
        //Add the new subgroup to the group's list
        subGroups.Add(newSG);
        //Add the subgroup to the auxiliary list
        addedSubGroups.Add(reply.clusterId);
    }
}

RecommendationDTO dto = new RecommendationDTO {subGroups = subGroups};
Respond(divisionRequestId, LabeledContent.GROUP_DIVISION_REPLY, dto);
```

Figura 60 – Criação dos subgrupos com base no cluster dos membros do grupo original

Quando os subgrupos formados chegam ao MASService, força-se para que só existam subgrupos com um mínimo de três membros. Para cada subgrupo, começando por aqueles que possuem menos membros, compara-se a personalidade individual de cada membro com a personalidade média dos restantes subgrupos. Esta comparação é feita com base na fórmula da distância euclidiana utilizada no método *EuclideanPersonalityDTO*. O subgrupo com maior proximidade à personalidade do membro em análise passa a ser o subgrupo desse utilizador (Figura 61). Este processo repete-se até não existirem subgrupos com menos de 3 pessoas.

```
foreach (SubgroupMAMSDTO sub in subgroupsCopy)
{
    if (sub.members.Count() < 3)
    {
        //Obtain list with the other subgroups
        List<SubgroupMAMSDTO> otherSubgroups = (List<SubgroupMAMSDTO>)listSubGroups.Where(o => o.id != sub.id).ToList();

        foreach (UserMAMSDTO us in sub.members)
        {
            //Variable to save the best distance of a subgroup with less than 3
            double bestDistance = Double.MaxValue;
            //Variable to save the best subgroup id
            long bestSubgroup = long.MinValue;

            // sort other subgroups by distance to ensure consistent selection
            otherSubgroups.Sort((s1, s2) =>
            Utils.Distance.EuclideanPersonalityDTO(us.personality, s1.personality).CompareTo(Utils.Distance.EuclideanPersonalityDTO(us.personality, s2.personality)));

            // select closest subgroup
            SubgroupMAMSDTO newSub = otherSubgroups.FirstOrDefault();

            newSub.members.Add(us);
            //Update totalMembers
            newSub.numMembers = newSub.members.Count();
        }

        listSubGroups.Remove(sub);
        break;
    }
}
```

Figura 61 - Reprocessamento dos subgrupos para terem no mínimo três membros

Para cada um dos subgrupos originados após este processo final, é criado um *GroupAgent* encarregue pela sua gestão através do método *CreateNewSubGroupAgent* (Figura 62).

```
foreach (SubgroupMAMSDTO sub in recRequestToEngine.mainGroup.subgroups)
{
    //Verify if the subgroup exists
    if (TestMASConnection(recRequestToEngine.mainGroup.id.ToString() + "sub" + sub.id.ToString()).Result.Equals("failed"))
    {
        //Create the subgroup agent if it doesn't already exist
        await CreateNewSubGroupAgent(sub, recRequest);
    }
}
```

Figura 62 – Criação do *GroupAgent* de cada subgrupo

Para cada *GroupAgent* de cada subgrupo criado, é enviada uma mensagem etiquetada com "GROUP_RECOMMENDATION_REQUEST_OPINION" (Figura 63). Quando o *GroupAgent* recebe esta mensagem, sinaliza que um pedido de *feedback* está a ser processado através da variável *opinionOngoing* e efetua uma chamada ao método *HandleAgentsOpinion*. O *conversationId* é passado como argumento para que seja possível responder ao *MASService* no fim do processamento (Figura 64).

```
foreach (var subgroup in recRequestToEngine.mainGroup.subgroups)
{
    //Ask for agents opinions before send any recommendation list to group
    LabeledContent responseOpinion = SendMessageAndExpectResponse(recRequestToEngine.mainGroup.id.ToString() + "sub" + subgroup.id.ToString(),
        LabeledContent.GROUP_RECOMMENDATION_REQUEST_OPINION, recommendationsAfterRenegotiation);
```

Figura 63 – Envio de mensagens de etiqueta"GROUP_RECOMMENDATION_REQUEST_OPINION" para os *GroupAgents* dos subgrupos

```
case LabeledContent.GROUP_RECOMMENDATION_REQUEST_OPINION:
    //Make sure that no recommendation request opinions are already ongoing
    if (opinionOngoing)
    {
        //If opinion is ongoing, ignore this request
        break;
    }

    //Opinions are not ongoing, update the current status to reflect that one has just started
    opinionOngoing = true;

    //Convert the contents into ResponseIndividualRecommendationsMAMSDTO format
    GroupDivisionPOIMAMSDTO individualRecommendationDTO = (GroupDivisionPOIMAMSDTO) receivedContent.content;
    this.opinionRequestId = message.ConversationId;
    //Save last recommendation
    this.groupRecommendationDTO = individualRecommendationDTO;
    //Handle the individual reccomendation request opinions
    HandleAgentsOpinion(this.id, individualRecommendationDTO);
    break;
```

Figura 64 – Receção de mensagens de etiqueta "GROUP_RECOMMENDATION_REQUEST_OPINION" nos *GroupAgents* dos subgrupos

Dentro do método HandleAgentsOpinion, uma mensagem de etiqueta "OPINION_GROUP_AGENT_REQUEST" é enviada aos *UserAgents* dos membros do cluster que originou inicialmente o subgrupo e aos *UserAgents* dos novos membros adicionados *à posteriori* no processamento executado no *MASService* (Figura 65).

```
private void HandleAgentsOpinion(String groupXId, GroupDivisionPOIMAMSDTO recommendationsMAMSDTO)
{

    //Ask opinon of each member of this group cluster
    foreach (string member in clusterMembers)
    {
        //Construct the labeled content for the response
        LabeledContent agentsOpinionInfo = new LabeledContent(
                                            LabeledContent.OPINION_GROUP_AGENT_REQUEST,
                                            recommendationsMAMSDTO);

        Send(member, agentsOpinionInfo, groupXId);
    }
}
```

Figura 65 – Pedido de feedback efetuado pelo GroupAgent

Quando o *UserAgent* recebe o pedido de feedback do *GroupAgent*, efetua uma chamada ao método *HandleGroupClusterMemberOpinion* que por sua vez responde ao GroupAgent com uma mensagem de etiqueta "GROUP_DIVISION_NEGOTIATION_REPLY" (Figura 66). A mensagem possui um objeto do tipo *VisitedAndOtherPOIsRatingDTO* já detalhado na secção 5.1.3 que é obtido através do método *groupVerifyOpinionAboutPOI* (Figura 67).

```
private void HandleGroupClusterMemberOpinion(String messageId, GroupDivisionPOIMAMSDTO recommendationsMAMSDTO)
{
    VisitedAndOtherPOIsRatingDTO visitedAndOtherPOIsRating = null;

    visitedAndOtherPOIsRating = groupVerifyOpinionAboutPOI(recommendationsMAMSDTO.mainGroup.recommendationsList);

    LabeledContent poiInfo = new LabeledContent(LabeledContent.OPINION_GROUP_AGENT_REPLY, visitedAndOtherPOIsRating);

    Send(messageId, poiInfo);
}
```

Figura 66 – Envio do objeto do tipo *VisitedAndOtherPOIsRatingDTO* ao *GroupAgent*

```
public VisitedAndOtherPOIsRatingDTO groupVerifyOpinionAboutPOI(List<RecommendationListDTO> recommendationList)
{

    //Set an object with all the POIs that had a rating and were in the recommendation list
    VisitedAndOtherPOIsRatingDTO visitedAndOtherPOIsRating = new VisitedAndOtherPOIsRatingDTO();
    visitedAndOtherPOIsRating.visitedPOIs = this.visitedPOIs;
    visitedAndOtherPOIsRating.otherPOIs = this.otherPOIs;
    visitedAndOtherPOIsRating.id = this.username;

    return visitedAndOtherPOIsRating;
}
```

Figura 67 – Criação do objeto do tipo *VisitedAndOtherPOIsRatingDTO*

Quando o *GroupAgent* recebe uma mensagem de etiqueta "GROUP_DIVISION_NEGOTIATION_REPLY" sinaliza que o *UserAgent* já enviou o seu feedback. Para isso, remove o id do *UserAgent* da lista auxiliar ao processamento *clusterMembers*, e adiciona o objeto do tipo *VisitedAndOtherPOIsRatingDTO* à lista *poiInfos* (Figura 68).

```
private void HandleAgentsOpinionReply(VisitedAndOtherPOIsRatingDTO info)
{
    //Remove the user from the opinion cluster members list
    clusterMembers.Remove(info.id);

    //Remember the Pois rating opinion
    poiInfos.Add(info);
}
```

Figura 68 – Receção do objeto do tipo *VisitedAndOtherPOIsRatingDTO* no *GroupAgent*

No método *ActDefault* do *GroupAgent*, se a variável *opinionOngoing* sinalize que um pedido de feedback está a decorrer e caso a lista *clusterMembers* esteja vazia, significa que todos os *UserAgents* dos membros a quem se requisitou feedback já responderam e o método *ObtainInclusionAndExclusionPOILists* é executado (Figura 69).

```
if (opinionOngoing & clusterMembers.Count == 0)
{

    ObtainInclusionAndExclusionPOILists();
    opinionOngoing = false;
    this.clusterMembers = new List<string>(this.clusterMembersConst);
}
```

Figura 69 – Processamento do *GroupAgent* após todos os *UserAgents* fornecerem *feedback*

A lógica do método *ObtainInclusionAndExclusionPOILists* é idêntica à do método *obtainPOILists* explicada na secção 5.1.3. A única diferença é que nos pedidos de recomendação de POI para grupo, a variável *dontWantToVisitAgain* consiste num contador que é incrementado à medida que um membro do subgrupo especifica que não pretende revisitar um dado POI (Figura 70) para posteriormente se verificar se a maioria pretende ou não visitar esse dado POI (Figura 71).

```
//Verify if the current rating is from a group member and if she/he don't want to visit again the current POI
if (!poiCheck.revisitable && this.membersSubgroup.Exists(m => m == rating.id))
{
    dontWantToVisitAgain++;
}
```

Figura 70 – Verificação da intenção de revisita a um dado POI por parte de um membro do subgrupo

```
//Verify if the majority don't want to visit again
else if( (this.membersSubgroup.Count() / 2) > dontWantToVisitAgain ){
    POINumberOfRatesDTO wantedPoi = new POINumberOfRatesDTO
    {
        poiId = existingId,
        rating = visitedRating,
        totalVisitedRating = totalVisited,
        totalOtherRating = totalOther
    };

    wantedPOIs.Add(wantedPoi);
}else if ((this.membersSubgroup.Count() / 2) <= dontWantToVisitAgain)
{
    POINumberOfRatesDTO unwantedPoi = new POINumberOfRatesDTO
    {
        poiId = existingId,
        rating = visitedRating,
        totalVisitedRating = totalVisited,
        totalOtherRating = totalOther
    };
    unwantedPOIs.Add(unwantedPoi);
}
```

Figura 71 – Priorização ou Penalização de um POI com base na opinião da maioria

Por fim, à semelhança do que acontece na secção 5.1.3, um objeto do tipo *RatingListsDTO* é criado e enviado numa mensagem de etiqueta “GROUP_RECOMMENDATION_REPLY_OPINION” para o *MASService* através do método *Respond* (Figura 72).

```
RatingListsDTO ratingListsDTO = new RatingListsDTO()
{
    unwantedPOIs = unwantedPOIs,
    wantedPOIs = wantedPOIs
};

Respond(opinionRequestId, LabeledContent.GROUP_RECOMMENDATION_REPLY_OPINION, ratingListsDTO);
```

Figura 72 – Envio do objeto do tipo *RatingListsDTO* ao *MASService*

Quando o *MASService* recebe o *feedback* enviado pelo *GroupAgent* do subgrupo em análise, este efetua um pedido de informação relativa a rules ao mesmo *GroupAgent* através de uma mensagem de etiqueta “GROUP_RULES_INFO_REQUEST” (Figura 73).

```
//Now check if there is rules
LabeledContent responseRulesInfo = SendMessageAndExpectResponse(recRequestToEngine.mainGroup.id.ToString() + "sub" + subgroup.id.ToString(),
    LabeledContent.GROUP_RULES_INFO_REQUEST);
```

Figura 73 - Pedido de informação relativa a rules ao *GroupAgent*

O *GroupAgent*, ao receber o pedido de informação relativa a rules, sinaliza o início da sua obtenção através da variável *rulesInfoOngoing* e efetua uma chamada ao método *HandleRulesInfoRequest*. O *conversationId* é passado como parâmetro para que no fim do processamento se possa enviar a informação relativa a *rules* ao *MASService* (Figura 74).

```
case LabeledContent.GROUP_RULES_INFO_REQUEST:
    //Make sure that no info request is already ongoing
    if (rulesInfoOngoing)
    {
        //If request is ongoing, ignore this request
        break;
    }

    //Rules info requests are not ongoing, update the current status to reflect that one has just started
    rulesInfoOngoing = true;

    this.rulesInfoRequestId = message.ConversationId;

    HandleRulesInfoRequest(this.id);

    break;
```

Figura 74 – Receção de uma mensagem de etiqueta“GROUP_RULES_INFO_REQUEST” no *GroupAgent*

O método *HandleRulesInfoRequest* envia uma mensagem de etiqueta “GROUP_RULES_INFO_AGENT_REQUEST” para os *UserAgents* dos membros do subgrupo que o *GroupAgent* gere através do método *Send* (Figura 75).

```
public void HandleRulesInfoRequest(String groupXId)
{
    //Send the members a request to obtain rules info
    foreach (string member in membersSubgroup)
    {
        //Construct the labeled content for the response
        LabeledContent  agentsRulesInfo = new LabeledContent(
                                            LabeledContent.GROUP_RULES_INFO_AGENT_REQUEST,
                                            groupXId);

        Send(member, agentsRulesInfo, groupXId);
    }
}
```

Figura 75 – Envio de uma mensagem de etiqueta “GROUP_RULES_INFO_AGENT_REQUEST” para os *UserAgents* dos membros do subgrupo do *GroupAgent*

Quando o *UserAgent* recebe mensagens de etiqueta “GROUP_RULES_INFO_AGENT_REQUEST”, executa o método *HandleRulesInfoRequest* para criar um objeto do tipo *RuleInfo*, que contém informação utilizada mais tarde no processamento de *rules,* e envia-o numa mensagem de etiqueta “GROUP_RULES_INFO_AGENT_REPLY” para o *GroupAgent* do subgrupo (Figura 76).

```
private void HandleRulesInfoRequest(String messageId)
{
    RuleInfo ruleInfo = new RuleInfo()
    {
        user_id = this.username,
        hasChildren = this.hasChildren,
        civil_state = this.civil_state,
        professional_situation = this.professional_situation
    };
    LabeledContent info = new LabeledContent(LabeledContent.GROUP_RULES_INFO_AGENT_REPLY, ruleInfo)

    Send(messageId, info);
}
```

Figura 76 - Envio de uma mensagem de etiqueta “GROUP_RULES_INFO_AGENT_REPLY” para o *GroupAgent*

Quando o *GroupAgent* recebe uma mensagem de etiqueta "GROUP_RULES_INFO_AGENT_REPLY", executa o método *HandleRuleInfoReply*. Este método é responsável por sinalizar a chegada de informação relativa ao processamento de *rules* através da remoção do id do utilizador que enviou a informação da lista *membersSubgroupAux* e da adição do objeto do tipo *RuleInfo* à lista *rulesInfo* (Figura 77).

```
private void HandleRuleInfoReply(RuleInfo ruleInfo)
{
    //Remove the user from the auxiliar subgroup member list
    membersSubgroupAux.Remove(ruleInfo.user_id);

    //Save the user rules info in a list
    rulesInfo.Add(ruleInfo);
}
```

Figura 77 – Receção de uma mensagem de etiqueta "GROUP_RULES_INFO_AGENT_REPLY" no *GroupAgent*

No método *ActDefault* do *GroupAgent*, é possível verificar se já todos os *UserAgents* dos membros do subgrupo enviaram a sua informação relativa ao processamento de *rules*. Esta condição verifica-se caso a variável *rulesInfoOngoing* indique que está a ocorrer um processo de obtenção desse tipo de informação e caso a lista *membersSubgroupAux* se encontre vazia (Figura 78).

```
if(rulesInfoOngoing & membersSubgroupAux.Count == 0) {

    ObtainValidRules();
    rulesInfoOngoing = false;
    this.membersSubgroupAux = new List<string>(this.membersSubgroup);
    this.rulesInfo = new List<RuleInfo>();
}
```

Figura 78 - Procedimento a executar caso todos os *UserAgents* dos membros do subgrupo tenham enviado a sua informação relativa ao processamento de *rules*

O método *ObtainValidRules* percorre uma lista de objetos do tipo *RuleDTO* que correspondem às *rules* do cluster do subgrupo. Para cada uma das rules em análise, é verificado se a maioria dos membros do subgrupo possui a condição necessária para *rule* ter efeito. Caso se verifique a validade de uma *rule*, esta é adicionada à lista *finalRules* que armazena as rules válidas. A análise da sua validade é feita através da informação relativa ao processamento de rules enviada pelos *UserAgents.* Por fim, uma mensagem de etiqueta "GROUP_RULES_INFO_REPLY" é enviada em conjunto com a lista *finalRules* ao *MASService* (Figura 79).

```
foreach(RuleDTO rule in rules)
{
    if(rule.hasChildren != null)
    {
        total = rulesInfo.Where(x => x.hasChildren == rule.hasChildren).Count();

        if (total >= halfSubgroupMembers)
        {
            finalRules.Add(rule);
        }


    }else if(rule.professional_situation != null)
    {
        total = rulesInfo.Where(x => x.professional_situation == rule.professional_situation).Count();

        if (total >= halfSubgroupMembers)
        {
            finalRules.Add(rule);
        }

    }
    else if (rule.civil_state != null)
    {
        total = rulesInfo.Where(x => x.civil_state == rule.civil_state).Count();

        if (total >= halfSubgroupMembers)
        {
            finalRules.Add(rule);
        }
    }
}

Respond(rulesInfoRequestId, LabeledContent.GROUP_RULES_INFO_REPLY, finalRules);
```

Figura 79 – Obtenção das *rules* válidas no *GroupAgent*

Quando o *MASService* recebe a lista *finalRules*, percorre-a e adiciona objetos do tipo *POINumberOfRatesDTO* às listas *unwantedPOIs* e *wantedPOIs* do objeto do tipo *RatingListsDTO* obtido anteriormente. Caso se trate de uma *rule* positiva, adiciona à lista *wantedPOIs*, caso contrário, adiciona à lista *unwantedPOIs* (Figura 80). Numa *rule* positiva, o POI associado deve ser priorizado, por outro lado, numa *rule* negativa, o POI deve ser penalizado. Por exemplo, para uma *rule* positiva que informe que é comum gostar do POI cujo id é 801 caso o utilizador tenha filhos (campo *hasChildren*), o POI é priorizado se pelo menos metade dos membros do subgrupo possuir filhos. Depois de o processo descrito ser efetuado para todos os subgrupos, um JSON formatado é criado e enviado para o microserviço responsável pelas recomendações que irá priorizar os POI constantes na lista *wantedPOIs* e penalizar os POI constantes na lista *unwantedPOIs*, tendo sempre em conta as categorias mais adequadas ao perfil do utilizador (Figura 81).

As recomendações de POI de cada subgrupo resultantes do pedido POST são finalmente enviadas como resposta ao pedido de recomendação de POI. Caso nenhum subgrupo tenha sido criado e o grupo original tenha prevalecido, a lógica descrita aplica-se de igual forma para esse grupo como se só existisse um subgrupo.

```
//Update all ratings of pois with the rules
foreach (RuleDTO rule in finalRules)
{
    //Add poi to wantedList
    if (rule.isPositive)
    {
        //If the poi exist in unwantedPOIs, remove it
        ratingLists.unwantedPOIs.RemoveAll(x => x.poiId == rule.poi);
        if (ratingLists.wantedPOIs.Where(x => x.poiId == rule.poi).Count() < 1)
        {
            POINumberOfRatesDTO ruleRating = new POINumberOfRatesDTO()
            {
                rating = 0,
                poiId = rule.poi,
                totalOtherRating = 0,
                totalVisitedRating = 0
            };
            ratingLists.wantedPOIs.Add(ruleRating);
        }
    }
    //Add poi to unwantedList
    else
    {
        //If the poi exist in wantedPOIs, remove it
        ratingLists.wantedPOIs.RemoveAll(x => x.poiId == rule.poi);
        if (ratingLists.unwantedPOIs.Where(x => x.poiId == rule.poi).Count() < 1)
        {
            POINumberOfRatesDTO ruleRating = new POINumberOfRatesDTO()
            {
                rating = 0,
                poiId = rule.poi,
                totalOtherRating = 0,
                totalVisitedRating = 0
            };
            ratingLists.unwantedPOIs.Add(ruleRating);
        }
    }
}
```

Figura 80 – Processamento das *rules* no *MASService*

```
myContent = JsonConvert.SerializeObject(opinionDTO);
buffer = Encoding.UTF8.GetBytes(myContent);
byteContent = new ByteArrayContent(buffer);
byteContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

result = await client.PostAsync(URL_REMS_GROUP, byteContent);
result.EnsureSuccessStatusCode();
json = await result.Content.ReadAsStringAsync();
recommendationsAfterRenegotiation = JsonConvert.DeserializeObject<GroupDivisionPOIMAMSDTO>(json);

return recommendationsAfterRenegotiation;
```

Figura 81 – Pedido de recomendações de POI para grupo microserviço responsável pelas recomendações (REMS)

5.1.5 Inicializar os agentes do sistema com *feedback* sobre POI

O método *BootstrapMAS* do *UserController* é utilizado para recriar os agentes de todos os utilizadores existentes na base de dados do MAMS Service. Este método é importante uma vez que sempre que o microserviço é reiniciado, os agentes morrem. A lista com todos os utilizadores é obtida através do método *getAllUsers* do *registerUserService* e passada como parâmetro na chamada ao método *CreateNewUserAgentFromList* do *MASService* (Figura 82).

```
[HttpPost("bootstrapMAS")]
public async Task<IActionResult> BootstrapMAS()
{
    try
    {
        //Retrieve and initiate all users in the MAS
        await masService.CreateNewUserAgentFromList(await registerUserService.getAllUsers());

        return Ok(new { message = "MAS bootstrapped successfully!" });
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 82 – Método da camada *Controller* utilizado para reviver os agentes do MAMS Service

No método *CreateNewUserAgentFromList*, foi necessário adicionar uma chamada ao método *obtainRatingsFromPOIService* (Figura 83) para efetuar um pedido dos ratings dos utilizadores ao microserviço POI Service, de modo a modelar os agentes com essa informação posteriormente no método *CreateNewUserAgent* (Figura 84).

```
public async Task<List<POIRatingInterestDTO>> obtainRatingsFromPOIService()
{

    var result = await client.GetAsync(URL_POIS);
    result.EnsureSuccessStatusCode();
    var json = await result.Content.ReadAsStringAsync();
    return JsonConvert.DeserializeObject<List<POIRatingInterestDTO>>(json);
}
```

Figura 83 – Pedido de rating ao POI Service

```
public async Task CreateNewUserAgentFromList(List<CreateUserDTO> listUserDTO)
{

    //Obtain all existing ratings
    List<POIRatingInterestDTO> ratingsObtained = await obtainRatingsFromPOIService();

    foreach (CreateUserDTO userDTO in listUserDTO)
    {
        List<POIRatingInterestDTO> userRatings = ratingsObtained.Where(r => r.username == userDTO.nickname).ToList();

        CreateNewUserAgent(userDTO, userRatings);

    }
}
```

Figura 84– Criação dos agentes dos utilizadores existentes na base de dados do MAMS Service

Dentro do método *CreateNewUserAgent,* mapeiam-se os ratings para listas de objetos do tipo *POIRatingDTO* e *POIRatingVisitedDTO* conforme o valor existente no campo *visited* dos objetos de tipo *POIRatingInterestDTO* recebidos no pedido ao POI Service (Figura 85).

```
public async Task CreateNewUserAgent(CreateUserDTO createUserDTO, List<POIRatingInterestDTO> ratingsObtained)
{
    //List to save the ratings
    List<POIRatingVisitedDTO> visitedPOIs = new List<POIRatingVisitedDTO>();
    List<POIRatingDTO> interestedPOIs = new List<POIRatingDTO>();

    foreach (POIRatingInterestDTO r in ratingsObtained)
    {
        if (r.visited == false)
        {
            interestedPOIs.Add(RatingMapper.toPOIRatingDTO(r));
        }
        else
        {
            visitedPOIs.Add(RatingMapper.toPOIRatingVisitedDTO(r));
        }
    }

    //Creating the dictionary the agent will use to respond
    ConcurrentDictionary<Guid, Message> responseDictionary = new ConcurrentDictionary<Guid, Message>();
    //Associating the dictionary to the agent for later lookups
    agentResponseDictionary.Add(createUserDTO.nickname, responseDictionary);
    //Adding the agent to the MAS
    environment.Add(new UserAgent(createUserDTO, visitedPOIs, interestedPOIs, responseDictionary), createUserDTO.nickname);
}
```

Figura 85 – Modelação dos *UserAgents* com os *ratings* recebidos

5.1.6 Recalcular o processo de *clustering*

À medida que novos utilizadores se registam na aplicação e preenchem o questionário de personalidade, a média da personalidade dos vários clusters existentes na aplicação é atualizada e o cluster que outrora era o mais indicado para um dado utilizador, pode já não o ser mais. Um *BackGroundService* é executado em simultâneo ao arranque do MAMS Service e efetua chamadas ao método *recalculation* do *UpdateUserProfileService* de 48 em 48 horas.

O método *recalculation* começa por preencher a lista *allUsers* e a lista *allClusters* com todos os *Users* e *Clusters* da aplicação, respetivamente. Para isso, serve-se dos métodos *GetAllUsers* e *GetAllClusters* disponibilizados pela camada *Repository* (Figura 86).

```
List<User> alllUsers = await this.userRepository.GetAllUsers();

List<Cluster> allClusters = await this.clusterRepository.GetAllCluster();
```

Figura 86 – Obtenção de todos os *Users* e *Clusters* da base de dados do MAMS Service

Para todos os *Users* presentes em *allUsers*, compara-se a sua personalidade com a personalidade média dos utilizadores de cada *Cluster* existente na lista *allClusters* (Figura 87). Tudo isto acontece dentro de um ciclo *do-while* controlado pela variável *personalityAverageHasChanged* cujo valor *default* (*false*) sinaliza que nenhum utilizador atualizou o seu cluster.

```
do
{
    personalityAverageHasChanged = false;

    foreach (User u in alllUsers)
    {


        Cluster bestCluster = null;
        double bestSimilarity = Double.MinValue;
        int clusterCounter = 0;

        Personality personalityUser = new Personality
        {
            Openness = u.openness,
            Conscientiousness = u.conscientiousness,
            Extraversion = u.extraversion,
            Agreeableness = u.agreeableness,
            Neuroticism = u.neuroticism
        };


        foreach (Cluster c in allClusters)
        {
```

Figura 87 – Análise de todos os *Clusters* existentes em *allClusters* para cada *User* existente em *allUsers*

A comparação das personalidades é efetuada através do método *NormalizePersonalityEuclideanDistance* da classe *Distance* que utiliza a fórmula da distância euclidiana. A similaridade resultante é depois tratada de igual forma ao que já foi explicado no algoritmo de *clustering* da secção 5.1.2. A única diferença é que se recorre ao método *UpdateAsync* da classe *UserManager* para atualizar o *Cluster* associado ao *User* em análise e que se atualiza o valor da variável *personalityAverageHasChanged* para *true* de modo a sinalizar que uma alteração de cluster ocorreu (Figura 88).

```
    euclideanDistanceSimilarity = Distance.NormalizePersonalityEuclideanDistance(personalityUser, averagePersonality);

    if (bestSimilarity < euclideanDistanceSimilarity && euclideanDistanceSimilarity >= MIN_SIMILARITY)
    {
        bestSimilarity = euclideanDistanceSimilarity;
        bestCluster = c;
    }


} //for each cluster


//Adds user to closest cluster if better than the actual cluster or creates new cluster if no cluster is appropriate to the user
if (bestCluster != null)
{

    u.cluster = bestCluster;
    personalityAverageHasChanged = true;

    await userManager.UpdateAsync(u);
}
else if (bestCluster == null)
{
    Cluster newCluster = new Cluster
    {
        centroidUserName = u.UserName,
        clusterId = Guid.NewGuid().ToString()
    };

    u.cluster = newCluster;
    personalityAverageHasChanged = true;

    await userManager.UpdateAsync(u);
}
```

Figura 88 – Atualização do *Cluster* de um dado *User* caso seja encontrado um melhor que o atual

5.2 Testes

O processo de testagem de software visa diminuir significativamente as anomalias existentes num produto de software através da deteção do maior número possível de falhas no código desenvolvido. Existe uma grande variedade de testes capazes de verificar a confiabilidade e previsibilidade do software [87]. Os tipos de testes escolhidos para analisar os desenvolvimentos efetuados no protótipo de recomendação de POI para grupos são detalhados nesta secção.

5.2.1 Testes de integração

Testes de integração visam aferir o correto funcionamento dos módulos de software individuais quando cooperam entre si para responder a um determinado requisito [88].

Para implementar os testes desta tipologia, utilizou-se a plataforma Postman. Esta plataforma disponibiliza uma interface gráfica que possibilita efetuar pedidos HTTP REST à API Gateway do protótipo de recomendação de POI para grupos. Os testes desenvolvidos permitem avaliar o código da resposta HTTP de cada pedido ou aferir o tempo de resposta. A análise do conteúdo da resposta não é tida em conta uma vez que, para as principais funcionalidades, nomeadamente os pedidos de recomendação de POI individuais e para grupo, este conteúdo pode sofrer alterações fruto de novo feedback sobre POI introduzido na aplicação. A Figura 89 representa um exemplo de um teste de integração desenvolvido no Postman.

Figura 89 – Teste de Integração desenvolvido na plataforma Postman

5.2.2 Testes de carga

A presente tipologia de testes simula um contexto real para identificar a máxima capacidade de processamento de informação de um produto de software. Um plano de testes foi desenvolvido através da plataforma JMeter para aferir se o protótipo está preparado para, num curto intervalo de tempo, receber múltiplos pedidos de recomendação de POI. A Figura 90 mostra os resultados obtidos num teste de carga que utilizou 50 *threads* para simular 50 utilizadores a efetuar pedidos de recomendação de POI individuais com um intervalo máximo de 5 segundos entre cada uma. O protótipo aparenta estar preparado para responder a uma situação semelhante visto que todos os pedidos obtiveram um *status* de sucesso.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	18:10:56.334	Thread Group 1-2	HTTP Request	763		3183	384	763	155
10	18:10:57.593	Thread Group 1-11	HTTP Request	762		3183	384	762	143
11	18:10:57.734	Thread Group 1-12	HTTP Request	715		3165	384	715	147
12	18:10:57.874	Thread Group 1-13	HTTP Request	765		3165	384	765	142
13	18:10:58.013	Thread Group 1-14	HTTP Request	750		3183	384	750	152
14	18:10:58.154	Thread Group 1-15	HTTP Request	701		3183	384	701	155
15	18:10:58.293	Thread Group 1-16	HTTP Request	633		3165	384	633	144
16	18:10:58.434	Thread Group 1-17	HTTP Request	668		3165	384	668	145
17	18:10:58.574	Thread Group 1-18	HTTP Request	672		3183	384	672	149
18	18:10:58.714	Thread Group 1-19	HTTP Request	734		3165	384	734	146
19	18:10:58.855	Thread Group 1-20	HTTP Request	705		3165	384	705	144
2	18:10:56.473	Thread Group 1-3	HTTP Request	660		3183	384	660	155
20	18:10:58.994	Thread Group 1-21	HTTP Request	699		3183	384	699	152
21	18:10:59.133	Thread Group 1-22	HTTP Request	726		3183	384	726	146
22	18:10:59.273	Thread Group 1-23	HTTP Request	730		3165	384	730	143
23	18:10:59.413	Thread Group 1-24	HTTP Request	650		3165	384	650	144
24	18:10:59.553	Thread Group 1-25	HTTP Request	629		3183	384	628	159
25	18:10:59.693	Thread Group 1-26	HTTP Request	664		3183	384	664	145
26	18:10:59.834	Thread Group 1-27	HTTP Request	645		3165	384	645	142
27	18:10:59.974	Thread Group 1-28	HTTP Request	827		3165	384	827	159
28	18:11:00.114	Thread Group 1-29	HTTP Request	732		3165	384	732	152
29	18:11:00.254	Thread Group 1-30	HTTP Request	710		3165	384	710	140
3	18:10:56.614	Thread Group 1-4	HTTP Request	797		3165	384	797	143
30	18:11:00.394	Thread Group 1-31	HTTP Request	696		3165	384	696	140
31	18:11:00.535	Thread Group 1-32	HTTP Request	712		3165	384	712	137
32	18:11:00.674	Thread Group 1-33	HTTP Request	696		3165	384	696	143

Figura 90 – Teste de carga desenvolvido na plataforma JMeter

6 Experimentação e avaliação da solução

O presente capítulo procura avaliar a qualidade do protótipo de sistema de recomendação de POI melhorado. Inicialmente, definem-se as hipóteses de investigação a comprovar e descreve-se a metodologia de avaliação utilizada. Por fim, é feita a análise dos resultados.

6.1 Especificação das hipóteses de investigação

A avaliação do MAMS a melhorar pode ser dividida em duas vertentes distintas:

- Avaliação do correto funcionamento do sistema e do seu desempenho: Garantir que um produto de software é analisado de forma a detetar e corrigir possíveis falhas é crucial. Torna-se necessário verificar a boa comunicação entre os vários componentes do sistema e mesurar o tempo necessário para a conclusão de um pedido de recomendação de POI.
- Avaliação da satisfação do utilizador: Garantir que o utilizador está satisfeito com a utilização da aplicação é provavelmente o fator mais importante a considerar. A opinião de cada utilizador com as recomendações que lhe são atribuídas é fundamental para que se identifique problemas e outros aspetos a melhorar com base na experiência de utilizadores reais.

De modo a obter conclusões, utilizaram-se testes de hipótese de modo a efetuar uma comparação entre a hipótese nula (H0) e uma ou mais hipóteses para rejeitar ou não rejeitar H0.

Relativamente à vertente "Avaliação do correto funcionamento do sistema e do seu desempenho" estabeleceram-se dois critérios distintos a ser testados:

Critério 1: Avaliação do correto funcionamento do sistema

- H0: A execução dos testes de software não apresenta qualquer falha;
- H1: A execução dos testes de software apresenta uma ou mais falhas;

Critério 2: Avaliação do desempenho do sistema

- H0: O sistema demora menos de 10 segundos a responder a pedidos de recomendação de POI para grupos;
- H1: O sistema demora mais de 10 segundos a responder a pedidos de recomendação de POI para grupos;

Relativamente à vertente "Avaliação da satisfação do utilizador" definiu-se o seguinte critério a ser testado:

Critério 3: Avaliação da satisfação do utilizador

- H0: Em média, as respostas a um inquérito avaliam positivamente o sistema.
- H1: Em média, as respostas a um inquérito avaliam negativamente o sistema.

6.2 Descrição da metodologia de avaliação

Estipularam-se metodologias distintas para avaliar as hipóteses cujos critérios se inserem em vertentes heterogéneas. Para avaliar o Critério 1 recorre-se à utilização de vários tipos de testes de software, nomeadamente, testes de integração do MAMS com o protótipo do GRS de modo a verificar a boa comunicação entre os vários microserviços e testes de carga para aferir a capacidade de resposta da aplicação a pedidos de recomendação de vários utilizadores em simultâneo.

Relativamente ao Critério 2, efetuam-se 20 pedidos de recomendação de POI para um mesmo grupo de excursão, registando a duração de cada pedido numa folha de Excel de forma a obter a média dos resultados para posterior análise. A média é o fator a considerar para procurar rejeitar H0. O tempo de 10 segundos foi baseado nas várias comunicações que existem entre os microserviços da aplicação durante um pedido de recomendação de POI.

No que diz respeito ao último critério, Critério 3, grupos de utilizadores utilizam o GRS para efetuar pedidos de recomendação individuais e de grupo para que possam fornecer a sua opinião através de inquéritos. Os inquéritos em questão são constituídos por várias perguntas que avaliam aspetos relacionados com a utilidade do GRS. Cada pergunta possui respostas que seguem uma escala de valores de zero a quatro, onde zero corresponde a “Discordo Totalmente” e quatro a “Concordo Totalmente”.

6.3 Análise de resultados

Nesta secção realiza-se uma análise aos resultados obtidos para cada um dos critérios estipulados na secção 6.2.

6.3.1 Análise do Critério 1

Para avaliar este critério recorreu-se à execução dos vários testes já mencionados na secção 5.2.

O caso de teste “Efetuar um pedido de recomendação de POI para um grupo cujos membros estão registados no protótipo” visa verificar se o processo de recomendação para o grupo é efetuado com sucesso (Tabela 17).

Tabela 17- Caso de Teste: Efetuar um pedido de recomendação de POI para um grupo cujos membros estão registados no protótipo

Caso de Teste: Efetuar um pedido de recomendação de POI para um grupo cujos membros

estão registados no protótipo.	
Resultado Previsto	**Resultado obtido**
O processo de recomendação para o grupo é efetuado com sucesso.	O processo de recomendação para o grupo é efetuado com sucesso.

O caso de teste "Efetuar um pedido de recomendação de POI para um grupo cujos membros não estão todos registados no protótipo" visa verificar se o processo de recomendação para o grupo é efetuado com sucesso (Tabela 18).

Tabela 18 - Caso de Teste: Efetuar um pedido de recomendação de POI para um grupo cujos membros não estão todos registados no protótipo.

Caso de Teste: Efetuar um pedido de recomendação de POI para um grupo cujos membros não estão todos registados no protótipo.	
Resultado Previsto	**Resultado obtido**
O processo de recomendação para o grupo não é efetuado com sucesso.	O processo de recomendação para o grupo não é efetuado com sucesso.

O caso de teste "Efetuar 50 pedidos de recomendação de POI individuais" visa verificar se o processo de recomendação é efetuado com sucesso em todas as 50 *threads* que simulam os utilizadores (Tabela 19).

Tabela 19 – Caso de Teste: Efetuar 50 pedidos de recomendação de POI individuais

Caso de Teste: Efetuar 50 pedidos de recomendação de POI individuais	
Resultado Previsto	**Resultado obtido**
Os 50 pedidos de recomendação são efetuados com sucesso.	Os 50 pedidos de recomendação são efetuados com sucesso.

O caso de teste "Efetuar pedido de recomendação de POI individual antes e depois de fornecer *feedback*" visa verificar se caso um utilizador informe que não pretende revisitar um dado POI que lhe apareceu como recomendação num primeiro pedido de recomendação individual, esse mesmo POI não aparece no segundo pedido de recomendação (Tabela 20).

Tabela 20 – Caso Teste: Efetuar pedido de recomendação de POI individual antes e depois de fornecer *feedback*

Caso de Teste: Efetuar pedido de recomendação de POI individual antes e depois de fornecer *feedback*	
Resultado Previsto	**Resultado obtido**
O POI especificado para não voltar a visitar, não consta na lista de recomendação obtida.	O POI especificado para não voltar a visitar, não consta na lista de recomendação obtida.

Todos os casos de teste obtiveram o resultado esperado, o que demonstra que o protótipo cumpre com a hipótese definida.

6.3.2 Análise do Critério 2

Para executar a metodologia definida para a análise do critério 2, utilizou-se um computador ligado a uma rede WiFi com 66.9 Mbps de transferência e 20.7 Mbps de carregamento. A Tabela 21 representa uma análise dos resultados após a execução da metodologia.

Tabela 21 – Análise ao tempo de resposta de um pedido de recomendação de POI para grupo

Maior tempo (s)	Menor tempo (s)	Tempo médio (s)
7.68	6.87	7.21

O tempo médio de resposta é de 7.21 segundos, logo, o sistema cumpre a hipótese estipulada ao respeitar o tempo limite de 10 segundos.

6.3.3 Análise do Critério 3

A empresa Sistrade possui em desenvolvimento uma aplicação web que efetua pedidos à API Gateway do protótipo de recomendação de POI para grupos. Foram recrutadas 13 pessoas para validar a integração dos dois protótipos em duas fases distintas.

Fase 1

1. Dar 5 min a cada pessoa para num computador ou telemóvel procurar as atividades que gostaria de fazer no Porto;
2. Utilizar a *app* da Sistrade para efetuar um pedido de recomendação de POI individual;
3. Preencher o questionário de validação, comparando os dois métodos.

Fase 2

1. Em grupo (as 13 pessoas), pedir para se organizarem e decidirem uma lista de atividades a fazer;
2. Utilizar a *app* da Sistrade para efetuar um pedido de recomendação de POI para grupo;
3. Preencher o questionário de validação, comparando os dois métodos.

A Tabela 22 mostra o número de respostas para cada opção, na escala definida na secção 6.2.

Tabela 22 – Análise dos questionários de validação do sistema

Questão	0	1	2	3	4

Foi mais fácil procurar manualmente os locais a visitar do que usar a aplicação para isso?	1	10	2	0	0
Foi mais rápido procurar manualmente os locais a visitar do que usar a aplicação para isso?	2	6	4	1	0
Gostei mais de usar a aplicação para obter recomendações do que procurar manualmente os locais?	0	0	2	7	4
As recomendações sugeridas pela aplicação foram do meu agrado?	0	1	3	5	4
Consegui encontrar os 10 locais dentro do tempo permitido?	5	3	1	1	3
Foi mais fácil procurar manualmente os locais a visitar do que usar a aplicação para isso? (Grupo)	1	6	3	1	2
Foi mais rápido procurar manualmente os locais a visitar do que usar a aplicação para isso? (Grupo)	1	7	3	1	1
Gostei mais de usar a aplicação para obter recomendações do que procurar manualmente os locais? (Grupo)	1	1	3	4	4
As recomendações sugeridas pela aplicação foram do meu agrado? (Grupo)	0	1	2	8	2
Consegui encontrar os 10 locais dentro do tempo permitido? (Grupo)	0	0	1	4	8

Considerando as hipóteses formuladas para este critério e analisando a Tabela 22, é possível afirmar que existe uma satisfação geral com o sistema.

´

7 Conclusão

Neste capítulo realiza-se uma avaliação ao trabalho desenvolvido, mediante a apresentação dos objetivos atingidos. Descreve-se ainda as direções a seguir num desenvolvimento futuro assim como algumas limitações encontradas. Por fim, é feita uma apreciação pessoal acerca do trabalho realizado.

7.1 Objetivos atingidos

Este capítulo apresenta o grau de concretização de cada objetivo estabelecido na secção 1.3. A Tabela 23 mostra a percentagem de realização desses objetivos.

Tabela 23 – Grau de concretização dos objetivos estabelecidos na secção 1.3

Objetivo	**Grau de concretização**
Objetivo 1	100%
Objetivo 2	100%
Objetivo 3	90%
Objetivo 4	100%

O objetivo 3 não possui um grau de concretização de 100% porque ficou por implementar um mecanismo baseado num algoritmo *apriori* que gerasse rules com base numa análise efetuada ao feedback e ao perfil dos membros de cada cluster. Por exemplo, se 90% dos utilizadores que possuem filhos de um dado cluster atribuíram um *rating* de 1 numa escala de 1 a 5 ao POI Torre dos Clérigos, significa que uma rule negativa associada a esse POI deve ser criada nesse cluster.

Desta forma, se pelo menos metade dos membros de um subgrupo originado através desse cluster possuir filhos, o POI Torre dos Clérigos é associado à lista de POI a penalizar independentemente da maioria do cluster ter atribuído um *rating* superior a 3 numa escala de 1 a 5 a esse POI. Esta lógica está implementada e foi testada através da criação manual de rules apesar da geração automática de rules não ter sido implementada.

7.2 Limitações e trabalho futuro

Um dos principais desafios encontrados foi aprender a trabalhar com a *framework* ActressMAS no qual o MAS do MAMS estava desenvolvido. Embora possua uma simplicidade característica, não existem muitos exemplos da sua utilização disponíveis. Outra dificuldade, foi conseguir perceber a lógica que já tinha sido implementada no MAS e compreender como seria possível introduzir as novas funcionalidades sem provocar conflitos.

Uma vez tratar-se de um protótipo em desenvolvimento no âmbito da investigação, novos requisitos surgiram e outros acabaram por ser atualizados, o que levou a que se refizessem funcionalidades com uma lógica totalmente diferente à que tinha sido inicialmente estipulada.

Relativamente ao trabalho futuro, passaria pela implementação do mecanismo que utiliza um algoritmo *apriori* para gerar as rules que ainda não são geradas automaticamente tal como foi explicado na secção 7.1. Outro aspeto a ter em conta, seria a aplicação de alguns padrões comuns à arquitetura de microserviços que lidam com a manutenção da consistência dos dados.

7.3 Apreciação final

A possibilidade de utilizar novas tecnologias no desenvolvimento de um projeto cujos conceitos eram igualmente novos para mim foi desafiante e simultaneamente motivador. O GECAD permitiu que tivesse uma experiência no ramo da investigação diferente de tudo o que já tinha experienciado e sempre se mostrou prestável em me ajudar com qualquer dificuldade que surgisse ao longo do desenvolvimento do protótipo de recomendação de POI para grupos.

É gratificante saber que os meus desenvolvimentos contribuíram para a escrita de dois artigos científicos, nomeadamente o artigo "Improving Group Recommendations using Personality, Dynamic Clustering and Multi-Agent MicroServices", submetido para a conferência internacional ACM RecSys 2023 (CORE2021: A), e o artigo, que se encontra atualmente em revisão, "Grouplanner, a Group Recommender System for Tourism: are Heterogeinity and Conflicting Preferences no Longer a Problem?", submetido para a revista Applied Soft Computing (JCR IF 8.263).

Em suma, estou feliz com os resultados obtidos e com os novos conhecimentos que consegui adquirir ao longo do projeto.

Referências

[1] B. Hui, L. Zhang, X. Zhou, X. Wen, and Y. Nian, "Personalized recommendation system based on knowledge embedding and historical behavior", *Applied Intelligence*, vol. 52, no. 1, pp. 954–966, May 2021, doi: 10.1007/s10489-021-02363-w.

[2] J. Borràs, A. Moreno, and A. Valls, "Intelligent tourism recommender systems: A survey", *Expert Systems with Applications*, vol. 41, no. 16, pp. 7370–7389, Nov. 2014, doi: 10.1016/j.eswa.2014.06.007.

[3] R. Collier, E. O'neill, D. Lillis, and G. O'hare, "MAMS: Multi-Agent MicroServices", *WWW '19: Companion Proceedings of The 2019 World Wide Web Conference,* New York, USA, May, 2019, doi: 10.1145/3308560.3316509.

[4] M. Batet, A. Moreno, D. Sánchez, D. Isern, and A. Valls, "Turist@: Agent-based personalised recommendation of tourist activities", *Expert Systems with Applications*, vol. 39, no. 8, pp. 7319–7329, Jun. 2012, doi: 10.1016/j.eswa.2012.01.086.

[5] P. Alves, J. Carneiro, G. Marreiros, and P. Novais, "Modeling a Mobile Group Recommender System for Tourism with Intelligent Agents and Gamification", International Conference on Hybrid Artificial Intelligence Systems, Springer, Cham, 2019.

[6] P. Alves *et al.*, "Modeling Tourists' Personality in Recommender Systems", *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, Jul. 2020, doi: 10.1145/3340631.3394843.

[7] P. Alves, D. Gomes, C. Rodrigues, and J. Carneiro, "Grouplanner: a Group Recommender System for Tourism with Multi-Agent MicroServices," Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation - The PAAMS Collection, Springer, Cham, 2022.

[8] K. Schwaber and J. Sutherland, "The Scrum Guide TM The Definitive Guide to Scrum: The Rules of the Game", 2016. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf

[9] R. Popli and N. Chauhan, "SCRUM: AN AGILE FRAMEWORK", *International Journal of Information Technology*, vol. 4, no. 1, pp. 147–149, 2011, [Online]. Available: http://www.csjournals.com/IJITKM/PDF%204-1/30.Rashmi%20Popli1%20%26%20Naresh%20Chauhan2.pdf

[10] Atlassian, "Bitbucket," *Bitbucket*, 2020. https://bitbucket.org/

[11] F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh, "Recommendation systems: Principles, methods and evaluation", *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261–273, Nov. 2015, doi: 10.1016/j.eij.2015.06.005.

[12] M. Kunaver and T. Požrl, “Diversity in recommender systems – A survey”, *Knowledge-Based Systems*, vol. 123, pp. 154–162, May 2017, doi: 10.1016/j.knosys.2017.02.009.

[13] D. Roy and M. Dutta, “A systematic review and research perspective on recommender systems”, *Journal of Big Data*, vol. 9, no. 1, May 2022, doi: 10.1186/s40537-022-00592-5.

[14] Z. Fayyaz, M. Ebrahimian, D. Nawara, A. Ibrahim, and R. Kashef, “Recommendation Systems: Algorithms, Challenges, Metrics, and Business Opportunities”, *Applied Sciences*, vol. 10, no. 21, p. 7748, Nov. 2020, doi: 10.3390/app10217748.

[15] P. H. Aditya, I. Budi, and Q. Munajat, “A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X”, *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, Oct. 2016, doi: 10.1109/icacsis.2016.7872755.

[16] S. Dhelim, N. Aung, M. A. Bouras, H. Ning, and E. Cambria, “A survey on personality-aware recommendation systems”, *Artificial Intelligence Review*, vol. 55, no. 3, pp. 2409–2454, Sep. 2021, doi: 10.1007/s10462-021-10063-7.

[17] S. Dhelim, H. Ning, N. Aung, R. Huang, and J. Ma, “Personality-Aware Product Recommendation System Based on User Interests Mining and Metapath Discovery”, *IEEE Transactions on Computational Social Systems*, vol. 8, no. 1, pp. 86–98, Feb. 2021, doi: 10.1109/tcss.2020.3037040.

[18] M. Braunhofer, M. Elahi, and F. Ricci, “User Personality and the New User Problem in a Context-Aware Point of Interest Recommender System”, *Information and Communication Technologies in Tourism 2015*, pp. 537–549, Dec. 2014, doi: 10.1007/978-3-319-14343-9_39.

[19] J. Masthoff, “Group Recommender Systems: Combining Individual Models”, *Recommender Systems Handbook*, pp. 677–702, Oct. 2010, doi: 10.1007/978-0-387-85820-3_21.

[20] T. N. Nguyen and F. Ricci, “A chat-based group recommender system for tourism”, *Information Technology & Tourism*, vol. 18, no. 1–4, pp. 5–28, Dec. 2017, doi: 10.1007/s40558-017-0099-y.

[21] M. Furmankiewicz, A. Sołtysik-Piorunkiewicz and P. Ziuziański, “Artificial Intelligence Systems for Knowledge Management in e-Health: The Study of Intelligent Software Agents”, *Latest Trends on Systems*, 2014.

[22] Bo Chen and H. H. Cheng, “A Review of the Applications of Agent Technology in Traffic and Transportation Systems,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 485–497, Jun. 2010, doi: 10.1109/tits.2010.2048313.

[23] A. Dorri, S. S. Kanhere, and R. Jurdak, “Multi-Agent Systems: A Survey”, *IEEE Access*, vol. 6, pp. 28573–28593, 2018, doi: 10.1109/access.2018.2831228.

[24] A. Amirkhani and A. H. Barshooi, “Consensus in multi-agent systems: a review”, *Artificial Intelligence Review*, vol. 55, no. 5, pp. 3897–3935, Nov. 2021, doi: 10.1007/s10462-021-10097-x.

[25] A. Garro *et al.*, “Intelligent Agents: Multi-Agent Systems”, *Encyclopedia of Bioinformatics and Computational Biology*, pp. 315–320, 2019, doi: 10.1016/b978-0-12-809633-8.20328-2.

[26] V. Julian and V. Botti, “Multi-Agent Systems”, *Applied Sciences*, vol. 9, no. 7, p. 1402, Apr. 2019, doi: 10.3390/app9071402.

[27] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers, “Evolutionary Dynamics of Multi-Agent Learning: A Survey”, *Journal of Artificial Intelligence Research*, vol. 53, pp. 659–697, Aug. 2015, doi: 10.1613/jair.4818.

[28] J. Thones, “Microservices”, *IEEE Software*, vol. 32, no. 1, pp. 116–116, Jan. 2015, doi: https://doi.org/10.1109/ms.2015.11.

[29] A. Singleton, “The Economics of Microservices”, *IEEE Cloud Computing*, vol. 3, no. 5, pp. 16–20, Sep. 2016, doi: https://doi.org/10.1109/mcc.2016.109.

[30] C. Richardson, “Microservices.io,” *microservices.io*, 2017. https://microservices.io/

[31] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, “Microservices: The Journey So Far and Challenges Ahead”, *IEEE Software*, vol. 35, no. 3, pp. 24–35, May 2018, doi: https://doi.org/10.1109/ms.2018.2141039.

[32] K. Kravari and N. Bassiliades, “StoRM: A social agent-based trust model for the internet of things adopting microservice architecture”, *Simulation Modelling Practice and Theory*, vol. 94, pp. 286–302, Jul. 2019, doi: https://doi.org/10.1016/j.simpat.2019.03.008.

[33] Microsoft, “A tour of C# - Overview”, *learn.microsoft.com*, Dec. 13, 2022. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[34] S. Nakov *et al.*, “Fundamentals of Computer Programming with C# (The Bulgarian C# Programming Book)”, 2013. Available: https://www.introprogramming.info/wp-content/uploads/2013/07/Books/CSharpEn/Fundamentals-of-Computer-Programming-with-CSharp-Nakov-eBook-v2013.pdf

[35] Microsoft, “What is .NET? An open-source developer platform.,” *Microsoft*. https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

[36] Microsoft, “ASP.NET | Open-source web framework for .NET”, *Microsoft*. https://dotnet.microsoft.com/en-us/apps/aspnet

[37] Microsoft, “Create web APIs with ASP.NET Core”, *learn.microsoft.com,* Jan. 18, 2023. https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-5.0.

[38] J. Eguíluz Pérez, “Introducción a JavaScript”, e-REdING, Mar. 2009. Available: https://biblus.us.es/bibing/proyectos/abreproy/12051/fichero/libros%252Flibro-javascript.pdf

[39] S. H. Jensen, A. Møller, and P. Thiemann, “Type Analysis for JavaScript”, *Static Analysis*, pp. 238–255, 2009, doi: https://doi.org/10.1007/978-3-642-03237-0_17.

[40] A.K Hota and Madan Prabhu, “E-Gov Products and Services Technology Update WHAT IS NODE.JS?,” Jan. 2014. Available: https://informatics.nic.in/uploads/pdfs/26b47a73_node.js.pdf

[41] V. Bojinov, *RESTful Web API Design with Node.js 10, Third Edition : Learn to create robust RESTful web services with Node.js, MongoDB, and Express.js, 3rd Edition*. Birmingham: Packt Publishing, 2018.

[42] P. Coelho, *Programação em JAVA, Curso Completo*, 5th ed. FCA, 2016.

[43] M. Gajewski and W. Zabierowski, “Analysis and Comparison of the Spring Framework and Play Framework Performance, Used to Create Web Applications in Java”, *IEEE Xplore*, May 01, 2019. https://ieeexplore.ieee.org/document/8817390 (accessed Apr. 03, 2022).

[44] P. Webb *et al.*, “Spring Boot Reference Documentation”, Jun. 2020. Available: https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/pdf/spring-boot-reference.pdf

[45] E. Haro, T. Guarda, A. Zambrano Peñaherrera, and G. Ninahualpa Quiña, “Desarrollo backend para aplicaciones web, Servicios Web Restful: Node.js vs Spring Boot”, *Revista Ibérica de Sistemas e Tecnologias de Informação*, pp. 309, 321, Jan. 2019.

[46] X. Mao, “Comparison between symfony, asp. net mvc, and node. js express for web development”, Apr. 2018.

[47] H. K. Dhalla, “A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core”, *Journal of Physics: Conference Series*, vol. 1933, no. 1, p. 012041, Jun. 2021, doi: https://doi.org/10.1088/1742-6596/1933/1/012041.

[48] A. Dalbard and J. Isacson, “Comparative study on performance between ASP.NET and Node.js Express for web-based calculation tools”, Jun. 2021. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1572101&dswid=9199

[49] F. Leon, “ActressMAS, a .NET Multi-Agent Framework Inspired by the Actor Model”, *Mathematics*, vol. 10, no. 3, p. 382, Jan. 2022, doi: https://doi.org/10.3390/math10030382.

[50] F. Bergenti, G. Caire, S. Monica, and A. Poggi, “The first twenty years of agent-based software development with JADE”, *Autonomous Agents and Multi-Agent Systems*, vol. 34, no. 2, May 2020, doi: https://doi.org/10.1007/s10458-020-09460-z.

[51] G. Caire and F. Pieri, “LEAP User Guide”, Nov. 2011.

[52] P. F. Oliveira, P. Novais, and P. Matos, “Using Jason Framework to Develop a Multi-agent System to Manage Users and Spaces in an Adaptive Environment System”, *Advances in Intelligent Systems and Computing*, pp. 137–145, Sep. 2020, doi: https://doi.org/10.1007/978-3-030-58356-9_14.

[53] C. Pantoja, V. Souza De Jesus, and J. Viterbo, “Aplicando Sistemas Multi-Agentes Ubıquos em um Modelo de Smart Home Usando o Framework Jason”, II Workshop de Pesquisa e Desenvolvimento em Inteligência Artificial, Dec. 2016.

[54] H. V. V. Priyadarshana, K. T. M. U Hemapala, W. D. A. S Wijayapala, V. Saravanan, and M. A. Kalhan S. Boralessa, “Developing Multi-Agent Based Micro-Grid Management System in JADE”, *IEEE Xplore*, Aug. 01, 2019. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9036562&tag=1

[55] C.-V. Pal, F. Leon, M. Paprzycki, and M. Ganzha, “A Review of Platforms for the Development of Agent Systems”, 2020.

[56] L. Sebastiá Tarín, A. S. Giret Boggino, and I. García García, “A Multi Agent Architecture for Single User and Group Recommendation in the Tourism Domain”, *riunet.upv.es*, 2011. https://riunet.upv.es/handle/10251/35364

[57] L. Ravi *et al.*, “An intelligent location recommender system utilising multi-agent induced cognitive behavioural model”, *Enterprise Information Systems*, pp. 1–19, Sep. 2020, doi: https://doi.org/10.1080/17517575.2020.1812003.

[58] R. Hassannia, A. Vatankhah Barenji, Z. Li, and H. Alipour, “Web-Based Recommendation System for Smart Tourism: Multiagent Technology”, *Sustainability*, vol. 11, no. 2, p. 323, Jan. 2019, doi: https://doi.org/10.3390/su11020323.

[59] F. Lorenzi, S. Loh, and M. Abel, “PersonalTour: A Recommender System for Travel Packages”, *IEEE Xplore*, Aug. 01, 2011. https://ieeexplore.ieee.org/document/6040800

[60] G. HUGHES, “Turning new product development into a continuous learning process”, *Journal of Product Innovation Management*, vol. 13, no. 2, pp. 89–104, Mar. 1996, doi: https://doi.org/10.1016/0737-6782(95)00112-3.

[61] P. Belliveau, A. Griffin, S. Somermeyer, and Product Development & Management Association, *The PDMA toolbook for new product development*. New York: John Wiley & Sons, Inc, 2002.

[62] D. Newman, “The Design Squiggle”, *thedesignsquiggle.com*. https://thedesignsquiggle.com/

[63] F. Almqvist, “The fuzzy front-end and the forgotten back-end: User involvement in later development phases”, *The Design Journal*, vol. 20, no. sup1, pp. S2524–S2533, Jul. 2017, doi: https://doi.org/10.1080/14606925.2017.1352765.

[64] UNWTO, “Tourism Set to Return to Pre-Pandemic Levels in Some Regions in 2023”, *www.unwto.org*, Jan. 17, 2023. https://www.unwto.org/taxonomy/term/347

[65] R. de F. S. M. Russo and R. Camanho, “Criteria in AHP: A Systematic Review of Literature”, *Procedia Computer Science*, vol. 55, pp. 1123–1132, 2015, doi: https://doi.org/10.1016/j.procs.2015.07.081.

[66] E. A. Nantes, “Método analytic hierarchy process para la toma de decisiones : repaso de la metodología y aplicaciones”, *repositoriodigital.uns.edu.ar*, Nov. 2019, Available: https://repositoriodigital.uns.edu.ar/handle/123456789/6060

[67] A. Graf and P. Maas, “Customer value from a customer perspective: a comprehensive review”, *Journal für Betriebswirtschaft*, vol. 58, no. 1, pp. 1–20, Mar. 2008, doi: https://doi.org/10.1007/s11301-008-0032-8.

[68] A. Osterwalder and Y. Pigneur, “Modeling value propositions in e-Business”, *Proceedings of the 5th international conference on Electronic commerce - ICEC ’03*, 2003, doi: https://doi.org/10.1145/948005.948061.

[69] Entity Framework Tutorial, “What is Entity Framework?”, *Entityframeworktutorial.net*, 2013. https://www.entityframeworktutorial.net/what-is-entityframework.aspx

[70] The PostgreSQL Global Development Group, “PostgreSQL: The world’s most advanced open source database”, *Postgresql.org*, 2019. https://www.postgresql.org/

[71] ElephantSQL, “ElephantSQL - PostgreSQL as a Service”, *www.elephantsql.com*. https://www.elephantsql.com/

[72] L. A. Macaulay, *Requirements Engineering*. Springer Science & Business Media, 2012.

[73] J. P. Mighetti, G. D. S. Hadad, J. P. Mighetti, and G. D. S. Hadad, “A Requirements Engineering Process Adapted to Global Software Development”, *CLEI Electronic Journal*, vol. 19, no. 3, pp. 181–209, Dec. 2016, Available: http://www.scielo.edu.uy/scielo.php?script=sci_arttext&pid=S0717-50002016000300181

[74] R. Malan, H.-P. Company, and D. Bredemeyer, “Functional Requirements and Use Cases Functional Requirements”, 2001. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=aceaa41855c38aebe7c823e60e94b39506a92b99

[75] J. Eckhardt, A. Vogelsang, and D. M. Fernández, “Are ‘non-functional’ requirements really non-functional?”, *Proceedings of the 38th International Conference on Software Engineering - ICSE ’16*, 2016, doi: https://doi.org/10.1145/2884781.2884788.

[76] D. Mairiza, D. Zowghi, and N. Nurmuliani, “An investigation into the notion of non-functional requirements”, *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC ’10*, 2010, doi: https://doi.org/10.1145/1774088.1774153.

[77] C. Sequeira, “Especificação rigorosa de requisitos de usabilidade para software”, 2017.

[78] P. Kruchten, “Architecture blueprints---the ‘4+1’ view model of software architecture”, *Tutorial proceedings on TRI-Ada ’91 Ada’s role in global markets: solutions for a changing complex world - TRI-Ada ’95*, 1995, doi: https://doi.org/10.1145/216591.216611.

[79] S. Brown, “The C4 Model for Software Architecture,” *InfoQ*, 2018. https://www.infoq.com/articles/C4-architecture-model/

[80] C. Zhang and D. Budgen, “What Do We Know about the Effectiveness of Software Design Patterns?”, *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1213–1231, Sep. 2012, doi: https://doi.org/10.1109/tse.2011.79.

[81] M. Fowler, “Patterns”, *IEEE Software*, vol. 20, no. 2, pp. 56–57, Mar. 2003, doi: https://doi.org/10.1109/ms.2003.1184168.

[82] Visual Paradigm, “What is Model-View and Control?”, *www.visual-paradigm.com*. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-model-view-control-mvc/

[83] M. Fowler, “P of EAA: Data Transfer Object”, *martinfowler.com*. https://martinfowler.com/eaaCatalog/dataTransferObject.html
Fowler

[84] M. Prajapati, “International Journal of All Research Writings 23 ASP.NET MVC -GENERIC REPOSITORY PATTERN AND UNIT OF WORK”, vol. 1, no. 1, 2582, Available: http://www.ijarw.com/PublishedPaper/IJARW1005.pdf

[85] M. Fowler, “P of EAA: Service Layer”, *martinfowler.com*. https://martinfowler.com/eaaCatalog/serviceLayer.html

[86] C. Richardson, “Microservices Pattern: A pattern language for microservices”, *microservices.io*. https://microservices.io/patterns/

[87] D. S. K. S. and D. A. Singh, *SOFTWARE TESTING*. Vandana Publications, 2012

[88] H. K. N. Leung and L. White, “A study of integration testing and software regression at the integration level”, *IEEE Xplore*, Nov. 01, 1990. https://ieeexplore.ieee.org/abstract/document/131377

[89] G. Marreiros, “Agentes de apoio à argumentação e decisão em grupo”, *repositorium.sdum.uminho.pt*, Jan. 25, 2008. http://repositorium.sdum.uminho.pt/handle/1822/7643

[90] J. Carneiro, D. Martinho, G. Marreiros, and P. Novais, “Arguing with Behavior Influence: A Model for Web-Based Group Decision Support Systems”, *International Journal of Information Technology & Decision Making*, vol. 18, no. 02, pp. 517–553, Mar. 2019, doi: https://doi.org/10.1142/s0219622018500542.

[91] G. Marreiros, R. Santos, C. Ramos, J. Neves, and J. Bulas-Cruz, “ABS4GD: A Multi-agent System that Simulates Group Decision Processes Considering Emotional and Argumentative Aspects”, *DBLP Computer Science Bibliography*, 2008. https://dblp.uni-trier.de/rec/conf/aaaiss/MarreirosSRNB08.html?view=bibtex.

Anexo A – Vista de processos nível 3 (UC3)