NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

SOFIA ALEXANDRA SALGUEIRO DA ROCHA

Licenciada em Engenharia Informática

AUTOMATED SUPPORT FOR CREATING EXPERIMENTAL SETUPS IN MATERIAL SCIENCES

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Setembro, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

AUTOMATED SUPPORT FOR CREATING EXPERIMENTAL SETUPS IN MATERIAL SCIENCES

SOFIA ALEXANDRA SALGUEIRO DA ROCHA

Licenciada em Engenharia Informática

Orientador: Matthias Knorr
Professor Auxiliar, Universidade NOVA de Lisboa

Júri

Presidente: Doutor João Carlos Antunes Leitão
Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Arguente: Doutor João F. Ferreira
Professor Auxiliar, Instituto Superior Técnico da Universidade de Lisboa

Vogal: Doutor Jörg Matthias Knorr
Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Setembro, 2022

Automated Support for Creating Experimental Setups in Material Sciences

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeX e o modelo NOVAthesis (v6.9.6) [17].

Agradecimentos

Gostaria de agradecer a todas as pessoas que me apoiaram ao longo deste projeto e que me ajudaram a superar as dificuldades que nele foram surgindo.

Em primeiro lugar, quero agradecer ao meu orientador, o professor Matthias Knorr, pela sua disponibilidade e apoio ao longo deste trabalho. Agradeço também a leitura das inúmeras versões deste documento e de todos os comentários dados para que eu pudesse fazer sempre mais e melhor.

Gostaria de agradecer à VICARTE, em especial às professoras Márcia e Andreia, que se mostraram sempre disponíveis para ajudar no que fosse necessário.

Quero ainda agradecer aos meus pais e aos meus irmãos, toda a paciência que tiveram ao longo deste ano tão difícil para todos e todo o apoio que me têm dado ao longo de toda a vida.

Por fim, não posso de deixar de agradecer aos meus colegas e amigos, Bruno Machado, Carolina Silva e Hugo Moreira que sempre me motivaram ao longo do meu percurso académico, estando presentes nos bons e, principalmente, nos maus momentos.

Resumo

Na área de conservação e restauro, o interesse na investigação de fontes escritas históricas tem vindo a crescer nos últimos anos, no entanto, estes estudos raramente exploram a evolução de materiais de pintura e a reprodução de receitas. As receitas são conjuntos de instruções para preparar materiais (vidro, pigmentos, esmaltes) e são reproduzidas com o objetivo de avaliar os métodos utilizados, bem como de determinar as causas da deterioração dos materiais.

Na unidade de investigação VICARTE - Vidro e Cerâmica para as Artes, no âmbito da investigação da linha temática de estudos do património cultural, investigam-se estas receitas históricas através da sua reprodução, o que requer planear experiências tendo em consideração o material e os recursos disponíveis. No entanto, o número elevado de possíveis combinações de utilização destes recursos, tornam a realização do planeamento de forma manual complicada e morosa.

Nesta dissertação foi desenvolvida uma solução para este problema, em colaboração com a unidade de investigação VICARTE, recorrendo ao uso de *Answer Set Programming* de forma a criar horários para a realização das experiências, otimizando os recursos disponíveis. Como resultado foi desenvolvida uma aplicação que facilita a criação dos mesmos.

Palavras-chave: Programação por Conjuntos de Resposta, Sistemas Inferência ASP, Problemas de Agendamento

Abstract

In the field of conservation and restoration, interest in the investigation of historical written sources has been growing in recent years, however, these studies rarely explore the evolution of painting materials and the reproduction of recipes. The recipes are a set of instructions for preparing materials (glass, pigments, enamels) and are reproduced in order to evaluate the methods used, as well as to determine the causes of deterioration of the materials.

In the research unit VICARTE - Glass and Ceramics for the Arts, within the scope of research in the thematic line of studies of cultural heritage, these historical recipes are investigated through their reproduction, which requires planning experiments taking into account the material and resources available. However, the high number of possible combinations for the use of these resources makes manual planning complicated and time consuming.

In this dissertation, was developed a solution to this problem, in collaboration with the research unit VICARTE, using *Answer Set Programming* in order to create schedules for performing the experiments, optimizing the available resources. As a result, was developed an application that facilitates the creation of these.

Keywords: Answer Set Programming, Inference Systems, Scheduling Problems

Índice

Índice de Figuras

Índice de Tabelas

Índice de Listagens

Siglas

ASP *Answer Set Programming*

KRR *Knowledge Representation and Reasoning*

ORM *Object Relational Mapping*

1

Introdução

Neste capítulo será apresentado o contexto e a descrição do problema que se pretende abordar nesta dissertação, bem como a motivação. Por fim, serão apresentados os objetivos finais desta dissertação, as contribuições chave e a forma como o documento está estruturado.

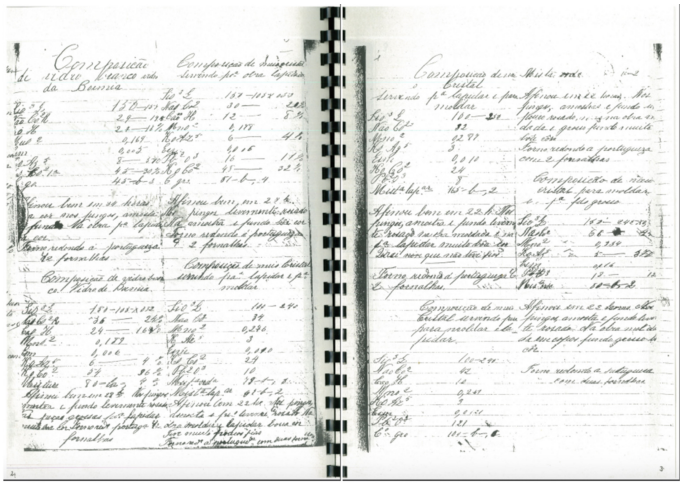
1.1 Contexto e Motivação

Nos últimos anos, no âmbito da área de conservação e restauro, tem vindo a crescer o interesse na investigação de fontes escritas históricas, onde são descritas técnicas de produção e decoração de vidro, bem como os materiais utilizados para tal [18]. No entanto, nestes estudos raramente se explora a evolução dos materiais de pintura e a reprodução de receitas históricas.

Essas receitas são conjuntos de instruções para preparar materiais (vidro, pigmentos, esmaltes) e são reproduzidas com o objetivo de avaliar os métodos utilizados, bem como de determinar as causas da deterioração dos materiais.

Cada receita existente tem o seu contexto histórico. Estas receitas encontram-se em livros escritos à mão, onde nem sempre é possível entender na totalidade a sua preparação, uma vez que ou se encontram escritas de uma forma geral ou têm informação codificada para o caso do livro ser roubado [22]. Além disso, estes livros encontram-se muitas vezes rasurados, sem existência de unidades de medida, ou com diferentes unidades de peso que são usadas em diferentes países e diferentes épocas, sendo por isso necessário realizar conversões de unidades. Na Figura 1.1, podemos observar exemplos de receitas presentes nestes livros.

(A)



(B)

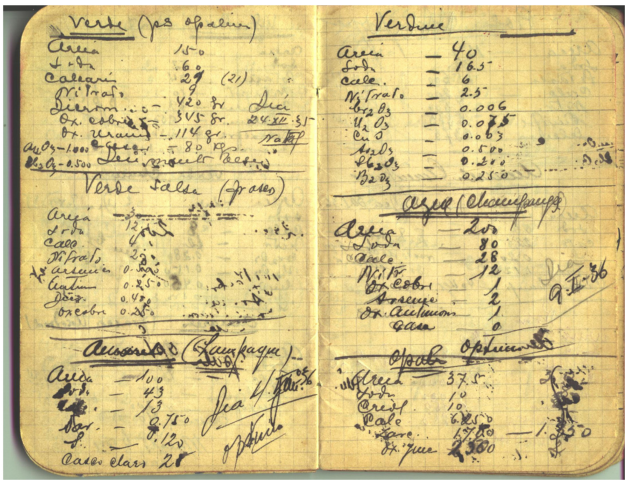


Figura 1.1: Exemplos de como se encontram escritas as receitas nos livros.

Para a preparação das receitas podem ser necessários vários passos, onde em cada um são realizadas ações (misturar, fundir, aplicação de esmaltes). Em alguns casos, os elementos necessários para a realização dum passo da receita têm a sua própria receita, ou seja, existe a noção de sub-receitas [23]. Esta noção fica mais clara se olharmos para a Figura 1.2, onde por exemplo a receita nº124, utiliza as receitas nº18 e nº93 no seu procedimento. Como tal, são sub-receitas da receita nº124. Por sua vez, a receita nº93 utiliza a receita nº62, portanto esta é sub-receita da nº93.

Existem ainda restrições relativamente aos fornos que podem ser utilizados, sendo que certas receitas só podem ir para um determinado forno. Também é importante referir que pode existir mais do que uma amostra nas mesmas condições no forno. Este é um factor que irá depender não só do tamanho do forno, como também do tamanho dos cadinhos utilizados. Os cadinhos são recipientes que resistem a temperaturas muito elevadas e que permitem fundir materiais a altas temperaturas.

Estas experiências podem consumir muito tempo, o que pode dificultar a coordenação

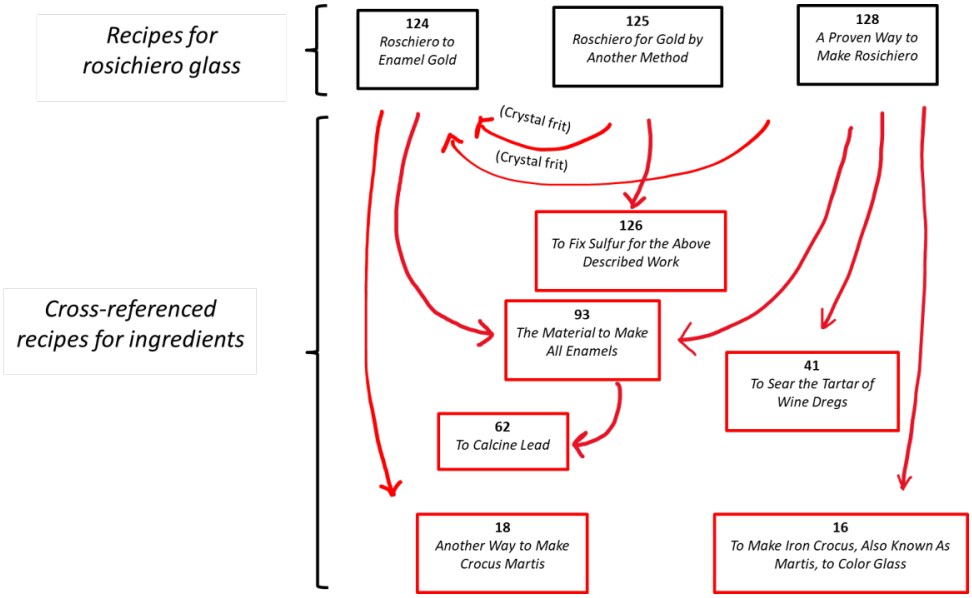


Figura 1.2: Exemplo do conceito de sub-receitas.

dos vários projetos de investigação, tendo em conta não só os recursos materiais, como também os recursos humanos, uma vez que estes são variáveis tendo em conta o acolhimento de estudantes de doutoramento/mestrado no âmbito das suas teses, bem como de outros investigadores. Assim sendo, a realização de planeamentos de forma manual torna-se complicada e morosa, devido ao número elevado de combinações possíveis de utilização dos recursos existentes.

Desta forma, com o fim de gerir melhor o tempo e os recursos, pretende-se, em colaboração com a unidade de investigação VICARTE - Vidro e Cerâmica para as Artes, criar horários para a realização das experiências, otimizando os recursos disponíveis.

Como sabemos a criação de horários, já é por si um problema combinatório. Se adicionarmos as restrições relativas a fornos, materiais e pessoas disponíveis para trabalhar, este problema torna-se mais complexo. Como tal, terá de ser utilizada uma abordagem que permite resolver este problema de uma forma simples e eficaz.

ASP é uma linguagem que permite resolver problemas de forma declarativa [14]. Esta linguagem tem vindo a ser utilizada em áreas como a Inteligência Artificial e Representação de Conhecimento e Raciocínio. Nos últimos tempos, tem sido aplicada com o objetivo de resolver problemas no contexto industrial, por exemplo no e-turism [12].

Esta é uma linguagem que é ideal para problemas que envolvem combinatória, sendo por isso indicada para endereçar este problema. Além disso, permite reduzir o esforço de implementação devido ao facto de ser uma linguagem que se foca em descrever o problema e não em como o resolver.

Para a criação dos horários, pretende-se utilizar ASP de forma a otimizar os recursos disponíveis de forma a que sejam cobertas as combinações possíveis de factores a serem testados. Será também desenvolvida uma aplicação que facilitará a criação dos horários.

As informações relativas às várias receitas encontram-se numa tabela, cujo o seu conteúdo e a sua estrutura passaram por um processo de formalização, de forma a se obter uma melhor estruturação dos dados, que foi então utilizada numa base de dados.

1.2 Objetivos Finais e Contribuições Chave

Os objetivos desta dissertação serão explorar o uso de ASP para a criação de horários para a realização das receitas, de forma a que os recursos disponíveis sejam otimizados. Para tal, será desenvolvida uma aplicação que facilite a criação dos mesmos.

Com o trabalho desenvolvido nesta dissertação pretende-se:

1. Estruturar os dados das receitas e disponibilizá-los numa base de dados;
2. Desenvolver uma aplicação recorrendo a ASP para a criação de horários que permitirá responder às seguintes perguntas:

   - Quantas receitas de uma dada cor ou material se podem fazer em X tempo?

   - Quanto tempo é necessário para produzir X receitas de uma cor ou material?

   - Quantas e quais receitas se podem fazer com X quantidade de elemento ou composto?

As perguntas apresentadas resultaram de reuniões com membros da VICARTE, onde se discutiram os requisitos para a aplicação. Foi definido ainda que a resposta obtida para cada pergunta seria uma tabela com o horário resultante para realizar as receitas.

1.3 Estrutura do Documento

Este documento encontra-se estruturado da seguinte maneira:

- No Capítulo 2 são apresentados os conceitos e tecnologias que serão importantes para a compreensão dos seguintes capítulos desta dissertação. O capítulo começa com uma introdução a *Answer Set Programming* e prossegue com a apresentação de ferramentas para edição de ficheiros ASP e alguns sistemas de inferência. Por último, serão analisadas estas tecnologias de forma a ser tomada uma escolha ponderada.
- No Capítulo 3 serão analisados dois artigos que para além de endereçarem problemas com um propósito semelhante ao desta dissertação, foram também utilizadas técnicas que se pretendem explorar no contexto da mesma.
- No Capítulo 4 é descrito como estão organizados os dados das receitas.
- No Capítulo 5, será apresentado o processo de implementação da aplicação, com as justificações das opções tomadas, onde serão descritas as várias componentes da aplicação.

- No Capítulo, 6, serão apresentados os resultados da avaliação. Serão analisados os resultados de desempenho e os resultados do questionário realizado aos membros da VICARTE que testaram a aplicação.
- Por fim, no Capítulo 7, serão apresentadas as conclusões desta dissertação e serão mencionados possíveis trabalhos futuros.

2

Background

Neste capítulo serão apresentados os conceitos e tecnologias relevantes para a compreensão dos capítulos seguintes.

2.1 *Answer Set Programming*

Answer Set Programming (ASP) [14] é uma linguagem que permite resolver problemas de forma declarativa, ou seja, em vez de se focar em como resolver o problema, limita-se simplesmente a descrevê-lo e a usar um *reasoner* ou um *solver* com base na descrição. Esta linguagem é utilizada em diversos contextos, nomeadamente, *Knowledge Representation and Reasoning* (KRR), Bases de Dados, Programação em Lógica.

Consegue tirar-se um excelente partido desta abordagem para problemas combinatórios, uma vez que são problemas que possuem uma grande quantidade de decisões e restrições.

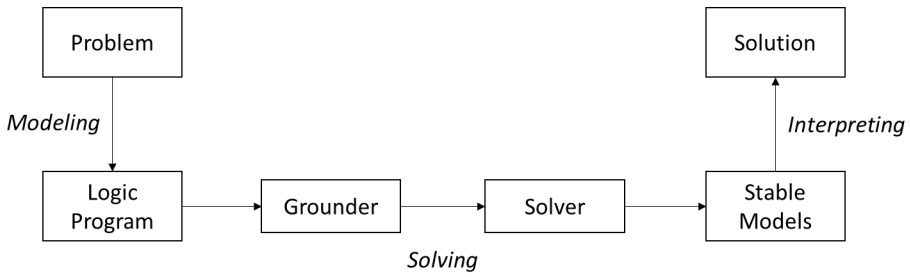


Figura 2.1: Resolução de problemas em ASP.

Na Figura 2.1, podemos observar que ao modelarmos o problema original, conseguimos obter uma representação formal do mesmo (*Logic Program*), modelado através de restrições. A partir dessa representação conseguimos computar os modelos que originam soluções para o problema. Para tal, o *grounder* irá gerar um conjunto finito de representações proposicionais do programa de *input*, através das quais o *solver* irá computar os modelos estáveis do programa proposicional. Por fim, a solução pode ser encontrada nos modelos estáveis resultantes.

Como vimos, o *grounder* gera um conjunto finito de representações proposicionais do programa de *input*. No entanto, a fase de *grounding* pode aumentar drasticamente o tamanho do programa de *input* [12]. Por essa razão, devem ser otimizados os *encodings*, principalmente no caso de problemas que tenham instâncias maiores, de forma a reduzir o tamanho do programa *grounded*.

Pode ocorrer uma situação semelhante relativamente ao *solver* pelo facto de existirem muitas decisões não determinísticas. Quando isto acontece pode tentar-se melhorar o *encoding*, no entanto, por norma, é necessário ser-se muito experiente em ASP. Outra possibilidade é ajustar as configurações do *solver* [12].

Num programa ASP, tipicamente geram-se primeiro todos os candidatos a conjuntos de resposta e, de seguida, eliminam-se os que não são válidos, por não respeitarem certas restrições impostas.

De seguida, será apresentada a sintaxe e a semântica de programas ASP.

Um programa lógico, P, sobre um conjunto A de átomos é um conjunto finito de regras.

Sintaxe

Uma regra normal, r, tem a seguinte forma

$$a_0 \leftarrow a_1, ..., a_m, \sim a_{m+1}, ..., \sim a_n,$$

onde $0 \leqslant m \leqslant n$ e cada $a_i \in A$ é um átomo para $0 \leqslant i \leqslant n$.

Consideremos agora as seguintes notações:

$head(r) = a_0$

$body(r) = \{a_1, ..., a_m, \sim a_{m+1}, ..., \sim a_n\}$

$body^+(r) = \{a_1, ..., a_m\}$

$body^-(r) = \{a_{m+1}, ..., a_n\}$

Um programa P diz-se positivo se $body^-(r) = \varnothing$ para todas as regras de P.

Nos exemplos abaixo, podemos observar um programa positivo e um programa normal.

Programa Positivo

$chuva \leftarrow$

$molhado \leftarrow chuva$

$molhado \leftarrow rega$

Programa Normal

$rega \leftarrow \sim chuva$

$chuva \leftarrow \sim rega$

$molhado \leftarrow chuva$

$molhado \leftarrow rega$

Semântica

Um conjunto de átomos X é fechado sob um programa positivo P se e só se para alguma $r \in P$, $head(r) \cap X \neq \varnothing$ sempre que $body^+(r) \subseteq X$.

O menor conjunto de átomos que é fechado sob um programa positivo P é denotado por $Cn(P)$.

O conjunto $Cn(P)$ de átomos é modelo estável de um programa positivo P.

Os modelos estáveis vão corresponder às soluções do problema inicial.

Podemos determinar se um modelo de um programa é estável recorrendo a um método iterativo. Este método começa com um modelo vazio e a cada iteração acrescentam-se todas as conclusões imediatas a partir do que já se concluiu anteriormente. Este método termina quando já não for possível concluir mais nada.

Voltando ao exemplo do programa positivo. Este programa tem 2 modelos clássicos (modelo em que todas as condições são verdadeiras): {*chuva*, *molhado*} e {*chuva*, *molhado*, *rega*}.

Para determinar se {*chuva*, *molhado*} é modelo estável, vamos utilizar o método iterativo.

$I_0 = \{\}$

$I_1 = \{chuva\}$

$I_2 = \{chuva, molhado\}$

$I_3 = \{chuva, molhado\}$

Uma vez que o que se concluiu na última iteração é igual ao modelo que queríamos averiguar, {*chuva*, *molhado*} é modelo estável.

Para programas que contenham negações, nem sempre é fácil de se intuir que modelos serão estáveis. Isto leva a que seja necessário definir o que é um modelo estável de um programa.

Consideremos P^X a redução de um programa P relativamente a um conjunto de átomos X definida como:

$$P^X = \{head(r) \leftarrow body^+(r) | r \in P \wedge body^-(r) \cap X = \emptyset\}$$

Um conjunto de átomos X é modelo estável de um programa P se $C_n(P^X) = X$.

Assim, para o caso do exemplo do programa normal, o objetivo é para um conjunto de átomos X, eliminar todas as regras que contenham no seu corpo átomos negados que pertençam a X e, de seguida, eliminar todos os átomos que estejam negados nas restantes regras.

Posteriormente, pode aplicar-se o método iterativo para determinar os modelos estáveis.

No caso do exemplo do programa normal, os modelos clássicos são {*rega*, *molhado*}, {*chuva*, *molhado*} e {*chuva*, *molhado*, *rega*}.

Vamos tentar determinar se {*rega*, *molhado*} é modelo estável. Realizando a redução do programa ficamos com as regras seguintes:

$rega \leftarrow$

$molhado \leftarrow chuva$

$molhado \leftarrow rega$

Ao aplicar o método iterativo chegamos à conclusão de que {*rega*, *molhado*} é modelo estável.

$I_0 = \{\}$

$I_1 = \{rega\}$

$I_2 = \{rega, molhado\}$

$I_3 = \{rega, molhado\}$

Para os restantes modelos clássicos do programa normal ao realizar a redução do programa normal, iríamos concluir que o mesmo tem dois modelos estáveis: {*rega*, *molhado*} e {*chuva*, *molhado*}.

2.2 Extensões da linguagem ASP

A linguagem ASP possui ainda construtores da linguagem como *choice rules*, *cardinality rules*, restrições de integridade e *optimization statements*.Estas regras permitem dar expressividade à linguagem, sendo por isso essenciais para criar programas ASP.

De seguida, serão apresentadas as extensões da linguagem.

2.2.1 *Choice Rules*

A ideia das *choice rules* é expressar escolhas sobre subconjuntos.

Para ilustrar melhor esta ideia, consideremos o exemplo abaixo. Este exemplo ilustra o que podemos comprar numa mercearia, ou seja, qualquer subconjunto dentro de pizza, vinho e milho.

```
{ buy(pizza), buy(wine), buy(corn) } :- at(grocery).
```

2.2.2 *Cardinality Rules*

As *cardinality rules* permitem controlar a cardinalidade dos subconjutos através de um limite inferior *l*.

Para ilustrar melhor esta ideia, consideremos o exemplo abaixo, que considera que uma pessoa para passar à disciplina *d* tem que passar a pelo menos 2 trabalhos.

```
pass(d) :- 2 { pass(t1), pass(t2), pass(t3) }.
```

2.2.3 Restrições de Integridade

As restrições de integridade permitem eliminar modelos com condições indesejadas para a solução. Neste exemplo pretendemos colorir os vértices de um grafo, de forma a que vértices adjacentes fiquem com cores diferentes. Assim, é possível eliminar todos os modelos onde o vértice 1 e o vértice 2 são adjacentes e têm os dois a cor verde.

```
:- edge(1,2), color(1,green), color(2,green).
```

2.2.4 *Optimization Statements*

A linguagem ASP permite expressar funções de custo de minimização e de maximização. No exemplo abaixo, o predicado allEmptySeats identifica o número total de lugares vazios numa sala. O código que permite minimizar o número de lugares vazios numa sala é o seguinte:

```
#minimize {W: allEmptySeats(W)}.
```

2.3 Ferramentas edição ASP

A escrita de programas ASP pode tornar-se morosa, caso não se possua muitos conhecimentos em relação à sintaxe da linguagem. Além disso, tarefas como testar o programa e realizar o seu *debugging*, podem ser um desafio sem o auxílio de ferramentas adequadas.

Nesta secção serão analisadas algumas ferramentas que nos facilitam a escrita de programas ASP. Primeiro, será apresentada a ferramenta LoIDE, de seguida SeaLion e, por fim, ASPIDE.

2.3.1 LoIDE

LoIDE [15] é um IDE *web-based* para Programação Lógica. Este IDE para além das funcionalidades básicas de edição de texto, estende ainda funcionalidades que auxiliam o processo de escrita de programas ASP, suportando *syntax highlighting* e *output highlighting*. É ainda de realçar o facto de parêntesis correspondentes ficarem destacados.

Desta forma, estas funcionalidades contribuem de maneira positiva para a usabilidade desta ferramenta, fazendo com que o utilizador da mesma realize a escrita de programas ASP com maior facilidade, maior conforto.

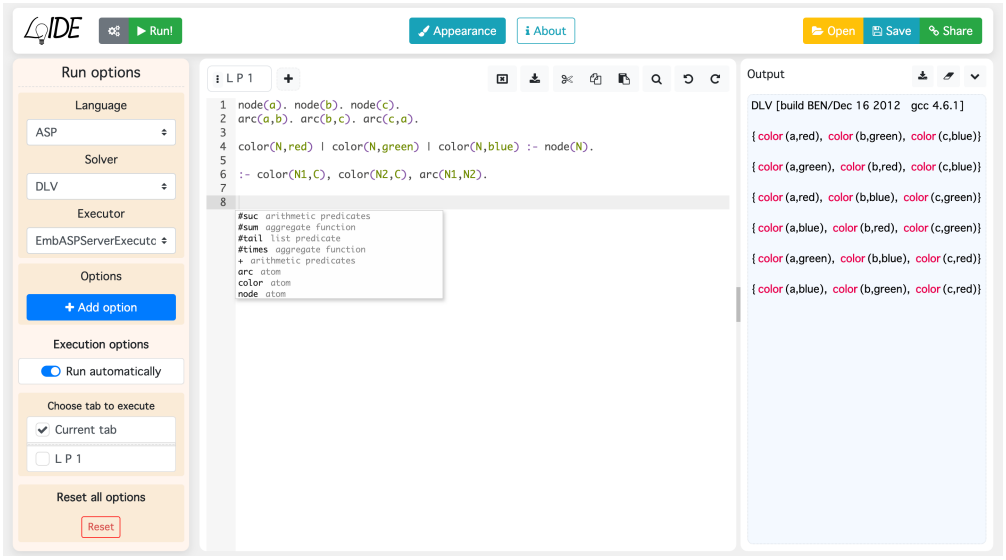


Figura 2.2: LoIDE: interface gráfica do utilizador.

Na Figura 2.2, encontra-se a interface gráfica desta ferramenta. No topo da figura, podemos observar a barra de navegação que é constituída pelos botões *Run*, *Apperance*, *Open*, *Save* e *Share*. Do lado esquerdo da figura, podemos encontrar o painel com todas as opções disponíveis. Do lado direito, está o painel do *output* onde são apresentadas as computações. Por fim, na zonal central, encontra-se o editor de código.

2.3.2 SeaLion

SeaLion [19] é um IDE para ASP que se encontra como um plugin do Eclipse. Suporta as linguagens DLV e Gringo. Tal como o LoIDE, o SeaLion também dispõe da funcionalidade de *syntax highlighting*. Além disso, destacam-se ainda a verificação de sintaxe, a reportação dos erros e o seu realce.

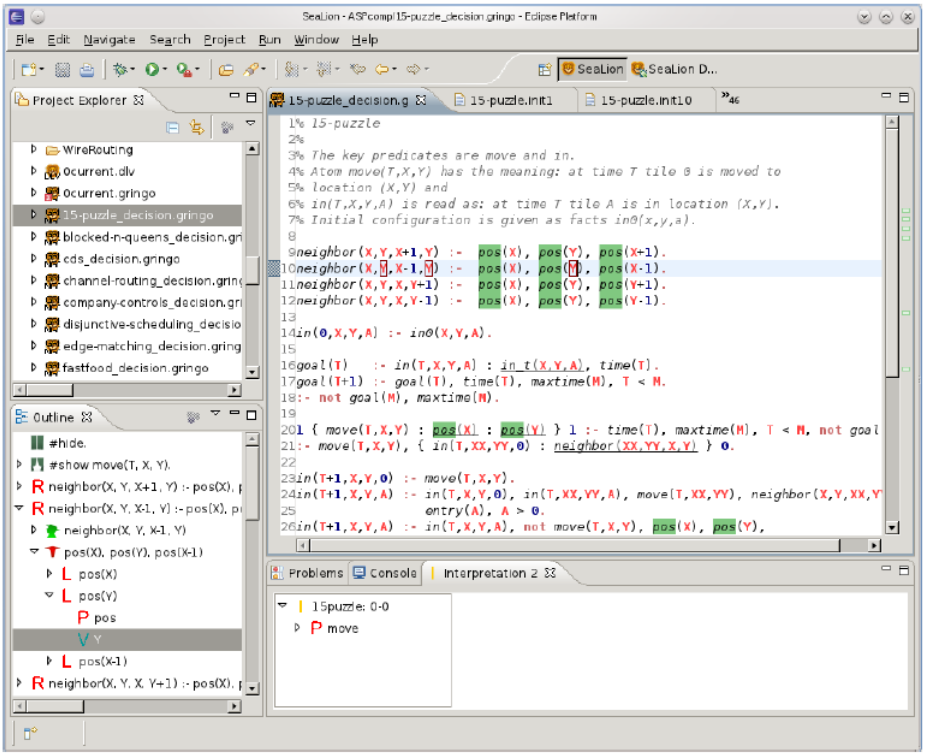


Figura 2.3: SeaLion: interface gráfica do utilizador.

Para facilitar a execução dos programas é possível criar configurações nas quais se podem definir os ficheiros de *input*, a configuração do *solver* e dos argumentos da linha de comandos que este necessita, tal como a forma de processamento do *output* (os conjuntos de resposta podem ser guardados numa vista para interpretações ou podem ser apresentados na consola do Eclipse).

É de realçar a funcionalidade de visualização e de edição visual de interpretações, onde através da qual é possível gerar a visualização a partir de um dos conjuntos de resposta obtidos para um dado programa associado com a interpretação a ser visualizada. Esta funcionalidade encontra-se também como um plugin do Eclipse cujo nome é Kara [16].

Na Figura 2.3, pode observar-se a interface gráfica do SeaLion. No topo da figura, podemos observar a barra de navegação. Do lado esquerdo da figura, podemos encontrar o painel com o *outline* do programa e o *project explorer*. O painel na zona mais abaixo da figura é a vista de interpretações. Por fim, a janela central é o editor de código.

2.3.3 ASPIDE

ASPIDE [13] é um IDE que permite a organização de programas ASP em vários projetos dentro de um *workspace*, similarmente ao Eclipse.

Tal como o LoIDE e o SeaLion, suporta as funcionalidades básicas de edição de texto. Para facilitar a edição de ficheiros ASP, dispõe ainda de funcionalidades mais avançadas tais como o preenchimento automático, *quick fix* (os erros ou avisos reportados podem ser resolvidos selecionando uma das sugestões) e verificação de código dinâmico e destaque de erros (os erros de sintaxe e condições como *safety* são verificados quando se escreve o programa e caso existam erros ou avisos, estes são imediatamente realçados).

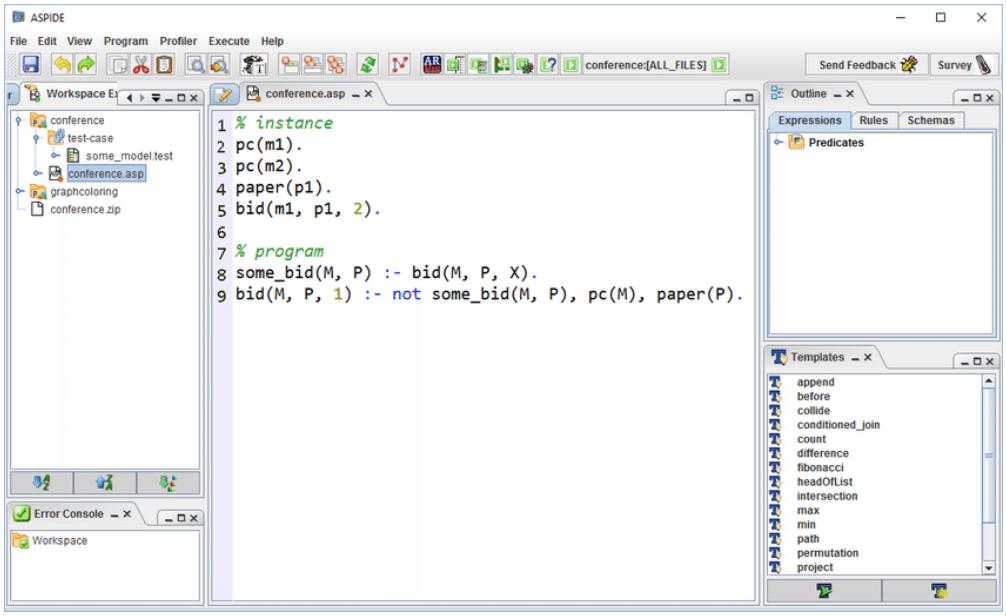


Figura 2.4: ASPIDE: interface gráfica do utilizador.

É possível configurar as condições de execução definindo programas de *input*. O resultado da execução do programa pode ser visualizado numa representação tabular ou na consola em formato de texto.

Possui um editor visual através do qual o utilizador pode "desenhar"programas para criar regras lógicas.

Na Figura 2.4, encontra-se a interface gráfica do ASPIDE. No topo da figura, podemos observar a barra de navegação. Do lado esquerdo da figura, podemos encontrar o painel com o *workspace*. Do lado direito, estão os painéis do *outline* do programa e *templates*. Por fim, na zonal central, encontra-se o editor de código.

2.4 Sistemas de inferência ASP

Um sistema de inferência tem como propósito estender uma base de conhecimento. A base de conhecimento é um conjunto de factos que representam o que o sistema conhece em relação a um domínio. O sistema processa as regras e os factos da base de conhecimento de forma a obter uma resposta.

2.4.1 DLV

DLV é um sistema baseado em *disjunctive logic programming*, O sistema incorpora vários *front-ends* para lidar com vários formalismos de representação de conhecimento avançados.

O DLV foi um dos primeiros sistemas monolíticos (i.e. sistema com o *grounder* e o *solver* na mesma implementação) de ASP confiável e eficaz, tornando-o assim indicado para aplicações do mundo real [1].

Tem sido utilizado na indústria para gestão de recursos humanos, turismo, encaminhamento inteligente de chamadas e medicina [1].

No entanto, este sistema também tem sido utilizado em meio académico em unidades curriculares como Bases de Dados e Inteligência Artificial [3].

O DLV tem sido atualizado ao longo do tempo com técnicas de avaliação modernas e plataformas de desenvolvimento. Primeiramente, foram lançados separadamente um *grounder*, $\mathcal{I}$-DLV, e um *solver*, `WASP`. Posteriormente, surgiu o DLV2, um sistema monolítico onde foram integrados $\mathcal{I}$-DLV e `WASP`.

2.4.2 DLV2

O DLV2 é um sistema monolítico que segue a linguagem *standard* ASP-Core-2. Este sistema tirou partido no que diz respeito à usabilidade, à performance e à facilidade de integração em aplicações baseadas em ASP, por ter integrado as características do *grounder* $\mathcal{I}$-DLV e do *solver* `WASP`.

Em comparação com a sua primeira versão, o DLV2 realizou melhorias que permitiram não só o acesso a dados de sistemas de gestão de bases de dados relacionais, como também a *graph databases*.

Mais detalhes sobre DLV2 podem ser encontrados em [4].

2.4.3 Clingo

O Clingo [14] é um sistema monolítico que resulta da combinação do *grounder gringo* e do *solver clasp*.

O *gringo* traduz programas lógicos em programas *ground* equivalentes. O resultado desta tradução é enviado para o *solver clasp*, que irá então computar os conjuntos de resposta.

O Clingo é um sistema *state-of-the-art*, que segue a linguagem standard ASP-Core-2.

É ainda de referir que o *solver clasp* é muito eficiente, como se pode comprovar pelos prémios que lhe foram atribuídos, entre os quais:

- No *Configurable SAT Solver Challenge 2013*, venceu o primeiro lugar nas categorias *Crafted Track* e *Random Track*.
- Na *ASP Competition 2011*, venceu o primeiro lugar na categoria *System Track (P Problems)*, *Model + Solve (Overall, P, NP, Beyond NP, Optimization)*, o segundo lugar na categoria *System Track (NP Problems)* e terceiro lugar na categoria *System Track (Overall)*.

2.5 *Embedding* de ASP

Existem ferramentas que permitem integrar ASP de forma a facilitar o desenvolvimento e a implementação de aplicações. Nas secções seguintes, serão descritas duas delas.

2.5.1 EMBASP

EMBASP é uma *framework* abstrata para a integração de ASP em sistemas externos para aplicações genéricas, que tem o objetivo de facilitar o desenho e a implementação de tarefas complexas de *reasoning* através de *solvers* em diferentes plataformas. Atualmente, providencia bibliotecas para incorporar vários *solvers*, entre eles encontram-se o Clingo, DLV e DVL2.

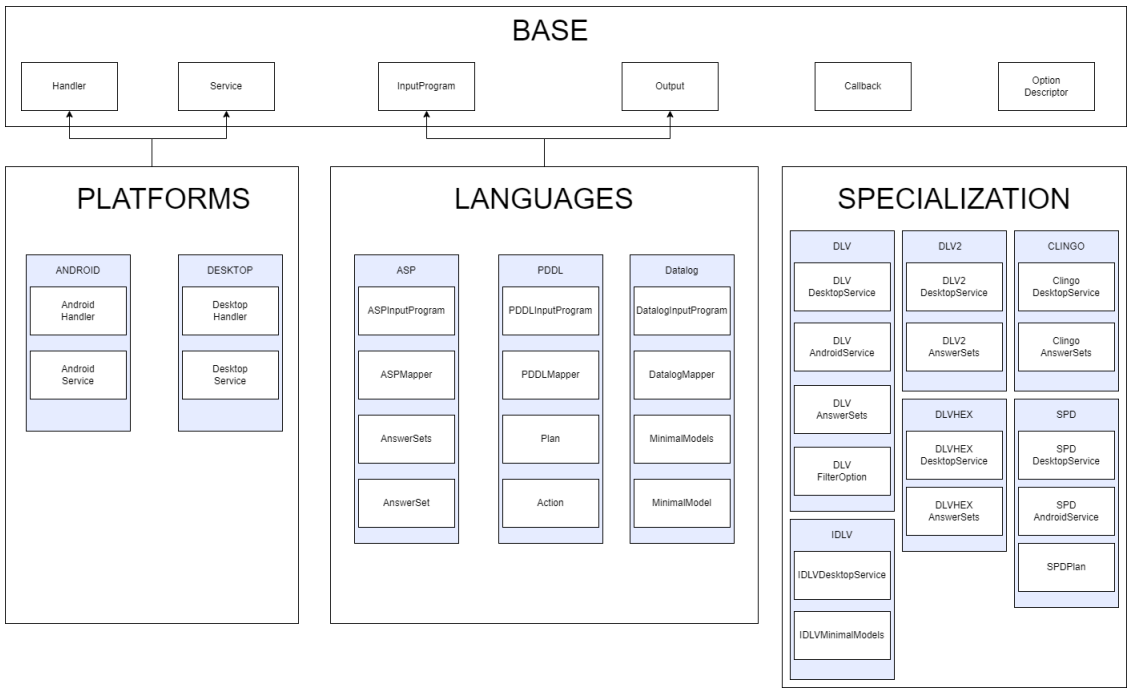


Figura 2.5: Arquitetura da *framework* abstrata EMBASP.

Na Figura 2.5, pode ser vista a arquitetura desta *framework* que é composta por quatro módulos: **Base** (define os componentes básicos da *framework*), **Plataformas** (contém o que é dependente da plataforma, logo os componentes *Handler* e *Service* do módulo Base devem de ser adaptados à plataforma em questão), **Linguagens** (define recursos específicos para cada linguagem suportada) e **Especialização** (define o que é dependente do sistema) [11].

A arquitetura apresentada é abstrata, no entanto existem implementações da mesma em linguagens de programação orientada a objetos. Atualmente, existem implementações em Python, Java e C# [11].

2.5.2 Clorm

O Clingo possui uma API que lhe permite fazer integração de ASP com Python. No entanto, esta API funciona num nível baixo quando se pretende enviar dados do *solver* ou para o *solver*. Esta situação pode tornar-se mais complexa à medida que o programa ASP evolui [6].

Para facilitar pode utilizar-se o Clorm que disponibiliza uma interface *Object Relational Mapping* (ORM) para o Clingo, através da qual é possível mapear predicados ASP para objetos no Python.

O programa ASP vai definir o domínio do problema. Como as instâncias dos problemas estão em constante alteração, não devem ser codificadas no ficheiro ASP. Deste modo, o programa do Python vai definir as instâncias, que depois serão enviadas para o *solver* para computar a solução [7].

Para definir o mapeamento dos predicados do Clingo para objetos do Python será necessário definir o modelo de dados, utilizando para tal as classes do Clorm `Predicate` e `ComplexTerm`.

Os parâmetros dos predicados são especificados utilizando classes *field*, nomeadamente, `ConstantField`, `StringField` e `IntegerField`. É de referir ainda que o número de *fields* dentro da definição de um `Predicate`, deve ser exatamente o mesmo que a cardinalidade do predicado. A ordem pela qual os *fields* são declarados também deve ser a mesma que a dos predicados [7].

Quando o modelo dos dados estiver criado, podemos utilizá-lo para instanciar os factos. Para isso, é necessário criar um objeto da classe `Control` que é a responsável por gerir as interações com o *solver*. Através da função `load` dessa classe é carregado o programa ASP.

Nas Figuras 2.6 e 2.7, pode ver-se um exemplo [7] da definição do modelo de dados. Neste exemplo, existem condutores que realizam entregas de vários itens, que podem ser entregues em quatro *slots* diferentes. Neste caso, o objetivo é atribuir condutores para realizarem as entregas de cada item, de forma a minimizar o número de condutores utilizados e de forma a entregar os itens o mais cedo possível.

Ao analisar as duas Figuras, podemos verificar o facto dos predicados terem o mesmo nome em ambos os programas. Além disso, a definição de cada predicado no programa Python tem exatamente o mesmo número de *fields* que o do programa ASP e os mesmos estão declarados exatamente pela mesma ordem.

```python
from clorm import Predicate, ConstantField, IntegerField
from clorm.clingo import Control

class Driver(Predicate):
    name=ConstantField

class Item(Predicate):
    name=ConstantField

class Assignment(Predicate):
    item=ConstantField
    driver=ConstantField
    time=IntegerField
```

Figura 2.6: Programa Python do exemplo das entregas de itens.

```
time(1..4).

1 { assignment(I, D, T) : driver(D), time(T) } 1 :- item(I).
:- assignment(I1, D, T), assignment(I2, D, T), I1 != I2.

working_driver(D) :- assignment(_,D,_).

#minimize { 1@2,D : working_driver(D) }.
#minimize { T@1,D : assignment(_,D,T) }.
```

Figura 2.7: Programa ASP do exemplo das entregas de itens.

2.6 *Python GUI Frameworks*

Esta secção tem como efeito indicar *Python GUI Frameworks* que poderiam ser utilizadas na fase de implementação da interface gráfica da aplicação.

2.6.1 Tkinter

O Tkinter [10] é uma biblioteca *open source* standard para desenvolver interfaces gráficas para aplicações. A sua sintaxe é simples e não necessita de instalação uma vez que está integrada com o Python.

O Tkinter permite desenvolver aplicações para Windows, Linux e macOS, correndo exatamente o mesmo código.

2.6.2 Kivy

O Kivy [8] é uma biblioteca de Python *open source* que permite desenvolver aplicações multiplataforma em Windows, macOS, Android, iOS, Linux e Raspberry Pi.

Esta biblioteca ao contrário do Tkinter, necessita de instalação.

O Kivy tem ainda uma biblioteca chamada KivyMD [9]. Esta biblioteca contém uma coleção de materiais de design que se tenta aproximar das especificações do material de design da Google sem comprometer a facilidade de utilização da mesma.

2.7 Análise e seleção das ferramentas

Ao longo deste capítulo foram mencionadas ferramentas que poderiam auxiliar o trabalho desenvolvido no contexto desta dissertação.

Em primeiro lugar, foram apresentadas ferramentas de edição para programas ASP. Estas ferramentas em termos de funcionalidades são bastante parecidas. No entanto, a ferramenta LoIDE, ao contrário das restantes apresentadas, não implementa nenhuma funcionalidade de edição visual.

As três ferramentas apresentadas seriam boas propostas, no entanto os utilizadores teriam sempre de ter algum conhecimento relativamente a ASP. Uma vez que as pessoas que irão utilizar a aplicação não pertencem à área de Informática, não estão familiarizadas com ASP, portanto não lhes faz sentido a utilização destas ferramentas de edição. Assim sendo, o resultado final deste trabalho teve de ser uma ferramenta com a qual os utilizadores consigam interagir, sem necessitarem destes conhecimentos. No entanto, como estas ferramentas servem para o desenvolvimento de programas, podem ser utilizadas nesse contexto por *developers* de ASP.

Relativamente aos sistemas de ASP, foram apresentados o DLV, DLV2 e Clingo. Estes sistemas têm em comum o facto de serem sistemas monolíticos. Um dos pontos fortes dos sistemas DLV2 e Clingo é o facto de seguirem a linguagem ASP-Core-2.

Em termos de eficiência, o sistema DLV2 é substancialmente melhor que o sistema DLV [4]. Quando comparado ao sistema Clingo, o DLV2 tem uma eficiência semelhante, sendo assim os dois sistemas comparáveis.

O grande ponto a favor do Clingo é o facto de permitir a integração com a linguagem de programação Python, o que será certamente relevante para a implementação da aplicação, tendo em conta que facilita o mapeamento dos predicados.

Com base na análise realizada, tendo em consideração que é um sistema *state-of-the-art* e considerando os prémios que já recebeu pela sua eficiência, optou-se por utilizar o sistema ASP Clingo.

Relativamente às *Python GUI Frameworks* foram apresentadas as bibliotecas Tkinter e Kivy.

As duas bibliotecas são *open source* e permitem têm a vantagem de apenas se precisar

de escrever o código uma vez, visto que a partir do mesmo código é possível criar aplicações para vários dispositivos de plataformas diferentes. No entanto, o Kivy permite que o código possa ser utilizado em plataformas móveis, ao contrário do Tkinter.

O Kivy necessita de instalação, ao contrário do Tkinter.

Além disso, o Kivy para além de permitir desenvolver interfaces esteticamente mais bonitas que o Tkinter, ainda possui uma outra biblioteca com uma coleção de materiais de design que se tenta aproximar das especificações do material de design da Google.

Tendo em conta que as duas bibliotecas são bastante similares, optou-se por utilizar o Kivy, visto que permite utilizar o mesmo código em mais plataformas que o Tkinter e pelo facto da interface produzida ser mais agradável.

3

Trabalho Relacionado

Neste capítulo serão apresentados dois artigos onde é explicada a utilização de Answer Set Programming em problemas de contexto real.

3.1 Artigos relacionados

Nesta secção serão apresentados dois artigos que para além de terem um propósito semelhante ao desta dissertação, foram também utilizadas técnicas que se pretendem explorar no contexto da mesma.

Os conceitos chave presentes nesses artigos encontram-se descritos detalhadamente na Secção 2.1.

3.2 *Answer Set Programming* na Saúde

No artigo [2], são apresentados problemas de agendamento no domínio da saúde. Estes problemas foram resolvidos utilizando ASP e foram aplicados principalmente em Génova e, a partir de 2020, na CLAIRE COVID-19 Task Force, no âmbito do Agendamento e Gestão de Recursos.

De seguida, serão descritos os problemas *Nurse scheduling problem* e *Operating Room Scheduling problem* e para cada um deles serão referidas as melhorias realizadas. A descrição apresentada de cada problema considera requisitos tanto para pequenos e médios hospitais italianos, como para institutos privados, resultando assim numa descrição de problemas em contexto real.

Nurse scheduling problem

O objetivo deste problema é organizar os turnos de trabalho dos enfermeiros numa unidade hospitalar, tendo sempre em atenção que deve ser mantido um elevado nível de qualidade de cuidados de saúde. Os requisitos deste problema são os seguintes [2]:

- Devem ser considerados turnos da parte da manhã, tarde e noite, onde para cada um deles deve existir um número mínimo e máximo de enfermeiros presentes;
- Os enfermeiros devem ter entre si uma distribuição equilibrada da carga de trabalho;
- Os enfermeiros têm direito a 30 dias de férias;
- Um turno atribuído a um enfermeiro deve começar pelo menos 24 horas depois do início do seu turno anterior;
- Cada enfermeiro tem direito a 2 dias de descanso por cada período de 40 dias. No entanto, caso este realize 2 turnos consecutivos noturnos, tem direito a mais um dia de descanso para além dos anteriores.
- Tem de existir um número definido de vezes que um enfermeiro pode ser atribuído aos turnos da manhã, tarde e noite.

De forma a garantir um bom nível de qualidade, estas propostas foram aperfeiçoadas. Assim, em vez de serem só marcados turnos de trabalho para os enfermeiros, generalizou-se o problema e atribuíram-se turnos de trabalho a todos os funcionários do hospital. Desta maneira, as restrições que vimos anteriormente, também serão generalizadas.

As atribuições dos turnos de trabalho são realizadas em 2 fases, para contemplar os casos em que um funcionário trabalha em mais do que uma unidade hospitalar. Desta forma, são primeiro agendados os turnos dos trabalhadores que trabalham apenas numa unidade hospitalar. Só depois desta atribuição é que se procede à atribuição dos turnos dos funcionários que trabalham em mais do que uma unidade hospitalar.

No contexto do *Nurse scheduling problem*, caso um enfermeiro necessitasse de se ausentar repentinamente, o calendário dos seus turnos computado anteriormente não poderia ser utilizado. Assim, outra melhoria que foi efetuada começa por alterar as marcações computadas previamente a partir do primeiro dia em que já não esteja ausente, dando-se sempre maior prioridade à minimização das diferenças entre o novo agendamento e o antigo.

Operating Room Scheduling problem

O objetivo é deste problema é atribuir o maior número de salas de cirurgia considerando os recursos disponíveis. Cada cirurgia está registada na lista de espera do hospital e tem associadas um nível de prioridade (alta, média ou baixa) e a especialidade a que corresponde.

As cirurgias mais urgentes ou que se encontram há mais tempo na lista de espera têm um nível de prioridade alto, devendo por isso ser atribuídas sempre a um horário. As cirurgias de prioridade média devem ser atribuídas, mas podem ser adiadas caso seja necessário, enquanto que as de prioridade mais baixa podem ser utilizadas para preencher furos existentes no plano definido.

Existe um horário estabelecido pelo hospital (*Master Surgical Schedule*), onde são definidos o número e a distribuição de blocos de tempo das salas de cada especialidade. Este horário terá de ser considerado na fase de planeamento.

As restrições associadas a este problema são as seguintes [2]:

- Cada cirurgia atribuída a um bloco de tempo tem de ter a mesma especialidade que está associada ao bloco;
- Cada cirurgia tem de ser atribuída a um único bloco de tempo e a uma cama da especialidade associada a esse bloco para todos os internamentos;
- O tempo total das durações das cirurgias atribuídas a um bloco de tempo não pode exceder o tempo do próprio bloco;
- As cirurgias que o nível de prioridade mais alto devem ser todas atribuídas. No entanto, deve ser minimizado o número de cirurgias que não foram atribuídas, dando prioridade às cirurgias com prioridade média.

Neste problema ainda podem ser realizadas algumas melhorias.

Como referido anteriormente, o hospital definia o horário *Master Surgical Schedule*, porém a gestão do hospital pode querer ajustar e modificar esse horário. Desta forma, foi necessário realizar a codificação deste horário, tendo em conta as salas e as sessões disponíveis e as especialidades. O *encoding* para atribuir as salas de cirurgia passou a receber como *input* o do *Master Surgical Schedule*.

Pretende-se também atribuir o maior número de pacientes às salas de operação, considerando as diferentes especialidades, duração das cirurgias e a disponibilidade de camas para o internamento tanto nas unidades de cuidados intensivos como nas enfermarias. Assim, o objetivo é atribuir o número máximo de cirurgias às salas de operações, considerando a disponibilidade de camas em cuidados intensivos e enfermarias. Uma das restrições que foi tida em conta foi o facto de só se atribuir um paciente a uma dada sala de operações, caso existam camas disponíveis para o seu internamento (tempo antes da cirurgia na enfermaria e tempo depois da cirurgia nos cuidados intensivos ou na enfermaria).

Existem enfermarias associadas a uma dada especialidade, onde o número de camas disponíveis varia entre as mesmas. Só podem ser aceites numa enfermaria pacientes cuja especialidade seja a mesma que a da própria enfermaria. Nas salas de cuidados intensivos podem estar pacientes de qualquer uma das especialidades.

Para além das camas presentes nas enfermarias e nos cuidados intensivos, também foram consideradas camas para cuidados pós-anestésicos, onde um paciente a ocupa por uns minutos. Estas camas encontram-se tanto nas enfermarias como nos cuidados intensivos e por isso foi necessário alterar as restrições de forma a que o paciente possa ir para uma ou para a outra.

No contexto deste problema também pode ser necessário remarcar cirurgias. Para abranger esta situação, as cirurgias que precisarem de ser adiadas devem de ser remarcadas para um dos blocos de tempo seguintes do planeamento original. Apenas será necessário alterar o planeamento da especialidade associada à cirurgia a ser adiada, visto que esse planeamento não afeta as restantes especialidades.

Para a codificação do novo plano, o plano antigo será passado como *input* e serão adicionadas novas restrições para retratar o processo de remarcação.

3.3 *Answer Set Programming* na Indústria

No artigo [12] são apresentadas algumas aplicações de ASP na indústria e são descritos os desafios que surgem relativamente ao *grounding* e às heurísticas.

De seguida, serão descritas algumas aplicações de ASP.

No porto de Gioia Tauro [21], foi utilizado ASP para atribuir tarefas a funcionários de forma a que as mesmas fossem realizadas e que fossem cumpridos os regulamentos de trabalho.

Outro sistema apresentado valida e gere planos para configurar uma nave espacial para manobras. Os planos são habitualmente preparados para as situações mais frequentes. Ainda assim, não é possível pré-planear todas as situações em caso de falhas, daí a necessidade de implementar este sistema baseado em ASP.

Outra aplicação de ASP foi num portal de e-turismo, onde dadas as preferências do cliente, as propriedades dos pacotes de férias e algum conhecimento de *background*, o sistema seleciona o pacote de férias que mais se enquadra às necessidades dos clientes.

Mais informações sobre aplicações de ASP podem ser encontradas em [12].

Neste capítulo foram apresentados dois artigos onde foi utilizado ASP em problemas de agendamento.

No entanto, apesar de existir trabalho relacionado, não vai ser possível implementar nenhuma solução a partir das sugeridas que se adequa aos horários que se pretendem realizar nesta dissertação, uma vez que cada problema tem as suas características.

Os horários pretendidos nesta dissertação são obtidos considerando restrições e procedimentos que costumam ser efetuados na VICARTE. Deste modo, este é um problema com características próprias, portanto nenhuma das soluções que foi encontrada nos artigos anteriores se adapta neste caso. No entanto, pode-se inspirar nas soluções obtidas nestes trabalhos para a implementação da solução que permite criar os horários.

4

Análise e estruturação dos dados

Neste capítulo será analisada a organização dos dados das receitas e serão descritos quais deles serão necessários para a realização dos planeamentos das experiências.

O objetivo desta dissertação é explorar o uso de ASP para a criação de horários para a realização das receitas, de forma a que os recursos disponíveis sejam otimizados. Como resultado, foi desenvolvida uma aplicação que facilita a criação dos mesmos, através da qual os membros da VICARTE conseguem obter planificações das receitas que podem realizar com base em certas perguntas.

Para atingir os objetivos, foi realizado um trabalho preparatório onde se exploraram conceitos e ferramentas que serão importantes, bem como o trabalho relacionado.

Assim, estudaram-se ferramentas de edição de ASP e sistemas de inferência, de forma a escolher os que melhor se enquadrariam neste contexto.

Na secção seguinte, será descrito como estão organizados os dados das receitas.

4.1 Acesso e processamento dos dados

Os dados relevantes das várias receitas encontram-se partilhadas num ficheiro excel. As diversas receitas encontram-se nesse ficheiro, onde existe uma tabela com as informações que são relevantes para os utilizadores entenderem e executarem as suas receitas (procedimento, elementos utilizados, ...). Existiram reuniões com membros da VICARTE, onde se discutiu como seriam apresentados os dados relativos às receitas de forma a estruturar os dados.

De seguida, são apresentados três excertos relevantes da tabela das receitas. Esta tabela não será totalmente apresentada, visto que é uma tabela muito grande e para atingir os objetivos propostos não serão necessários todos os dados aí presentes. Isto acontece, visto que a tabela terá dados para os utilizadores conseguirem executar as receitas da forma mais autónoma possível. Desta forma, os dados que serão essenciais para atingir os objetivos desta dissertação encontram-se presentes nos três excertos apresentados.

Na Figura 4.1, podemos observar um excerto da tabela onde temos dados relativos ao ID, *Written Source*, *Recipe or Chapter number*, *Material* e *Colour*.

ID	Written Source	Recipe or Chapter number	Material	Colour
K1679_128	Kunckel 1679	128	Enamel	Red
K1679_63	Kunckel 1679		Powder	Yellow
K1679_124_Frit	Kunckel 1679		Glass	Colourless
K1679_124_CT	Kunckel 1679		Powder	Black

Figura 4.1: Excerto da tabela do *template* das receitas: ID, *Written Source*, *Recipe or Chapter number*, *Material* e *Colour*.

Estes dados representam o seguinte:

- **ID**: Representa o identificador da receita.
- ***Written Source***: É a fonte escrita onde se encontra a receita.
- ***Recipe or Chapter number***: Indica qual o número da receita ou do capítulo na fonte escrita.
- ***Material***: Representa o material a ser produzido.
- ***Colour***: Indica qual a cor esperada como resultado final da receita produzida.

Na Figura 4.2, é apresentado o excerto relativo ao procedimento a ser executado para cada receita. Os passos de uma determinada receita têm sempre o seu identificador da própria receita associada.

Procedure					
Steps	Step number				
ID		Steps for time calculation	Element	Recipe ID	Compounds Used
K1679_128	1	1	Frit	K1679_124_Frit	
	2	2	Pb-Sn Calx	K1679_63	
	3	3	Burned copper		Burned Copper
	3		Crocus Martis		Crocus Martis
	4	4	Calcined Tartarate	K1679_124_CT	
	4		Crocus martis		Crocus martis
K1679_63	1	1	Pb-Sn Calx		Lead-Tin Alloy
K1679_124_Frit	1	1	Frit		Na2CO3
					K2CO3
					MgO
					CaO
K1679_124_CT	1	1	Calcined Tartarate		Tartarate

Figura 4.2: Excerto da tabela do *template* das receitas: Procedimento de uma receita.

Nas colunas presentes neste excerto, para além do ID que já foi visto anteriormente, estão definidos:

- ***Step Number***: Indica o número do passo da receita.
- ***Steps for time calculation***: Representa os passos das receitas, tendo em consideração passos que podem ser realizados de forma independente, por exemplo, se os passos 1 e 2 da receita não dependerem um do outro podem ser realizados num mesmo dia e, portanto, podem ser vistos como um só passo. Assim sendo, para o cálculo do tempo de preparação da receita, será considerada a informação presente nesta coluna, em vez da coluna *Step Number*.
- ***Element***: Define o elemento a ser utilizado num dado passo, sendo que este pode ter de ser produzido através de uma outra receita. Caso isto aconteça, a coluna *Recipe number* encontra-se preenchida.
- ***Recipe ID***: Corresponde ao ID da receita para produzir o elemento.
- ***Compounds Used***: Representa os compostos utilizados em cada passo.

Na Figura 4.3, encontram-se dados relativos às quantidades a ser utilizadas para a produção de uma dada receita, bem como da ação a ser efetuada e do tempo máximo de duração do passo.

								Operator
Compound quantity (g)	Element quantity (g)	Element quantity produced (g)	Temperature Programme	Max temperature (ºC)	Max Time for calculation (min)	Colour	Action	Y/N
	30			1100	300		Melting	N
	1			1100	60		Stiring	Y
0,3				1100	30		Stiring	Y
0,3				1100			Stiring	Y
	1,2			1100	30		Stiring	Y
0,1				1100				
50		50		650	960		Calcination	N
53		200		1200	960		Melting	N
2,94								
8,36								
14,94								
20		20		600	120		Calcination	N

Figura 4.3: Excerto da tabela do *template* das receitas: *Compound quantity (g)*, *Element quantity (g)*, *Element quantity produced (g)*, *Max temperature (ºC)*, *Max Time for calculation (min)*, *Action* e *Operator*.

As colunas relevantes para a computação dos horários são as seguintes:

- ***Compound quantity (g)***: Representa a quantidade em gramas de compostos a serem utilizados. Esta medida só é utilizada quando uma receita só tem um passo.
- ***Element quantity (g)***: Define a quantidade em gramas do elemento a ser utilizado. Esta medida só é utilizada quando uma receita tem mais do que um passo.
- ***Element quantity produced (g)***: Representa para as subreceitas, a quantidade de elemento que é possível produzir com a realização dessa receita.
- ***Max temperature (ºC)***: Temperatura máxima em graus Celsius que um dado passo está no forno.
- ***Max Time for calculation (min)***: Tempo máximo em minutos que um dado passo pode demorar.
- ***Action***: Tipo de ação a ser realizada (fundir, misturar, ...).
- ***Operator***: Define se para realizar um dado passo é necessário estar presente um membro da VICARTE para o executar.

5

Implementação da aplicação

Neste capítulo será apresentado o processo de implementação da aplicação. Primeiramente, será apresentada uma overview da arquitetura do sistema. Seguidamente, serão descritas as várias componentes da aplicação, nomeadamente o desenho da base de dados, o modelo de dados utilizado para mapear os predicados do clingo, em classes do Python, as duas formas de planeamento de receitas e, por fim, a interface da aplicação.

Nesta dissertação o objetivo era desenvolver uma aplicação que permitisse a criação de horários com base em dados de receitas.

Esta aplicação funciona da seguinte maneira:

- Primeiro o utilizador decide através da interface da aplicação que receitas tenciona realizar, sendo que os dados relativos à receitas e aos fornos têm de estar previamente importados.

- De seguida, com base na pesquisa que o utilizador realizou, são efetuadas consultas à base de dados, de forma a obter os dados relevantes.

- Os dados obtidos com as consultas serão convertidos para um formato que seja processável para serem utilizados pelo programa ASP.

- Depois dos dados estarem no formato processável, é executado o programa ASP que irá permitir encontrar o planeamento, tendo em conta todas as restrições impostas relativamente aos fornos e às receitas.

- Quando for obtida a solução, o planeamento resultante das receitas será apresentado na interface gráfica da aplicação.

Na secção seguinte, será explicada em mais detalhe a arquitetura da aplicação.

5.1 Arquitetura da aplicação

A arquitetura da aplicação foi dividida em 3 camadas, tal como se pode observar na Figura 5.1: a camada de apresentação, a camada lógica e a camada de dados.

O utilizador irá interagir através da camada de apresentação, que foi desenvolvida através do Kivy e KivyMD. Ambas são bibliotecas Python e permitem desenvolver aplicações para diversas plataformas.

A escolha do Python tanto para o desenvolvimento do *frontend*, como para o do *backend* deveu-se a vários fatores. O Python é uma linguagem *open-source* simples e fácil de se aprender [5]. Permite desenvolver aplicações para várias plataformas e é utilizado em diversos domínios, como por exemplo, desenvolvimento web e aprendizagem automática. É por isso uma linguagem bastante versátil, utilizada tanto por engenheiros, matemáticos e cientistas. O Python possui ainda um grande conjunto de bibliotecas *standard* que facilitam a escrita de programas, contribuindo para a modularidade dos mesmos, visto que o programador pode utilizar o mesmo módulo, sem necessidade de reescrever o código. Além disso, existem diversas Python *frameworks* que podem auxiliar no processo de implementação.

Tendo ainda em consideração que o sistema Clingo permite a integração desta linguagem de forma bastante simples através da biblioteca Clorm, a implementação da aplicação em Python tornou-se uma opção bastante viável.

Na camada lógica são realizados os mapeamentos dos predicados ASP em objetos do Python através da biblioteca Clorm. Estes mapeamentos são necessários, uma vez que os dados obtidos através das consultas à base de dados, não estão num formato que possa ser utilizado diretamente no programa ASP, sendo que por isso têm de ser processados primeiro. Após estabelecidos os mapeamentos, é possível criar as instâncias com base na pesquisa realizada pelo utilizador. Estas instâncias são enviadas para o sistema Clingo, onde será executado o programa ASP correspondente de forma a encontrar a solução.

Todos os dados relativos às receitas e aos respetivos procedimentos, fornos e cores, estão guardados numa base de dados MySQL que corre no servidor *localhost*. Esta base de dados encontra-se na camada de dados.

Nas secções seguintes, serão explicadas em mais detalhe as várias componentes da aplicação.

5.2 Desenho da base de dados

Os dados das receitas encontram-se numa tabela dum ficheiro excel. Estes dados são necessários não só para o funcionamento da aplicação, como também são utilizados pelos membros da VICARTE para realizarem as receitas da forma mais autónoma possível. Assim sendo, na aplicação para simplificar, foi necessário ler os dados desse ficheiro em formato .csv.

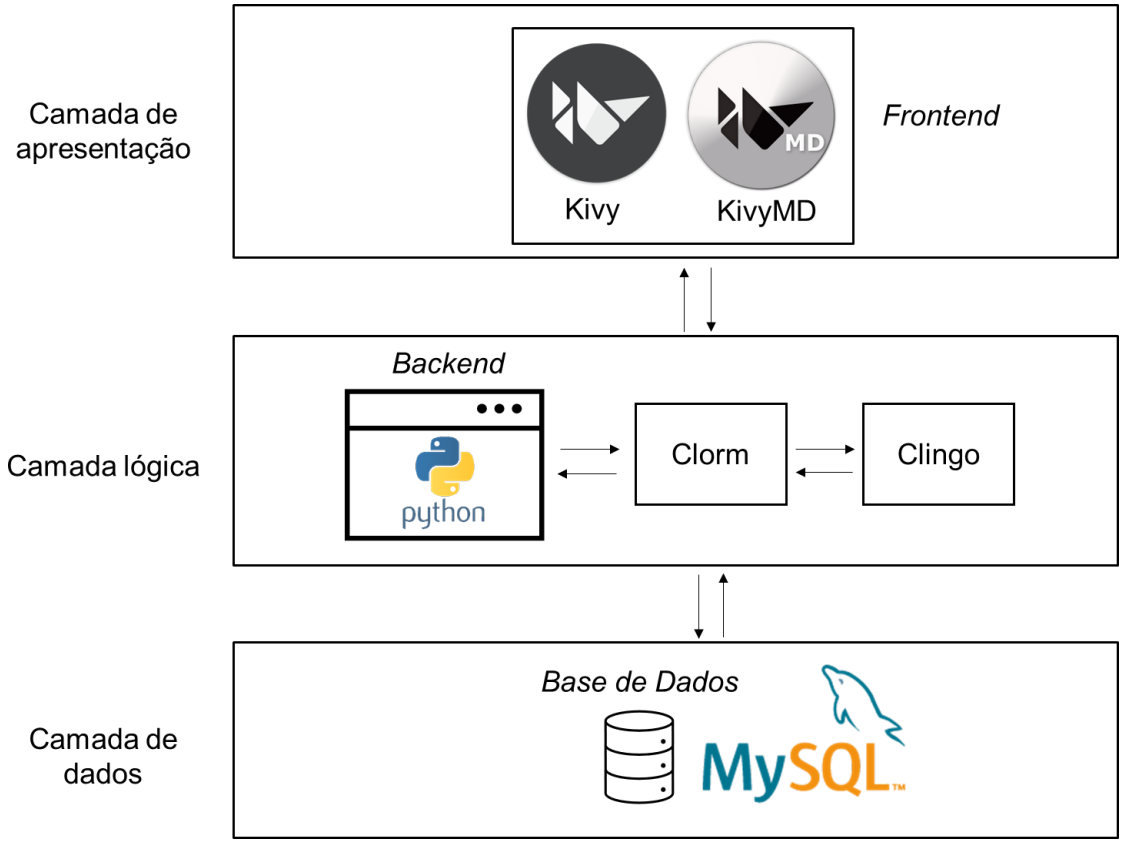


Figura 5.1: Arquitetura da aplicação.

Para se evitar ler os ficheiros de dados sempre que se abria a aplicação, foi decidido guardar os dados numa base de dados. Desta forma, não só o acesso aos dados é mais fácil, como também simplifica na resolução do problema, uma vez que se podem realizar consultas à base de dados para obter a informação necessária.

Para o desenho da base de dados foi realizado um Modelo de Entidade-Relações. Nesse modelo, estão definidas dez entidades e dez relações entre entidades. Para representar as receitas, foi criada a entidade **Recipe**, que tem como atributos o identificador da receita, a fonte escrita a que pertence, o número da receita ou capítulo, a quantidade em gramas que é produzida com a receita e, por último, um atributo para representar se a receita já foi produzida anteriormente.

Para definir o material e a cor das receitas, foram criadas as entidades **Material** e **Subcolour**. A entidade **Material** é definida pelo nome do material, enquanto a entidade **Subcolour** é definida pelo nome da subcor da receita. Isto deve-se ao facto das receitas terem cores específicas, por exemplo, *Opaque Red*. Uma vez que a pesquisa de receitas se realiza pela cor geral, no caso do exemplo referido, pela cor *Red*, a entidade **Subcolour** está associada à entidade **Colour**, onde estão representadas todas as cores gerais. As receitas por sua vez estão associadas com **Subcolour** e, por isso, desta forma é possível encontrar mais facilmente as receitas relevantes para uma pesquisa por cor. É de referir ainda que

as entidades **Material** e **Colour** foram criadas com o intuito de evitar erros de escrita.

Os vários passos das receitas são guardados através da entidade fraca **Step**. Desta forma, a entidade será representada pelo identificador da receita e pelo número do passo, visto que a entidade forte é a entidade **Recipe**. Para além desses atributos, ainda estão definidas a temperatura máxima a que a receita vai ao forno nesse passo, o que influencia em que fornos pode ser realizado, a duração do mesmo e um atributo para identificar se o passo pode ser realizado sem a presença de um elemento da VICARTE.

Em cada passo são efetuadas ações e são usados compostos ou elementos com uma dada quantidade. Assim, foram criadas entidades e as respetivas relações para representar estes conceitos. É ainda de referir que existe uma relação para representar os passos que têm uma subreceita associada. Isto acontece quando os elementos necessários para a realização dum passo da receita têm a sua própria receita. As subreceitas devem ser realizadas antes do passo que necessita delas.

Finalmente, para guardar os dados relativos aos fornos e aos cadinhos, existem as entidades **Furnace** e **Melting Pot**. Enquanto a entidade **Furnace** tem como atributos o identificador do forno e a temperatura máxima do mesmo, a entidade **Melting Pot** apenas tem como atributo a capacidade máxima do cadinho. Existe ainda uma relação para representar quantos cadinhos de cada capacidade cabem em cada forno, visto que pode existir mais do que uma amostra nas mesmas condições no forno. O número de amostras no forno depende não só do tamanho do forno, como também do tamanho dos cadinhos utilizados.

O Modelo de Entidade-Relações resultante encontra-se na Figura 5.2. De seguida, serão apresentadas algumas decisões que foram tomadas em relação ao mesmo.

Este modelo não tem uma relação entre os fornos e os passos das receitas. Apesar de à partida ser mais intuitivo existir essa relação, onde cada passo estaria associado a um forno que permitisse realizar o passo, isto não seria uma boa ideia. De facto, não é necessário que essa informação esteja presente na base de dados, uma vez que o programa ASP, vai permitir estabelecer a relação entre os fornos que tenham uma temperatura máxima que permitam realizar o passo. É de referir que uma receita pode estar atribuída a fornos diferentes pelo facto de a temperatura do passo permitir escolher entre dois fornos ou pelo facto de se ter de mudar de forno devido à ação que é realizada num passo. Desta forma, optou-se por não ter esta relação na base de dados, por causa dos passos que podem ser atribuídos a mais do que um forno, visto que para o planeamento final, um passo da receitas fica atribuído a um único forno.

Relativamente aos elementos e aos compostos utilizados num passo, como já foi referido, existem duas relações que traduzem esta ideia. Justifica-se a existência dessas relações para representar as quantidades utilizadas para os elementos e para os compostos, visto que estes são tratados de forma distinta. Os elementos utilizados num passo provém da realização de subreceitas, enquanto isso não acontece com os compostos. Desta forma, optou-se por não utilizar uma única relação, tendo em consideração que existem receitas que utilizam nuns passos a quantidade do elemento e noutros a quantidade do composto.

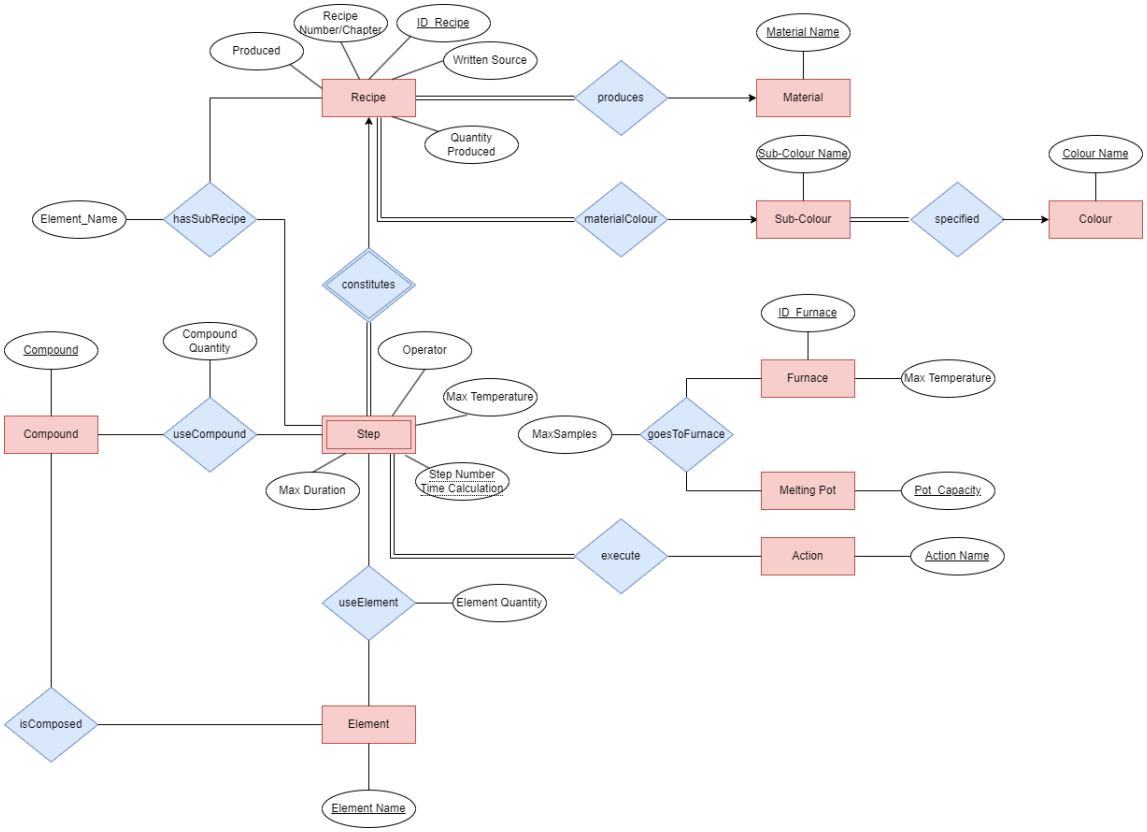


Figura 5.2: Modelo Entidade-Relações da base de dados

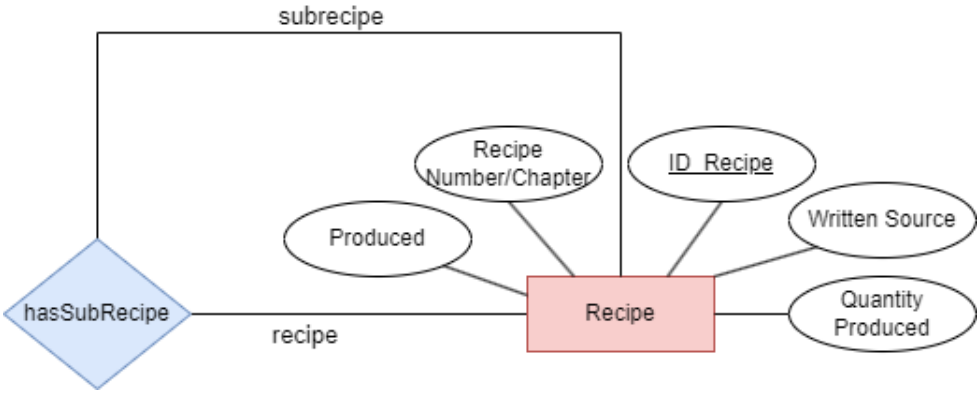


Figura 5.3: Relação reflexiva entre o conjunto de entidades **Recipe**.

Assim, com esta representação na base de dados, estes conceitos ficam separados de uma forma clara.

Em relação à representação da ideia de uma receita poder ter subreceitas, poderia-se ter sido utilizada uma relação reflexiva, ou seja, uma relação entre a mesma entidade, neste caso **Recipe**. Esta proposta encontra-se na Figura 5.3, mas apesar de expressar que uma receita tem subreceitas, não se consegue saber em que passo da receita é que as subreceitas são utilizadas.

Visto que a relação reflexiva não era uma opção viável, optou-se por utilizar uma

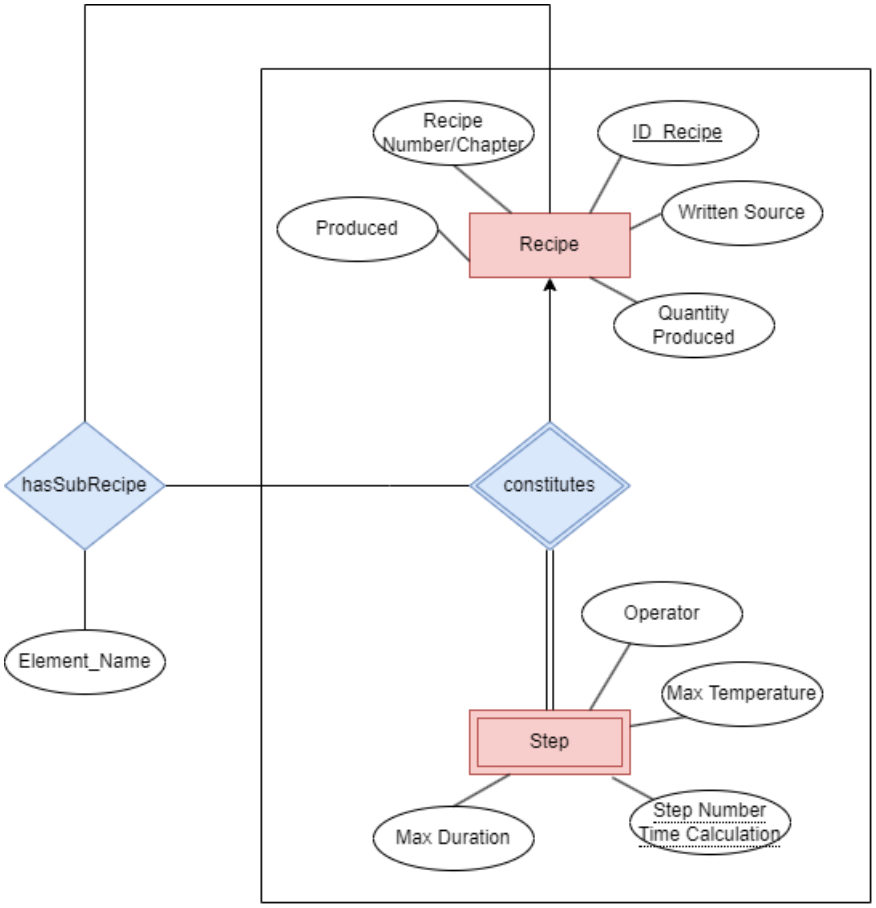


Figura 5.4: Agregação entre a relação *constitutes* e o conjunto de entidades **Recipe**.

agregação, que pode ser encontrada na Figura 5.4. Basicamente, a agregação define que as receitas são constituídas por passos e que cada combinação receita/passo pode estar associada a outra receita. Desta forma, seria possível associar o passo onde existe a subreceita. Para simplificar, visto que a relação entre **Step** e **Recipe** é uma relação de cardinalidade muitos-para-um, o que significa que uma receita pode ter vários passos, mas um passo pertence a uma receita, a relação *hasSubrecipe* ficou definida entre as entidades **Recipe** e **Step**. Isto deveu-se ao facto de se conseguir identificar a partir do passo a receita à qual ele pertence.

A partir do Modelo de Entidade-Relações resultante, foi obtido o esquema relacional, que depois foi simplificado de maneira a eliminar relações redundantes. O esquema simplificado encontra-se no Apêndice A e o modelo relacional no Apêndice B. Desta forma, foram obtidas as tabelas da base de dados, onde os dados serão posteriormente guardados num sistema de base de dados.

Existem diversos sistemas de base de dados, mas optou-se por utilizar o sistema de base de dados MySQL, visto que para a implementação desta solução não existia nenhuma característica que exigisse o uso de outro sistema.

Atualmente, o MySQL é utilizado por grandes empresas como o Facebook, LinkedIn, Netflix ou Uber [20].

Uma das vantagens do uso de MySQL é que permite o uso de uma grande variedade de linguagens, entre elas estão, por exemplo, Python, C, C++ ou Java [20]. Visto que a aplicação foi desenvolvida em Python, podíamos beneficiar disto. Além disso, é uma base de dados *open source*.

Este sistema de gestão de base de dados é eficiente, seguro e existe muita documentação sobre o mesmo. É de referir ainda que o MySQL é fácil de se instalar e de se utilizar.

O MySQL oferece ainda uma grande disponibilidade. Em caso de falha, é possível lidar com a mesma através da replicação dos dados. Caso seja necessário recuperar, existem mecanismos de recuperação (*checkpoints* e *rollbacks*).

Existiam duas opções para a base de dados: correr num servidor ou correr em *localhost*. Foi decidido que a base de dados iria correr em *localhost*, em vez do servidor, uma vez que cada utilizador pode precisar de alterar dados das receitas já existentes na base de dados, como a temperatura ou quantidades dos elementos/compostos utilizados, para realizar várias experiências no seu trabalho. Assim sendo, foi decidido que o ficheiro dos dados das receitas iria conter os dados que são comuns a todos, podendo assim existir variações locais entre cada um deles.

Os utilizadores a partir da aplicação vão precisar de importar os ficheiros .csv com os dados necessários, de forma a guardá-los na base de dados. Sempre que queiram adicionar ou alterar receitas na base de dados, terão de importar o ficheiro .csv correspondente. Este ficheiro não precisa de ter todas as receitas anteriores, basta aquelas que vão ser alteradas ou adicionadas, desde que o formato dos dados seja o do *template* existente para os ficheiros de dados das receitas.

5.3 Definição do modelo de dados

Depois do utilizador selecionar as receitas que pretende realizar através da interface da aplicação, são realizadas consultas à base de dados. Estes dados contém informações necessárias para que seja possível executar o programa ASP. Deste modo, os dados obtidos têm de ser processados de forma a que possam ser utilizados no mesmo, visto que estão num formato que não pode ser utilizado diretamente no programa ASP.

Para resolver essa situação, utilizou-se o Clorm que providencia uma interface ORM para o sistema Clingo. Como foi visto na secção 2.4.3, através do Clingo é possível obter os conjuntos de resposta para o problema em questão.

Desta forma, com o uso do Clorm, é possível traduzir os dados da base de dados em objetos Python, que serão utilizados para criar os predicados correspondentes no programa ASP. Este programa ASP é dado como *input* ao sistema Clingo, onde será executado para obter os conjuntos de resposta.

O programa ASP vai definir unicamente o domínio do problema, visto que as instâncias dos problemas estão em constante alteração. Assim sendo, não devem ser codificadas no próprio ficheiro ASP. Deste modo, o programa do Python vai definir as instâncias, que depois serão enviadas para o *solver* do Clingo para computar a solução.

Para definir o mapeamento dos predicados do Clingo para objetos do Python foi necessário definir o modelo de dados, tal como foi explicado na Secção 2.5.2, utilizando a classe Predicate do Clorm. Como tal, teve-se o cuidado dos nomes dos predicados serem exatamente os mesmos no programa Python e no programa ASP, bem como das declarações dos parâmetros de cada predicado terem exatamente o mesmo número de *fields* declarados exatamente pela mesma ordem. Além disto, para cada predicado foram especificados os tipo de dados dos parâmetros utilizando as classes **StringField** e **IntegerField**.

É ainda de referir que os predicados e os respetivos parâmetros definidos no modelo de dados estão conforme os dados presentes nas tabelas da base de dados.

Este modelo está representado na Listagem 5.1, onde foram definidos os seguintes mapeamentos para passar informação da base de dados para o programa ASP:

- **Recipe**: define as principais informações sobre a receita (identificador, fonte escrita, se já foi produzida anteriormente e a cor da receita).
- **Step**: representa passos das receita. É de referir que se existem passos da mesma receita que podem ser executados nas mesmas condições (com a mesma temperatura e ações), estes podem formar um conjunto, sendo por isso criado apenas um facto que terá a duração total desses passos. Desta forma evita-se a criação de vários factos de forma desnecessária.
- **HasSubrecipe**: representa as receitas que têm subreceitas num determinado passo;
- **Furnace**: define os fornos que estão disponíveis.
- **Day**: define o número máximo de dias que pode ser utilizado. Caso este número não seja definido pelos utilizadores, o número máximo de dias que será considerado poderá variar entre o número de receitas existentes e o número de blocos de cada receita.
- **Assign**: representa as atribuições de dias e fornos a cada facto **Step**.
- **PotsUsed**: representa o número de cadinhos que são necessários para realizar as receitas em cada **Step**.
- **Limit**: define uma estimativa do número mínimo de dias que é necessário para obter um planeamento para as receitas pretendidas.

Através deste modelo de dados, foi possível criar os factos necessários para cada instância. Na Listagem 5.2, é apresentado um exemplo de uma instância com 5 receitas. Nesta instância, estão representados os quatros fornos disponíveis: "Forno A (Blue)", "Forno B (6)", "Forno C (4)"e "Forno D (redondo)". Para cada um deles, nas linhas 1 a 5, define-se a temperatura máxima do forno e a capacidade, por exemplo, o forno A tem temperatura máxima de 1300ºC e tem capacidade para colocar 4 cadinhos de 50g.

```
1  from clorm import Predicate, IntegerField, StringField
2
3  class Recipe(Predicate):
4      ID_Recipe=StringField
5      Written_Source=StringField
6      Produced=StringField
7      Colour_Name=StringField
8  class Step(Predicate):
9      ID_Recipe=StringField
10     Step_Number=IntegerField
11     Max_Duration=IntegerField
12     Recipe_Max_Temp=IntegerField
13     Action_Name=StringField
14     Operator=StringField
15     Times=IntegerField
16 class HasSubrecipe(Predicate):
17     ID_Recipe=StringField
18     Step_Number= IntegerField
19     ID_Subrecipe=StringField
20 class Furnace(Predicate):
21     ID_Furnace=StringField
22     Max_Temperature=IntegerField
23     Capacity=IntegerField
24 class Assign(Predicate):
25     IDRecipe=StringField
26     StepNumber=IntegerField
27     ID_Furnace = StringField
28     Day=IntegerField
29     Times = IntegerField
30 class Day(Predicate):
31     Day= IntegerField
32 class PotsUsed(Predicate):
33     IDRecipe=StringField
34     BlockNumber=IntegerField
35     Pots=IntegerField
36     Times=IntegerField
37 class Limit(Predicate):
38     Limit=IntegerField
```

Listagem 5.1: Modelo de dados para mapear predicados do clingo em classes do Python

Os passos das receitas estão definidos nas linhas 8 a 12. Para exemplificar, a receita "N97"tem um único passo onde se executa a ação de *Stiring* durante 300 minutos a uma temperatura máxima de 1200ºC.

Nas linhas 14 a 17, estão definidas as receitas que têm subreceitas num determinado passo. No caso do exemplo apresentado, a receita "N97"no passo 1 tem como subreceita "N93_Base Enamel".

Esta receita nesse passo irá precisar apenas de utilizar 1 cadinho de 50g para colocar a quantidade necessária que será produzida. O número de cadinhos que serão necessários em cada passo das receitas estão definidos nas linhas 20 a 25.

Por último, na linha 27 definiu-se que neste exemplo o número máximo de dias para realizar estas receitas seriam 5 dias.

Na linha 28 está representado o limite mínimo de dias necessários para se encontrar um planeamento para as receitas definidas.

Listagem 5.2: Factos criados para a instância com 5 receitas

```
1  %furnace, MaxTemp, MaxPots
2  furnace("Forno A (Blue)",1300,4).
3  furnace("Forno B (6)",1200,4).
4  furnace("Forno C (4)",1000,2).
5  furnace("Forno D (redondo)",900,1).
6
7  %recipeID, BlockNumber, MaxDuration, MaxTemp, Action,Operator, N
8  step("M110",1,610,1100,"Melting","N",1).
9  step("N97",1,300,1200,"Stiring","Y",1).
10 step("N98",1,300,1200,"Stiring","Y",1).
11 step("N99",1,300,1200,"Stiring","Y",1).
12 step("N93_Base Enamel",1,960,1200,"Melting","N",1).
13
14 %RecipeID, BlockNumber, SubrecipeID
15 hasSubrecipe("N97",1,"N93_Base Enamel").
16 hasSubrecipe("N98",1,"N93_Base Enamel").
17 hasSubrecipe("N99",1,"N93_Base Enamel").
18
19 %RecipeID,BlockNumber,Pots, N
20 potsUsed("M110",1,2,1).
21 potsUsed("N97",1,1,1).
22 potsUsed("N98",1,1,1).
23 potsUsed("N99",1,1,1).
24 potsUsed("K1679_124_CT",1,1,1).
25 potsUsed("N93_Base Enamel",1,3,1).
26
27 day(1..5).
28 limit(3).
```

5.4 Planeamento de receitas

Existem duas formas distintas de realizar o planeamento de receitas na VICARTE. Assim sendo, existem duas versões do planeamento de receitas em programas ASP.

Na primeira versão, pretende-se que num dia, no máximo, se possa realizar uma receita por forno. Por norma, esta versão é utilizada quando os membros da VICARTE pretendem testar novas receitas. Como tal, para tentar que não existam fatores que possam influenciar o resultado final das experiências, num dia só pode existir no máximo uma receita em cada forno.

De seguida, será explicado o código ASP para esta versão do planeamento. O código que for apresentado nesta secção é o código resultante após a realização de várias melhorias. Estas melhorias serão descritas e testadas no capítulo seguinte.

No código abaixo, podemos observar os três geradores utilizados para resolver o problema, onde cada um vai atribuir um passo a um dia e a um forno que permita realizar esse passo. O primeiro gerador é para os fornos A e B, o segundo para o forno C e terceiro para o forno D. As atribuições dos fornos C e D são feitas separadamente, visto que o forno C é só para passos de receitas que executem a ação *Annealing* e o forno D para os que executam a ação *Calcination*. Esta solução foi criada para esta situação concreta, tendo em consideração as restrições dos fornos.

```
1 %geral
2 1{assign(IDRecipe,StepNumber,F,D,T):furnace(F,FurnaceMaxTemp,_),
      day(D),RecipeMaxTemp<=FurnaceMaxTemp,F!="Forno D (redondo)",F
      !="Forno C (4)"}1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,
      Action,_,T), Action!="Calcination", Action!="Annealing".
3 %forno D
4 1{assign(IDRecipe,StepNumber,"Forno D (redondo)",D,T):furnace("
      Forno D (redondo)",FurnaceMaxTemp,_),day(D),RecipeMaxTemp<=
      FurnaceMaxTemp}1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,"
      Calcination",_,T).
5 %forno C
6 1{assign(IDRecipe,StepNumber,"Forno C (4)",D,T):furnace("Forno C
      (4)",FurnaceMaxTemp,_),day(D),RecipeMaxTemp<=FurnaceMaxTemp
      }1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,"Annealing",_,T)
      .
```

As restrições das linhas 7 e 8, fazem com que não seja possível o forno B estar livre (e o forno A ocupado) num dia e o forno A estar ocupado (e o forno B livre) noutro dia, quando uma das receitas que está no forno A tiver uma temperatura que permita estar no forno B. Estas restrições permitem gerir melhor o uso dos fornos, evitando assim estar a desperdiçar dias. Isto significa que se em dois dias diferentes o forno A está ocupado e o forno B está livre, então as receitas que aí são realizadas têm temperaturas

que só são possíveis de atingir com a utilização do forno A. A ideia destas restrições foi tentar quebrar as simetrias quando existe possibilidade de escolher entre dois fornos, mais concretamente, entre o forno A e o forno B.

A linha 9 impõe a restrição que as atribuições das receitas têm de começar no dia 1. Tem de se descartar modelos que não cumpram esta condição, visto que quando se tenta minimizar o número de dias utilizados no plano, existe a hipótese de uma das soluções válidas começar por exemplo no dia 5.

```
7 :-not assign(_,_,"Forno B (6)",D1,_),assign(IDRecipe,StepNumber
      ,"Forno A (Blue)",D1,T), step(IDRecipe,StepNumber,_,
      RecipeMaxTemp,_,_,T),RecipeMaxTemp<=1200, not assign(_,_,"
      Forno B (6)",D2,_),assign(_,_,"Forno A (Blue)",D2,_), D1<D2.
8 :-not assign(_,_,"Forno B (6)",D1,_), assign(_,_,"Forno A (Blue)
      ",D1,_), not assign(_,_,"Forno B (6)",D2,_),assign(IDRecipe2,
      StepNumber2,"Forno A (Blue)",D2,T2), step(IDRecipe2,
      StepNumber2,_,RecipeMaxTemp2,_,_,T2), RecipeMaxTemp2<=1200,
      D1<D2.
9 :- not assign(_,_,_,1,_).
```

As restrições das linhas 10 a 13 definem condições entre duas atribuições. A primeira garante que num dia no mesmo forno apenas está uma receita.

A mesma receita pode ter números de vez diferentes nos casos em que a quantidade final produzida da mesma não cabe um único forno. Imaginemos, por exemplo, uma receita que tenha uma quantidade final produzida de 300g e que possa ser atribuída tanto ao forno A, como ao forno B, devido à sua temperatura. Tendo em conta que esses fornos têm apenas capacidade para fazer 200g, esta receita vai ter de ser realizada(ou dividida) por duas vezes. Desta forma, receitas com números de vez diferentes, não podem ser atribuídas no mesmo dia no mesmo forno. A segunda restrição expressa o facto de uma receita não poder ser realizada no mesmo dia e no mesmo forno, caso o número da vez seja diferente. É preciso ainda garantir, através da linha 13, que receitas que tenham o mesmo número de vez sejam realizadas no mesmo dia.

```
10 :- assign(IDRecipe1,_,F,D,_), assign(IDRecipe2,_,F,D,_),
      IDRecipe1!=IDRecipe2.
11 :- assign(IDRecipe,_,F,D,T1), assign(IDRecipe,_,F,D,T2),T1<T2.
12
13 :- assign(IDRecipe,_,_,D1,T), assign(IDRecipe,_,_,D2,T), D1<D2.
```

No planeamento final as subreceitas vão ter de ser realizadas antes dos passos que as utilizam. A restrição da linha 14 garante que esta condição é verdadeira, uma vez que permite eliminar modelos onde as subreceitas são realizadas primeiro que as receitas.

A restrição da linha 16 garante que dentro de um forno não estão mais cadinhos do que a capacidade máxima do forno.

Com a intenção de minimizar o número de dias utilizados para realizar o planeamento, na linha 18 é calculado esse número (N) e na linha 19 esse valor vai ser minimizado. Existe ainda outra restrição que permite estabelecer um limite inferior para o número de dias necessários para se encontrar um planeamento. Esta restrição encontra-se presente na linha 21.

Finalmente, a linha 23 representa o facto das receitas serem atribuídas em dias consecutivos, de forma a acabar mais cedo.

```
14 :- hasSubrecipe(IDRecipe,StepNumber1,IDSubrecipe),assign(
       IDRecipe,StepNumber1,_,D1,_), assign(IDSubrecipe,StepNumber2,
       _,D2,_),D2>=D1.
15
16 :-S=#sum{P,IDRecipe,BN: potsUsed(IDRecipe,BN,P,_),assign(IDRecipe
       ,BN,F,D,_)},furnace(F,_,C),day(D),S>C.
17
18 numberDays(N):-N=#count{D: assign(_,_,_,D,_)}.
19 #minimize {N@2: numberDays(N)}.
20
21 :-numberDays(N),N<L,limit(L).
22
23 :- assign(_,_,_,D1,_), assign(_,_,_,D2,_),numberDays(N),D1<D2,
       D2-D1>=N.
```

O programa completo(recipesV1.lp) encontra-se no Apêndice C.

Na segunda versão do planeamento, é permitido realizar mais do que uma receita num dia e num forno, desde que sejam válidas as seguintes condições:

- Num dia não podem estar no mesmo forno receitas com cores diferentes;
- Se no mesmo forno estiverem passos de receitas que executem a ação *stiring*, então esses passos têm de pertencer à mesma fonte escrita;
- Se os passos das receitas têm exatamente a mesma duração e a mesma temperatura, então podem ser realizados no mesmo dia e no mesmo forno.

Estas condições representam a forma de como as várias receitas costumam ser realizadas na VICARTE.

O programa desta versão (recipesV2.lp) utiliza o mesmo código que o programa da primeira versão, com exceção da restrição que não permitia que num dia fossem realizadas mais do que uma receita num forno. Para além disso, foram adicionadas restrições para definir as condições anteriores.

No código da Listagem 5.3, a primeira linha codifica o facto de no mesmo dia não poderem estar no mesmo forno receitas com cores diferentes.

Na terceira linha, está codificado o facto de no mesmo dia, se estiverem passos de receitas onde seja realizada a ação *stiring*, então esses passos têm de pertencer à mesma fonte escrita.

As restrições das linhas cinco a nove, codificam o facto de os passos com a mesma temperatura e a mesma duração poderem ser realizados no mesmo dia no mesmo forno. Na linha cinco, são eliminados os modelos onde no mesmo dia e no mesmo forno estão passos com durações diferentes, enquanto na linha sete, são descartados modelos onde as temperaturas são diferentes.

A restrição da linha nove, permite eliminar modelos onde receitas com a mesma temperatura e a mesma duração estejam no mesmo dia, mas em fornos diferentes.

A restrição da linha onze permite descartar modelos em que a mesma receita foi atribuída ao mesmo dia e ao mesmo forno, mas com números de vez diferentes.

O código completo do programa recipesV2.lp encontra-se no Apêndice D.

Listagem 5.3: Restrições adicionadas ao programa ASP recipesV2.lp

```
1  :- assign(IDRecipe1,_,F,D,_), assign(IDRecipe2,_,F,D,_), recipe(
       IDRecipe1,_,_,Colour1),recipe(IDRecipe2,_,_,Colour2),
       IDRecipe1!=IDRecipe2, Colour1!=Colour2.
2
3  :- assign(IDRecipe1,BN1,F,D,_), assign(IDRecipe2,BN2,F,D,_),
       step(IDRecipe1,BN1,_,_,"Stiring",_,_),recipe(IDRecipe1,
       WrittenSource1,_,_),recipe(IDRecipe2,WrittenSource2,_,_),
       WrittenSource1!=WrittenSource2,IDRecipe1!=IDRecipe2.
4
5  :- assign(IDRecipe1,BN1,F,D,_), assign(IDRecipe2,BN2,F,D,_),step
       (IDRecipe1,BN1,MaxDur1,_,_,_,_),step(IDRecipe2,BN2,MaxDur2,_,
       _,_,_) ,IDRecipe1!=IDRecipe2,MaxDur1<MaxDur2.
6
7  :- assign(IDRecipe1,BN1,F,D,_), assign(IDRecipe2,BN2,F,D,_),step
       (IDRecipe1,BN1,_,MaxTemp1,_,_,_),step(IDRecipe2,BN2,_,
       MaxTemp2,_,_,_),IDRecipe1!=IDRecipe2,MaxTemp1<MaxTemp2.
8
9  :- assign(IDRecipe1,BN1,F1,D,_), assign(IDRecipe2,BN2,F2,D,_),
       step(IDRecipe1,BN1,MaxDur,MaxTemp,_,_,_),step(IDRecipe2,BN2,
       MaxDur,MaxTemp,_,_,_),IDRecipe1!=IDRecipe2,F1!=F2.
10
11 :- assign(IDRecipe,BN,F,D,T1), assign(IDRecipe,BN,F,D,T2),step(
       IDRecipe,BN,_,_,_,_,T1),step(IDRecipe,BN,_,_,_,_,T2),T1<T2.
```

5.5 Interface gráfica da aplicação

Nesta secção serão apresentadas as componentes da interface gráfica mais relevantes. A aplicação encontra-se na pasta partilhada em https://drive.google.com/drive/folders/1OPYrb96BJesHR0HPd5-4FdgxbDojUB1k?usp=sharing. As instruções para auxiliar no processo de instalação encontram-se no Apêndice E.

Uma vez que, por enquanto, o servidor MySQL corre localmente, para aceder à aplicação é necessário fornecer a palavra passe *root* do mesmo. Quando se entra pela primeira vez na aplicação é pedido para o utilizador introduzir a sua palavra passe do MySQL. Nas restante vezes, a aplicação utiliza por defeito a palavra passe que foi introduzida da primeira vez.

A página principal da aplicação encontra-se na Figura 5.5. A partir da mesma, é possível procurar planeamentos para receitas, importar dados das receitas, fornos e cores e alterar a palavra passe do MySQL.

Figura 5.5: Página principal da aplicação.

Caso um dia o utilizador decida mudar a sua palavra passe no MySQL, para continuar a utilizar a aplicação, deve alterá-la de seguida na aplicação.

Ao carregar no botão para importar dados, o utilizador tem de selecionar que tipo de

ficheiro vai importar, ou seja, se é um ficheiro com dados de receitas ou se é um ficheiro com os dados das cores e fornos. Depois, a partir do explorador de ficheiros é possível selecionar o ficheiro .csv pretendido. Como resultado, a base de dados será atualizada com os dados que constam no ficheiro.

5.5.1 Pesquisa de receitas

Para a efetuar a pesquisa das receitas pretendidas, foi implementado o ecrã presente na Figura 5.6.

Para filtrar a pesquisa, na lista de *dropdown* pode-se selecionar as opções de procura por cor, fonte escrita e material. Os planeamentos na VICARTE relativos a pesquisas por cor, são efetuados pela cor geral e não pela cor específica.

Além destas opções, pode-se pesquisar receitas pela quantidade de um elemento que exista atualmente feito, ou seja, pretende-se saber que receitas é que se podem realizar com essa quantidade disponível do elemento. Existe ainda a opção de pesquisar pelo número máximo de dias que se pretende que o planeamento tenha. Caso esta opção seja selecionada, o utilizador introduz o número de dias máximo pretendido. O planeamento pode ser apresentado utilizando menos dias que o máximo introduzido, caso este não precise de utilizar a totalidade dos dias para realizar as receitas pretendidas.

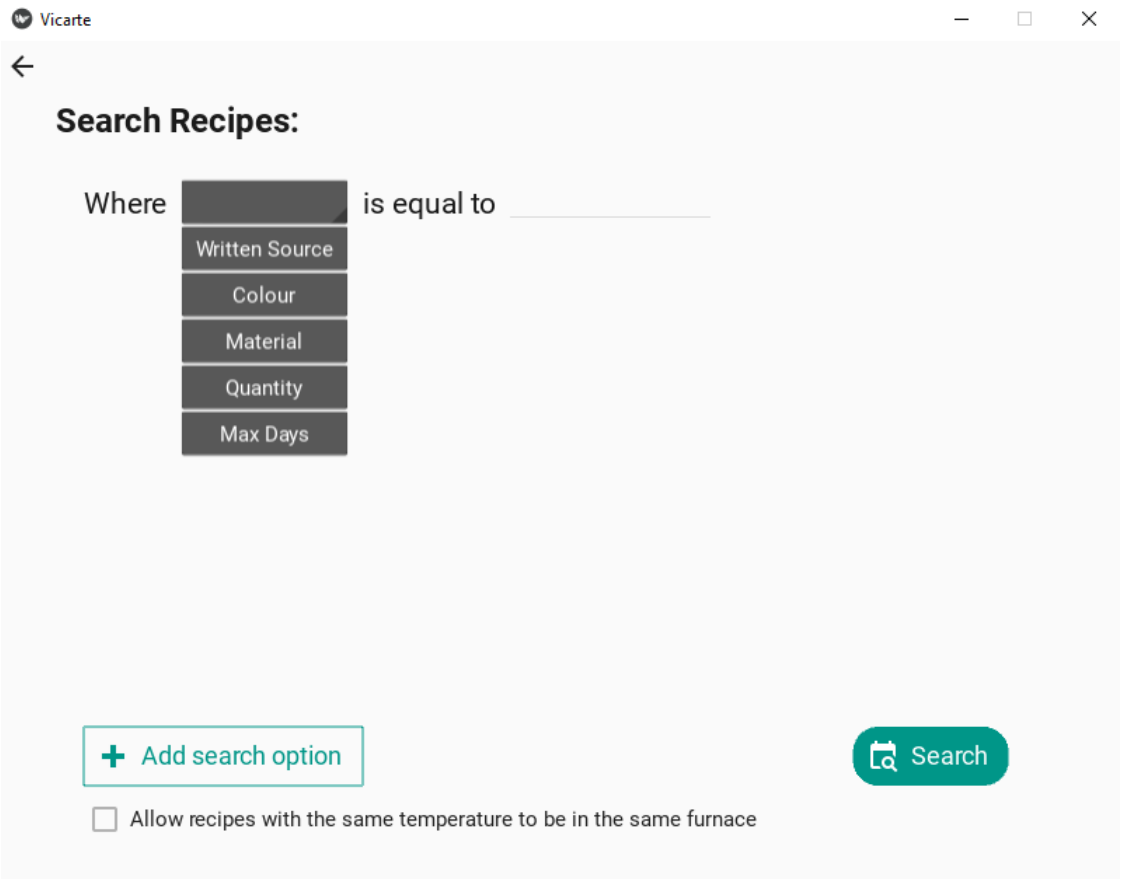


Figura 5.6: Componente da pesquisa de receitas - *dropdown*.

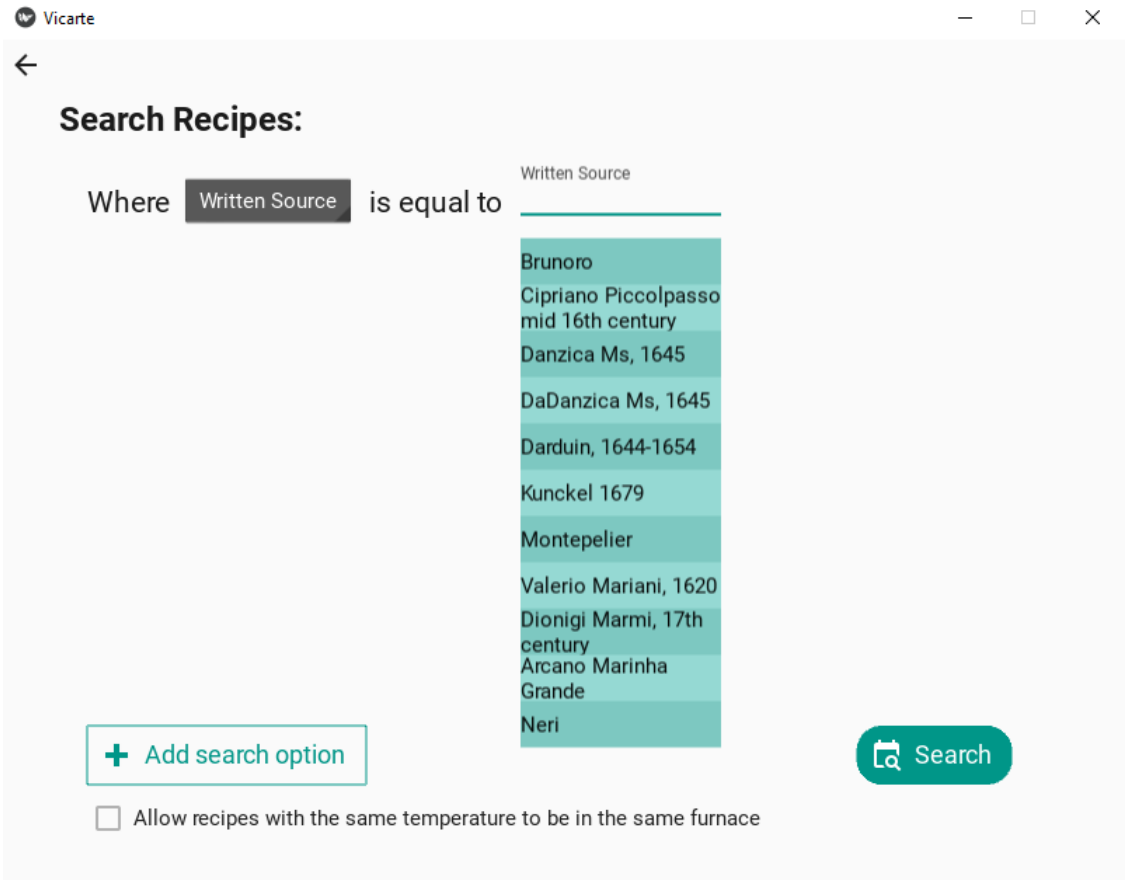


Figura 5.7: Componente da pesquisa de receitas - *text input* com pesquisa.

Imaginemos que se pretende realizar um planeamento para realizar receitas da fonte escrita "Kunckel 1679". Para isso será necessário selecionar a opção na lista de *dropdown* "*Written Source*". Para apresentar os resultados das fontes escritas existentes na base de dados podia-se ter utilizado outra lista de *dropdown*. No entanto, essa opção não seria muito viável, tendo em consideração que no futuro poderão existir centenas de receitas na base de dados e o utilizador teria de andar à procura da opção que pretende selecionar entre centenas.

Deste modo, implementou-se a apresentação dos resultados com uma mistura de *input* de texto e de uma lista de *dropdown*. Assim, o utilizador tanto pode selecionar o valor na lista, como pode escrever o que pretende selecionar. É ainda de referir que à medida que o utilizador escreve, as opções da lista serão atualizadas para corresponder à pesquisa pretendida. Pode-se observar esta implementação nas Figuras 5.7 e 5.8.

Nesta componente de pesquisa de receitas, pode-se ainda adicionar mais opções para filtrar as receitas através do botão "Add search option", sendo adicionada mais uma linha para filtrar, tal como se pode ver na Figura 5.9.

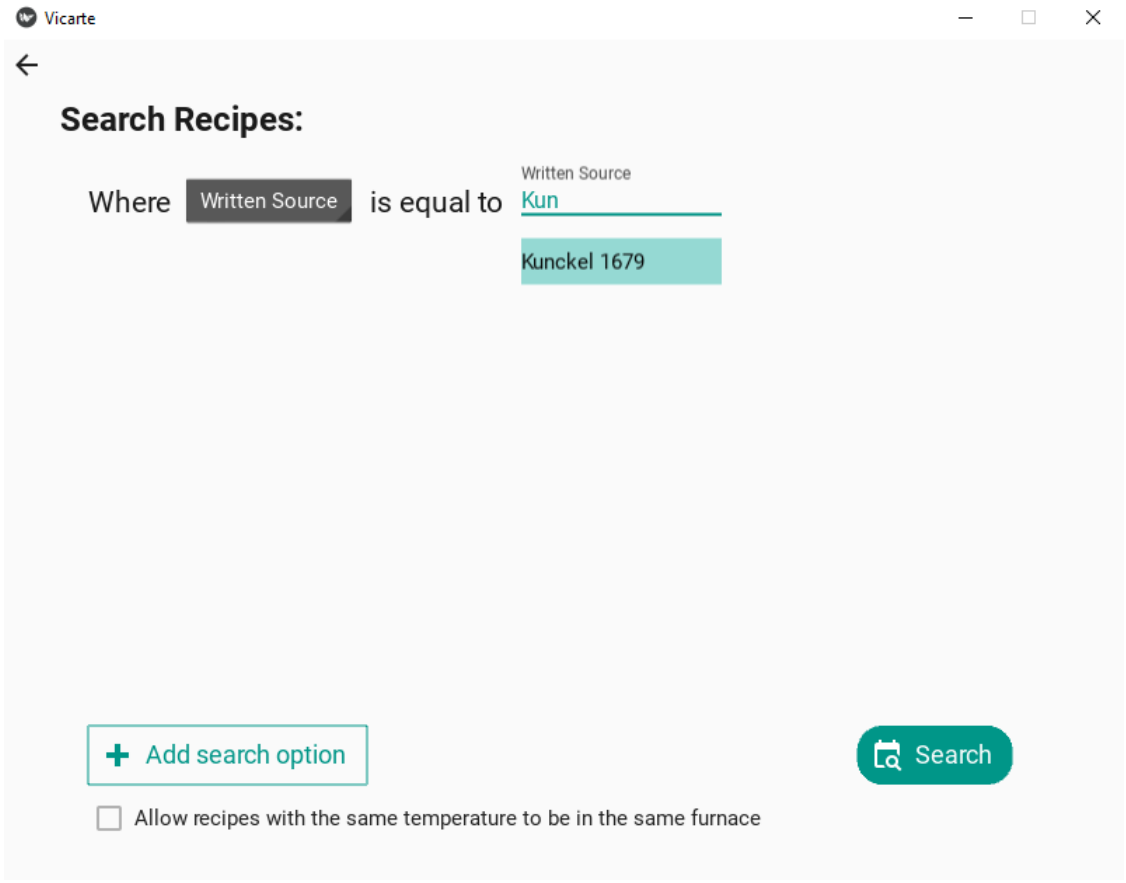


Figura 5.8: Componente da pesquisa de receitas - Alteração da lista de opções do *text input* com pesquisa.

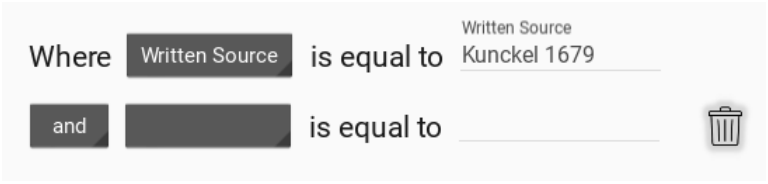


Figura 5.9: Componente da pesquisa de receitas - "Add search option".

Caso se pretenda encontrar um planeamento em que seja permitido num dia serem realizadas no mesmo forno mais do que uma receita (desde que estejam nas mesmas condições), basta selecionar a *checkbox* no fim do ecrã.

Finalmente, para realizar a pesquisa basta apenas carregar no botão "Search".

5.5.2 Apresentação do planeamento

Para a apresentação do planeamento, os resultados que foram obtidos através da execução do programa ASP, serão mostrados ao utilizador através de uma tabela para facilitar a leitura dos dados.

Nesta tabela, cada linha contém a seguinte informação:

- Identificação da receita a ser realizada;
- O dia em que a receita deve ser realizada;
- A hora de início e a duração dos passos que aí são realizados;
- O forno ao qual foi atribuída a receita;
- A quantidade produzida da receita.

Se o número de receitas que podem ser realizadas não cabe na totalidade na primeira página da tabela, pode-se navegar entre as várias páginas da mesma.

Caso o utilizador pretenda guardar os dados para visualizar melhor todas as receitas, pode fazê-lo a partir do botão "Export to csv file".

Imaginemos o caso de se obter o planeamento da Figura 5.10. Se o utilizador ao visualizar o planeamento decidir que afinal só quer realizar as receitas **K1679_124** e **K1679_125**, existe a possibilidade criar um novo plano a partir desse, selecionando primeiro as receitas que se pretendem realizar. Para encontrar um novo plano para as receitas selecionadas, basta carregar no botão "Find new plan". Do mesmo modo, caso o utilizador pretenda obter um planeamento sem as receitas **K1679_124** e **K1679_125**, bastaria selecionar todas as receitas com exceção dessas.

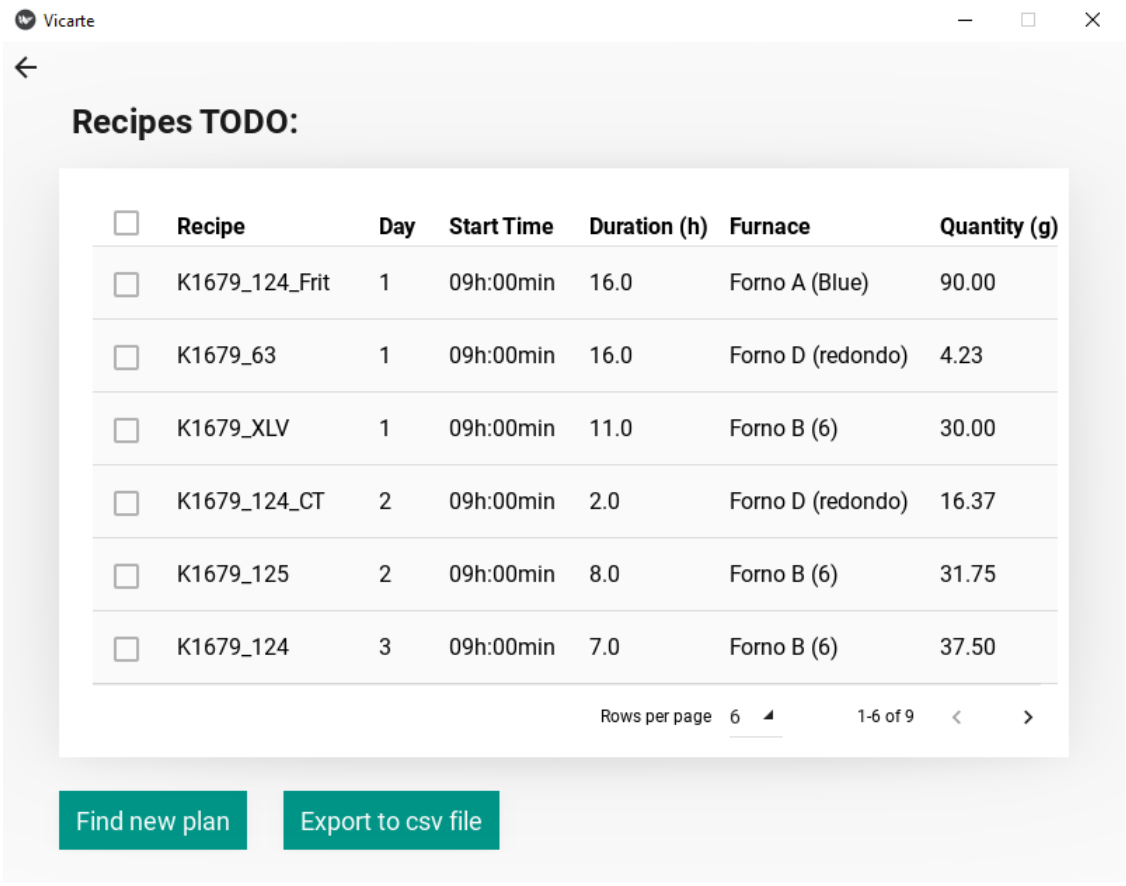


Figura 5.10: Componente da apresentação do planeamento.

6

Avaliação

Neste capítulo serão analisados os resultados de desempenho da aplicação, bem como dos resultados do questionário realizado aos membros da VICARTE que testaram a mesma.

6.1 Resultados do desempenho da aplicação

Até à obtenção da versão final do código dos dois programas ASP realizaram-se algumas melhorias no código. De seguida, vão ser descritas as mais significativas e serão apresentadas as partes do código que foram melhoradas.

Para simplificar, vamos considerar apenas o código da primeira versão do planeamento de receitas, visto que as alterações efetuadas no mesmo foram realizadas também no código segunda versão do planeamento.

Comecemos por definir as várias versões dos programas ASP que serão apresentadas:

- **encoding1.lp**: É o programa para a primeira versão do planeamento, sem otimizações;
- **encoding2.lp**: É o programa para a primeira versão do planeamento, com otimização no gerador;
- **recipes_V1.lp**: É o programa para a primeira versão do planeamento, com todas as otimizações aplicadas;
- **recipes_V2.lp**: É o programa para a segunda versão do planeamento, com todas as otimizações aplicadas;

O código inicial da primeira versão do planeamento (**encoding1.lp**) gera modelos onde não se tem em consideração a temperatura da receita e a temperatura máxima do forno, tal como se pode observar na Listagem 6.1. Como consequência, estão a ser gerados modelos onde as receitas são atribuídas a fornos que não têm uma temperatura máxima que permita realizar a receita. Desta forma, para garantir que as receitas são atribuídas

a fornos cuja temperatura máxima permita realizar a receita, existe uma restrição que permite eliminar modelos onde esta condição não se verifica.

Para além disso, o gerador atribui um passo a um forno, sem ter em consideração o facto dos passos com ações de *Annealing* e *Calcination*, irem para fornos específicos, respetivamente, para o forno C e para o forno D.

Assim, estão a ser gerados primeiro os modelos e, de seguida, através das restrições, são descartados os que não interessam, o que leva a um aumento no tempo de processamento.

Listagem 6.1: Código inicial do gerador do programa encoding1.lp

```
1 1{assign(IDRecipe,StepNumber,F,D,T):furnace(F,FurnaceMaxTemp,_),
      day(D)}1:-step(IDRecipe,StepNumber,_,_,_,_,T).
2
3 :-assign(IDRecipe,StepNumber,IDFurnace,_,_), furnace(IDFurnace,
      MaxTemperature,_),step(IDRecipe,StepNumber,_,RecipeMaxTemp,_,
      _,_), RecipeMaxTemp>MaxTemperature.
```

De forma reduzir o número de modelos gerados, melhorou-se o código substituindo o gerador que existia por três geradores: um gerador para os fornos A e B, um para o forno C e outro para o forno D. Além disso, em cada gerador as receitas são unicamente atribuídas a fornos que tenham temperatura suficiente para as realizar. Estas alterações encontram-se na Listagem 6.2 e foram guardadas no ficheiro **encoding2.lp**.

Listagem 6.2: Código melhorado com uso de 3 geradores

```
1 %geral
2 1{assign(IDRecipe,StepNumber,F,D,T):furnace(F,FurnaceMaxTemp,_),
      day(D),RecipeMaxTemp<=FurnaceMaxTemp,F!="Forno D (redondo)",F
      !="Forno C (4)"}1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,
      Action,_,T), Action!="Calcination", Action!="Annealing".
3 %forno D
4 1{assign(IDRecipe,StepNumber,"Forno D (redondo)",D,T):furnace("
      Forno D (redondo)",FurnaceMaxTemp,_),day(D),RecipeMaxTemp<=
      FurnaceMaxTemp}1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,"
      Calcination",_,T).
5 %forno C
6 1{assign(IDRecipe,StepNumber,"Forno C (4)",D,T):furnace("Forno C
      (4)",FurnaceMaxTemp,_),day(D),RecipeMaxTemp<=FurnaceMaxTemp
      }1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,"Annealing",_,T)
      .
```

Foram criadas cinco instâncias para avaliar o desempenho, onde em cada instância são adicionadas receitas à instância anterior (por exemplo, o ficheiro instance10.lp tem as receitas do ficheiro instance5.lp mais 5 receitas novas):

- **instance5.lp**: Tem 5 receitas (correspondem a um total de 5 blocos de passos criados).
- **instance15.lp**: Tem 10 receitas (correspondem a um total de 15 blocos de passos criados).
- **instance21.lp**:Tem 16 receitas (correspondem a um total de 21 blocos de passos criados).
- **instance26.lp**: Tem 21 receitas (correspondem a um total de 26 blocos de passos criados).
- **instance29.lp**: Tem 24 receitas (correspondem a um total de 29 blocos de passos criados).

A diferença existente entre o número de receitas e o número de blocos que são criados deve-se ao facto de algumas receitas terem de ser realizadas em vários fornos, devido às ações que são realizadas nalguns passos. Cada uma das 5 receitas que foram adicionadas no ficheiro **instance15.lp** têm de ser realizadas em 2 blocos de passos, uma vez que no procedimento dessas receitas existe um passo onde se realiza a ação de *Annealing*. Desta forma, enquanto que o primeiro bloco de passos pode ser realizado num dos fornos A ou B, o segundo bloco tem de ser efetuado no forno C.

A tabela 6.1 apresenta os resultados das melhorias realizadas. Os resultados apresentados foram obtidos utilizando a versão 5.5.2 do *gringo* para o *grounding* e a versão 3.3.8 do *clasp* para o *solving*. Esta avaliação foi realizada numa máquina com o Windows 10 Home (64bits), com processador AMD A9-9420 RADEON R5, 5 COMPUTE CORES 2C+3G (3.0GHz) e com 8GB RAM. Para cada *encoding* é apresentado o número de regras, o tempo médio μ(s) e o desvio padrão σ(s), as escolhas efetuadas pelo Clingo e os conflitos. A relação existente entre as escolhas e os conflitos é importante, visto que caso tenham valores semelhantes indica que existe um número elevado de combinatória. Caso o valor das escolhas seja muito maior quando comparado ao valor dos conflitos, indica que existe um número elevado de *backjumping* [14]. Na tabela 6.1, é ainda apresentado o número de dias para o planeamento ótimo e a estimativa do limite inferior desse número de dias. A média e o desvio padrão foram calculados com base nos dados de 10 tempos de execução. Foi definido um tempo máximo para a execução de 60 segundos, visto que existiam programas que não terminavam em tempo útil.

Com a modificação da Listagem 6.2, o número de regras reduziu praticamente para metade nas cinco instâncias. Notou-se principalmente o efeito da melhoria na **instance15.lp**, visto que o número de escolhas reduziu de 2375 para 285 e o número de conflitos de 2062 para 79. No entanto, esta melhoria não foi suficiente para as instâncias maiores, visto que o programa não terminou.

Deste modo, tentou-se melhorar o código do programa **encoding2.lp**. A ideia foi tentar quebrar as simetrias quando existe possibilidade de escolher entre dois fornos, tal

como foi visto na secção 5.4. Tendo isto em consideração, adicionaram-se as 2 restrições presentes na listagem 5.4 nas linhas 7 e 8. Por fim, adicionou-se uma outra restrição que estabelece um limite inferior para o número de dias necessários para se encontrar um planeamento que se encontra na listagem 5.4 na linha 21.

Com estas adições, o programa **recipes_V1.lp**, torna-se o programa final ASP da primeira versão do planeamento. Na tabela, aparece ainda no fim de cada instância, os resultados do programa final da segunda versão do planeamento (**recipes_V2.lp**). O código completo destes programas encontra-se nos Apêndices C e D.

Como se pode verificar, após estas alterações no código ASP, todas as instâncias conseguiram correr as duas versões do planeamento. Nas três maiores instâncias, podemos ainda verificar que existia um número elevado de combinatória, visto que o número de escolhas e conflitos é semelhante nos programas **encoding1.lp** e **encoding2.lp**, o que já não acontece nas versões finais de cada planeamento.

instance5.lp	**Regras**	**Tempo (s)**		**Escolhas**	**Conflitos**	**Nº dias ótimo**	**Limite inferior**
		μ(s)	σ(s)				
encoding1.lp	1568	0.022	0.008	63	28	3	3
encoding2.lp	785	0.020	0.007	67	36	3	3
recipes_V1.lp	943	0.023	0.010	45	16	3	3
recipes_V2.lp	1120	0.034	0.012	43	17	2	2
instance15.lp							
encoding1.lp	12563	0.216	0.025	2375	2062	5	5
encoding2.lp	5620	0.059	0.022	285	79	5	5
recipes_V1.lp	6740	0.064	0.008	208	93	5	5
recipes_V2.lp	8417	0.082	0.013	201	72	5	4
instance21.lp							
encoding1.lp	36738	*timeout*	0.023	766729$^{(*)}$	727009$^{(*)}$	8	8
encoding2.lp	17095	*timeout*	0.020	874427$^{(*)}$	840063$^{(*)}$	8	8
recipes_V1.lp	21204	0.151	0.018	4872	874	8	8
recipes_V2.lp	29186	1.207	0.163	19554	17474	6	5
instance26.lp							
encoding1.lp	73078	*timeout*	0.071	682849$^{(*)}$	637878$^{(*)}$	10	10
encoding2.lp	34828	*timeout*	0.023	771456$^{(*)}$	725895$^{(*)}$	10	10
recipes_V1.lp	43642	0.288	0.045	6640	889	10	10
recipes_V2.lp	59950	0.319	0.054	4384	600	7	7
instance29.lp							
encoding1.lp	102047	*timeout*	0.043	630802$^{(*)}$	259824$^{(*)}$	11	11
encoding2.lp	46088	*timeout*	0.039	746578$^{(*)}$	307262$^{(*)}$	11	11
recipes_V1.lp	57600	0.375	0.046	7560	921	11	11
recipes_V2.lp	79833	0.605	0.064	18805	3595	8	8

$^{(*)}$ Como o programa não terminou em tempo útil, o valor obtido variou por causa do tempo máximo imposto para a execução. Assim, o valor apresentado é o resultado da média para os 10 testes de execução.

Tabela 6.1: Resultados obtidos para os diferentes *encodings*.

Podemos verificar que a melhoria realizada no programa **recipes_V1.lp**, nas instâncias **instance5.lp** e **instance15.lp** fez com que a tempo médio de execução aumentasse ligeiramente, bem como o número de regras criadas. No entanto, nas últimas três instâncias este aumento do número de regras compensa, uma vez que o programa consegue encontrar a solução ótima e terminar. Os programas com menos otimizações conseguem encontrar a solução ótima, apesar de não terminarem.

Quando analisamos os resultados obtidos para a segunda versão do planeamento (**recipes_V2.lp**), verificamos que o número de regras criadas é maior que o do programa da primeira versão e que os tempos de execução também são ligeiramente maiores.

No caso das instâncias **instance15.lp** e **instance21.lp**, a estimativa do limite de dias não coincidiu com o valor ótimo de dias. Como consequência, o tempo médio de execução aumentou nessas instâncias. Caso o valor da estimativa seja o valor ótimo, então a procura pode parar por aí, logo não demora tanto tempo.

Após a inserção das melhorias nos 2 programas, podemos verificar que para a instância maior os programas correm em média em menos de 1 segundo, o que é excelente tendo em conta que as versões com menos otimizações não conseguem sequer terminar em tempo útil. Além disso, os efeitos das melhorias também se fazem sentir no valor das escolhas e dos conflitos que existem (principalmente nas instâncias maiores), uma vez que são valores muito inferiores aos das versões com menos otimizações.

Relativamente aos programas que atingiram o tempo máximo para a execução de 60 segundos, testou-se ainda alargar o tempo limite para 10 minutos. Cada programa foi executado 10 vezes, tal como os anteriores.

O primeiro caso onde os programas **encoding1.lp** e **encoding2.lp** atingiam o limite máximo de tempo foi na **instance21.lp**. Após se testar os dois programas concluiu-se que mesmo com o aumento do tempo máximo para 10 minutos, não foi possível obter uma resposta, visto que novamente os dois programas não terminaram em tempo útil. Posto isto, para as restantes instâncias onde foi atingido o tempo limite de 60 segundos, o resultado obtido aumentando o tempo para 10 minutos é o mesmo, tendo em conta que essas instâncias acrescentam novas receitas à instância anterior.

Por último, a razão pela qual o programa **recipes_V2.lp** encontra um planeamento com um número ótimo de dias menor que o programa **recipes_V1.lp**, deve-se ao facto de nessa versão vários passos poderem estar num dia no mesmo forno, caso tenham a mesma temperatura e a mesma duração.

6.2 Resultados do questionário de usabilidade

O questionário realizado teve como objetivo avaliar a interface gráfica da aplicação e a sua usabilidade. Com esse propósito, foram realizados testes de usabilidade a membros da VICARTE. No final de cada teste, cada pessoa respondeu ao questionário de forma anónima.

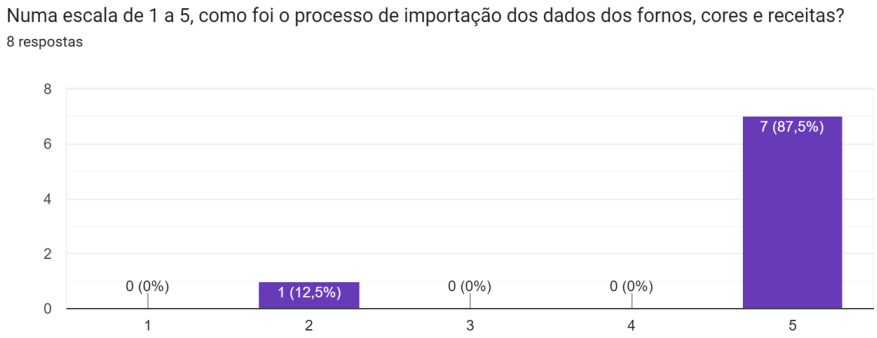


Figura 6.1: Resultados da facilidade na importação dos dados.

O questionário encontrava-se dividido em duas partes: avaliação de componentes da aplicação e a satisfação geral com a mesma. Uma vez que o grupo de utilizadores que testaram a aplicação era composto por 9 pessoas, não foram realizadas perguntas relativamente ao perfil do utilizador, nomeadamente, aos conhecimentos e à experiência que cada cada um possui dentro da VICARTE. De outra forma, os questionários deixavam de ser anónimos pelo facto do grupo ser pequeno. No entanto, todos os utilizadores tinham os conhecimentos necessários para compreender as dificuldades que existiam na realização de planeamentos de receitas de forma manual, portanto as suas opiniões eram relevantes, ao contrário do que aconteceria se os testes fossem realizados com mais pessoas, mas que não pertencessem à VICARTE.

É ainda importante referir que antes dos testes de usabilidade que foram realizados, que a aplicação foi validada com dois membros da VICARTE, que testaram de forma a garantir que os resultados apresentados estavam de facto corretos.

Apesar de se terem realizado 9 testes, apenas foi possível obter 8 respostas ao questionário.

Das respostas obtidas, no geral, os utilizadores consideraram que a importação dos dados dos ficheiros das receitas, fornos e cores foi um processo muito fácil, tendo sido atribuído o valor 5, numa escala de 1 a 5. Apenas um utilizador considerou que foi difícil, atribuindo assim o valor 2. Estes resultados encontram-se na Figura 6.1.

Durante os testes, uma questão interessante foi o facto dos utilizadores terem a intuição de importar primeiro os dados das receitas, em vez dos dados dos fornos e cores. No entanto, os dados das cores (gerais) têm de ser importados em primeiro lugar, visto que as receitas têm nos seus dados a cor específica da mesma, que está relacionada com a cor geral.

Relativamente à forma de pesquisa das receitas, a maioria dos utilizadores está satisfeita, sendo que apenas um utilizador atribuiu o valor 3 e um outro atribuiu o valor 4. Os restantes utilizadores atribuíram o valor 5. Os resultados estão apresentados no gráfico 6.2.

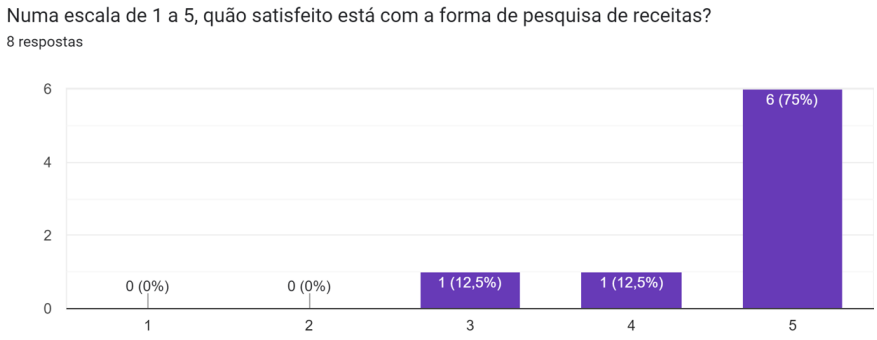


Figura 6.2: Resultados da forma pesquisa de receitas.

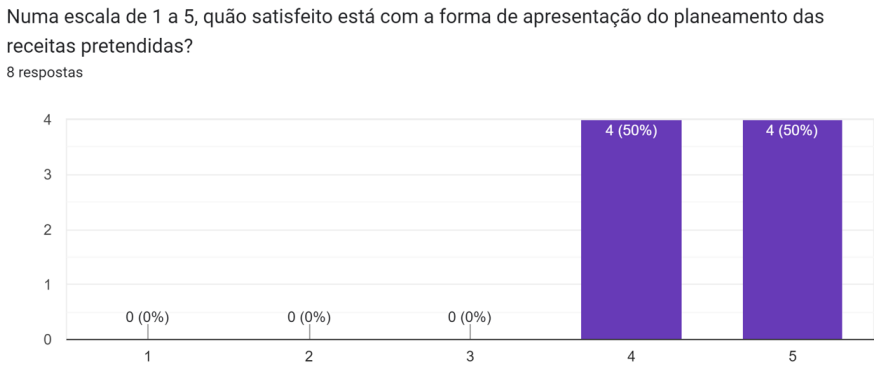


Figura 6.3: Resultados da apresentação final do planeamento de receitas.

Em relação à apresentação do planeamento encontrado, novamente, a maioria dos utilizadores estão satisfeitos, sendo que 50% atribuiu o valor 4 e os restantes 50% atribuíram o valor 5. Um dos utilizadores justificou na secção dos comentários e sugestões, ter dado o valor 4 nesta pergunta pelo facto de não ser apresentado o número total de receitas que se consegue fazer no planeamento apresentado, bem como a indicação de quais são as subreceitas de cada receita principal.

Ainda em relação ao planeamento de receitas, foi perguntado se a coluna que existe com a informação da quantidade que é produzida pela receita seria útil. Nesta questão, as opiniões dividiram-se entre os valores 2 (pouco útil) a 5 (muito útil). Apesar disso, a maioria das pessoas considera que essa informação é de facto útil.

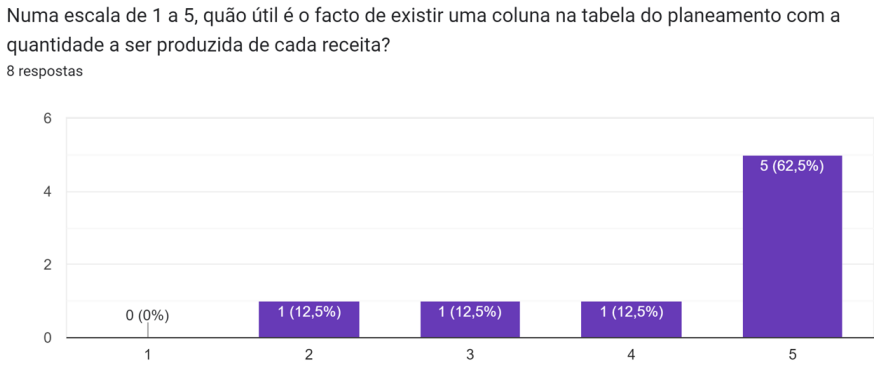


Figura 6.4: Resultados da utilidade da quantidade produzida das receitas.

Tentou-se perceber ainda se as descrições dos erros que poderão ocorrer durante o uso da aplicação estavam claras. Os resultados estão apresentados no gráfico 6.5 . Nesta questão 62,5% dos utilizadores atribuíram o valor 5 e 37,5% atribuíram o valor 4, o que pode indicar que algumas das descrições poderiam estar mais claras.

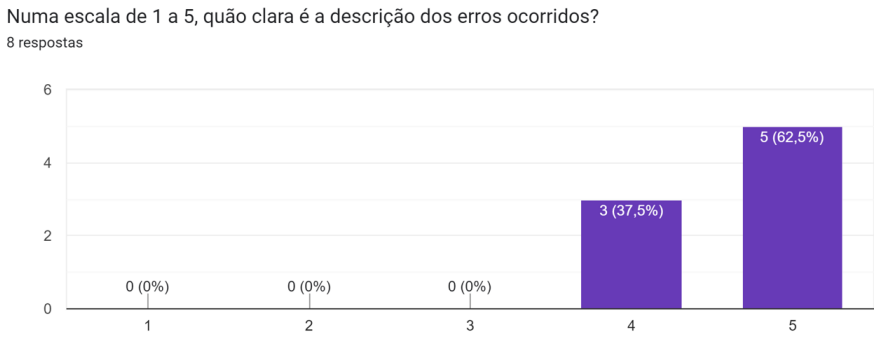


Figura 6.5: Resultados da clareza da descrição dos erros ocorridos.

Para finalizar a primeira secção de perguntas, foi pedido para avaliar tanto a organização da aplicação, como a facilidade de utilização da mesma. Os resultados relativos à organização encontram-se no gráfico 6.6 e os resultados relativos à facilidade de utilização são apresentados no gráfico 6.7.

Metade dos utilizadores atribuíram o valor 5 à questão da organização. Dos restantes, um avaliou a aplicação com o valor 3 e os outros atribuíram o valor 4. Quanto à facilidade de utilização da aplicação, a maioria considerou que a aplicação é fácil de usar, no entanto 2 utilizadores atribuíram o valor 3.

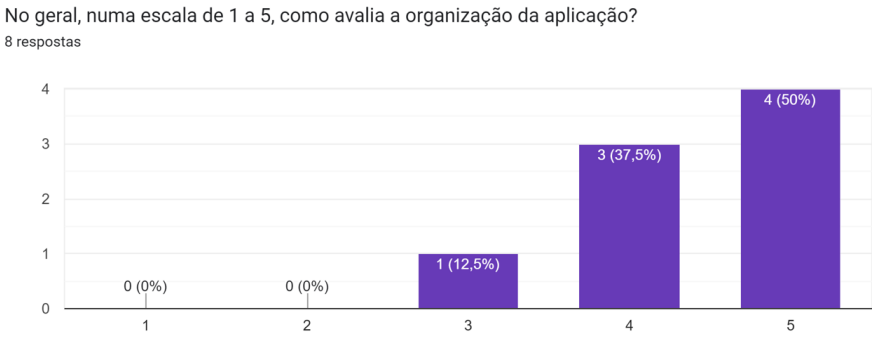


Figura 6.6: Resultados da organização da aplicação.

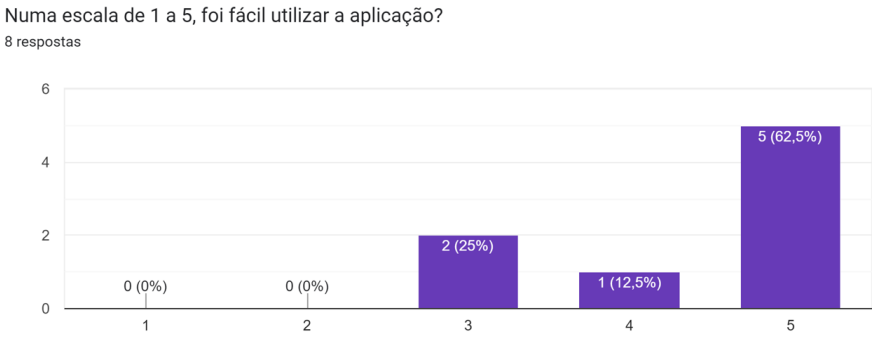


Figura 6.7: Resultados da facilidade de utilização da aplicação.

Na segunda parte do questionário, pretendia-se avaliar a satisfação geral dos utilizadores.

As respostas obtidas nas perguntas seguintes já eram expectáveis, tendo em conta o *feedback* positivo que os utilizadores foram dando durante a realização dos testes.

Os utilizadores de um modo geral, consideraram que a aplicação era útil e que sobretudo era simples e intuitiva, além de ser agradável de usar. Um dos utilizadores quis acrescentar o facto de ser uma aplicação prática.

Para concluir, os utilizadores ficaram bastante satisfeitos com a aplicação, tal como se pode observar pelos resultados que constam no gráfico 6.9. Das 8 pessoas que responderam ao questionário, uma avaliou com o valor 4 e as restantes pessoas com o valor 5.

O questionário terminou com um campo para os utilizadores deixarem os seus comentários e sugestões, se assim o entendessem. Seguem-se alguns deles:

- "Pequenas notas informativas ou uma breve introdução à mesma podem vir a ser muito úteis a novos utilizadores. De resto, é uma aplicação útil que funciona muito bem para o fim pretendido.";

- "Clarificar alguns aspectos estéticos como o nome ou aspecto dos botões, mas em termos de usabilidade tudo ótimo.";
- "Ligação direta dos resultados ao ficheiro de tabela de receitas".

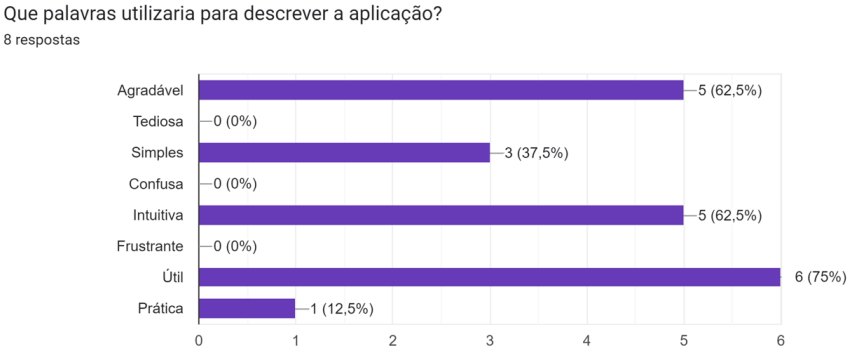


Figura 6.8: Palavras utilizadas para descrever a aplicação.

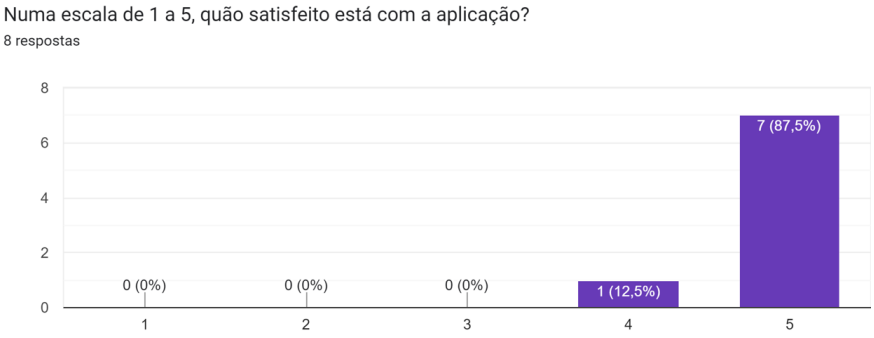


Figura 6.9: Resultados da satisfação geral.

7

Conclusões e Trabalho Futuro

Neste capítulo, serão apresentadas as conclusões desta dissertação e serão mencionados possíveis trabalhos futuros.

7.1 Conclusões

Nesta dissertação foi proposto implementar uma aplicação que facilitasse a criação de horários para a realização de experiências na VICARTE, otimizando os recursos disponíveis.

Para atingir esse objetivo, foram estudadas ferramentas de edição para programas ASP e sistemas de inferência ASP, de forma a selecionar as ferramentas adequadas.

Os dados existentes das várias receitas precisaram de ser estruturados de forma a que essas informações ficassem guardadas numa base de dados.

Ao longo da implementação da aplicação foram surgindo alguns desafios. Um deles foi o desenvolvimento da funcionalidade de pesquisa de receitas pela quantidade de elemento que existia atualmente produzido. Para esta pesquisa o objetivo era encontrar as receitas que utilizassem esse elemento e reajustar as quantidades de elementos e compostos que fossem utilizados nessas receitas. Esta pesquisa tornou-se desafiante pelo facto do utilizador poder indicar mais do que um elemento com quantidade já produzida, tendo por isso de se verificar se existiam receitas com esses elementos em comum. Caso existissem, teria de se verificar qual desses elementos teria uma maior diferença entre a quantidade que existe atualmente produzida e à que seria necessário utilizar para realizar as receitas, de acordo com os dados originais que constam no ficheiro de dados das receitas. Se não existissem receitas em comum entre os elementos selecionados, para cada elemento reajustavam-se as quantidades em cada receita a que o mesmo estivesse associado.

Existiu outro desafio na fase de otimização do *encoding* dos programas ASP. Neste caso, a versão inicial de cada programa teve de ser alterada para algo logicamente semelhante, mas que fosse mais eficiente, uma vez que os programas não conseguiam terminar em tempo útil. Desta forma, foram realizadas melhorias incrementais ao código de maneira a analisar o impacto produzido no desempenho.

Para realizar a avaliação da aplicação desenvolvida, foram realizados testes para analisar o desempenho da aplicação e da usabilidade da mesma. Depois de realizadas algumas melhorias no código dos programas ASP, foi possível correr todas as instâncias testadas em menos de 1 segundo.

Nos testes de usabilidade realizados aos utilizadores, concluiu-se que os mesmos ficaram muito satisfeitos com a aplicação. Através destes testes, foram ainda identificados alguns pontos a melhorar na aplicação que serão referidos na secção seguinte.

Para concluir, tendo em consideração os resultados obtidos através dos testes de desempenho e dos testes de usabilidade, atingiram-se os objetivos inicialmente propostos para este trabalho.

7.2 Trabalho Futuro

Existem ainda melhorias que podem ser realizadas em trabalho futuro relativamente à aplicação.

A aplicação atualmente corre num servidor local. Futuramente, poder-se-ia criar um servidor de base de dados onde as informações que são comuns, como as cores e os fornos, pudessem ficar guardadas, bem como de receitas que tenham já os seus métodos de preparação definidos.

Os planeamentos que são apresentados atualmente consideram à partida que todos os fornos estão disponíveis nos dias seguintes. Isto significa que se existirem duas pessoas que queiram planear receitas na mesma semana, que os fornos ocupados por uma das pessoas não estão a ser considerados no planeamento das receitas da outra. Desta forma, as duas pessoas podem ter no seu planeamento receitas no mesmo dia e no mesmo forno, o que pode não corresponder à realidade por causa das restrições de espaço no forno, temperatura, cores das receitas ou ações realizadas. Assim sendo, num trabalho futuro esta seria uma funcionalidade que poderia ser adicionada à aplicação, considerando os fornos que já estão ocupados. Introduzir esta questão pode ser um desafio, no entanto é de salientar que graças à utilização de ASP, o código dos programas desenvolvidos não necessita de ser reescrito.

Neste trabalho foi criado um ficheiro executável da aplicação para o sistema Windows. Uma vez que é possível criar *packages* para outros sistemas, seria interessante colocar a aplicação a correr no sistema Linux.

Alguns utilizadores quando testaram a aplicação deram sugestões de melhoria. Uma delas foi integrar um manual de ajuda na aplicação ou criar um tutorial. Assim, seria mais fácil a aprendizagem do funcionamento da aplicação para novos utilizadores.

Outra sugestão para um trabalho futuro, é a criação de uma ligação na apresentação do planeamento de receitas entre cada receita do plano e a descrição da receita no ficheiro.

Bibliografia

[1] W. T. Adrian, M. Alviano, F. Calimeri, B. Cuteri, C. Dodaro, W. Faber, D. Fuscà, N. Leone, M. Manna, S. Perri, F. Ricca, P. Veltri e J. Zangari. «The ASP System DLV: Advancements and Applications». Em: *Künstliche Intelligenz* 32.2-3 (2018), pp. 177–179 (ver p. 13).

[2] M. Alviano, R. Bertolucci, M. Cardellini, C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, M. Mochi, V. Morozan, I. Porro e M. Schouten. «Answer Set Programming in Healthcare: Extended Overview». Em: *IPS-RCRA@AI*IA*. Vol. 2745. CEUR Workshop Proceedings. CEUR-WS.org, 2020 (ver pp. 19, 21).

[3] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri e J. Zangari. *DLV: Evolution and Perspectives*. 2018 (ver p. 13).

[4] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri e J. Zangari. «The ASP System DLV2». Em: *LPNMR*. Vol. 10377. Lecture Notes in Computer Science. Springer, 2017, pp. 215–221 (ver pp. 13, 17).

[5] *Benefits of Python*. (Acedido em 06/12/2022). URL: https://www.discoverdatascience.org/articles/what-is-python-used-for-why-is-it-important-to-learn/ (ver p. 28).

[6] *Documentação do CLORM, An ORM Interface for Clingo*. (Acedido em 15/02/2022). URL: https://clorm.readthedocs.io/en/latest/clorm/background.html#an-orm-interface-for-clingo (ver p. 15).

[7] *Documentação do CLORM, Quick Start*. (Acedido em 15/02/2022). URL: https://clorm.readthedocs.io/en/latest/clorm/quickstart.html (ver p. 15).

[8] *Documentação do Kivy*. (Acedido em 15/02/2022). URL: https://kivy.org/doc/stable/ (ver p. 17).

[9] *Documentação do KivyMD*. (Acedido em 15/05/2022). URL: https://kivymd.readthedocs.io/en/latest/ (ver p. 17).

[10] *Documentação do Tkinter*. (Acedido em 15/02/2022). URL: https://docs.python.org/3/library/tkinter.html (ver p. 16).

[11] *Documentação EMBASP*. (Acedido em 27/01/2022). URL: https://www.mat.unical.it/calimeri/projects/embasp/#Documentation (ver p. 15).

[12] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe e E. C. Teppan. «Industrial Applications of Answer Set Programming». Em: *Kunstliche Intelligenz* 32 (2-3 2018-08), pp. 165–176. ISSN: 16101987. DOI: 10.1007/s13218-018-0548-6 (ver pp. 3, 7, 22).

[13] O. Febbraro, K. Reale e F. Ricca. «Testing ASP programs in ASPIDE». Em: *CILC*. Vol. 810. CEUR Workshop Proceedings. CEUR-WS.org, 2011, pp. 115–129 (ver p. 12).

[14] M. Gebser, R. Kaminski, B. Kaufmann e T. Schaub. *Answer set solving in practice*. Morgan & Claypool Publishers, 2012, p. 212. ISBN: 9781608459711. DOI: 10.2200/S00457ED1V01Y201211AIM019 (ver pp. 3, 6, 13, 48).

[15] S. Germano, F. Calimeri e E. Palermiti. «LoIDE: a web-based IDE for Logic Programming - Preliminary Technical Report». Em: (2017-09). URL: http://arxiv.org/abs/1709.05341 (ver p. 10).

[16] C. Kloimüllner, J. Oetsch, J. Pührer e H. Tompits. «Kara: A System for Visualising and Visual Editing of Interpretations for Answer-Set Programs». Em: *INAP/WLP*. Vol. 7773. Lecture Notes in Computer Science. Springer, 2011, pp. 325–344 (ver p. 11).

[17] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (ver p. ii).

[18] C. Machado, A. Machado, T. Palomar e M. Vilarigues. «Grisaille in Historical Written Sources». Em: *Journal of Glass Studies* 61 (2019), pp. 71–86 (ver p. 1).

[19] J. Oetsch, J. Pührer e H. Tompits. «The SeaLion has Landed: An IDE for Answer-Set Programming - Preliminary Report». Em: *INAP/WLP*. Vol. 7773. Lecture Notes in Computer Science. Springer, 2011, pp. 305–324 (ver p. 11).

[20] *Oracle - What is MySQL?* (Acedido em 25/09/2022). URL: https://www.oracle.com/mysql/what-is-mysql/ (ver pp. 32, 33).

[21] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano e N. Leone. «Team-building with answer set programming in the Gioia-Tauro seaport». Em: *Theory Pract. Log. Program.* 12.3 (2012), pp. 361–381 (ver p. 22).

[22] C. R. Santos, M. Vilarigues, P. Dabas, I. Coutinho e T. Palomar. «Reproducing crystal glass from three 18th-20th centuries Portuguese glass arcana». Em: *International Journal of Applied Glass Science* 11 (4 2020-10), pp. 743–755. ISSN: 20411294. DOI: 10.1111/ijag.15611 (ver p. 1).

[23] M. Vilarigues, C. Machado, A. Machado, M. Costa, L. Alves, I. Cardoso e A. Ruivo. «Grisailles: Reconstruction and characterization of historical recipes». Em: *International Journal of Applied Glass Science* 11.4 (2020-10), pp. 756–773. ISSN: 2041-1286. DOI: 10.1111/ijag.15793 (ver p. 2).

A

Esquema Relacional

Esta versão já está simplificada, portanto não se encontram presentes relações redundantes. As chaves primárias das tabelas encontram-se sublinhadas e as chaves estrangeiras encontram-se a negrito.

material(materialName)
colour(colourName)
subcolour(subcolourName, **colourName**)

- **colourName** é chave estrangeira da tabela colour.

furnace(idFurnace, maxTemperature)
meltingPot(potCapacity)
goesToFurnace(**idFurnace**, **potCapacity**, maxSamples)

- **idFurnace** é chave estrangeira da tabela furnace.
- **potCapacity** é chave estrangeira da tabela meltingPot.

recipe(idRecipe, writtenSource, recipeNumberChapter, produced, **materialName**, **subcolourName**, quantityProduced)

- **materialName** é chave estrangeira da tabela material.
- **subcolourName** é chave estrangeira da tabela subcolour.

step(**idRecipe**, stepNumberTimeCalculation, maxDuration, recipeMaxTemperature, operator)

- **idRecipe** é chave estrangeira da tabela recipe.

actions(actionName)
compound(compound)
element(elementName)
hasSubrecipe(**idRecipe**, **stepNumberTimeCalculation**, **idSubrecipe**, **elementName**)

- **idSubrecipe** é chave estrangeira da tabela recipe.

- (**idRecipe**, **stepNumberTimeCalculation**) é chave estrangeira da tabela step.
- **elementName** é chave estrangeira da tabela element.

useCompound(**compound**, **idRecipe**, **stepNumberTimeCalculation**, compoundQuantity)

- **compound** é chave estrangeira da tabela compound.
- (**idRecipe**, **stepNumberTimeCalculation**) é chave estrangeira da tabela step.

useElement(**elementName**, **idRecipe**, **stepNumberTimeCalculation**, elementQuantity)

- **elementName** é chave estrangeira da tabela element.
- (**idRecipe**, **stepNumberTimeCalculation**) é chave estrangeira da tabela step.

executeAction(**actionName**, **idRecipe**, **stepNumberTimeCalculation**)

- **actionName** é chave estrangeira da tabela actions.
- (**idRecipe**, **stepNumberTimeCalculation**) é chave estrangeira da tabela step.

isComposed(**compound**, **elementName**)

- **compound** é chave estrangeira da tabela compound.
- **elementName** é chave estrangeira da tabela element.

B

Modelo Relacional

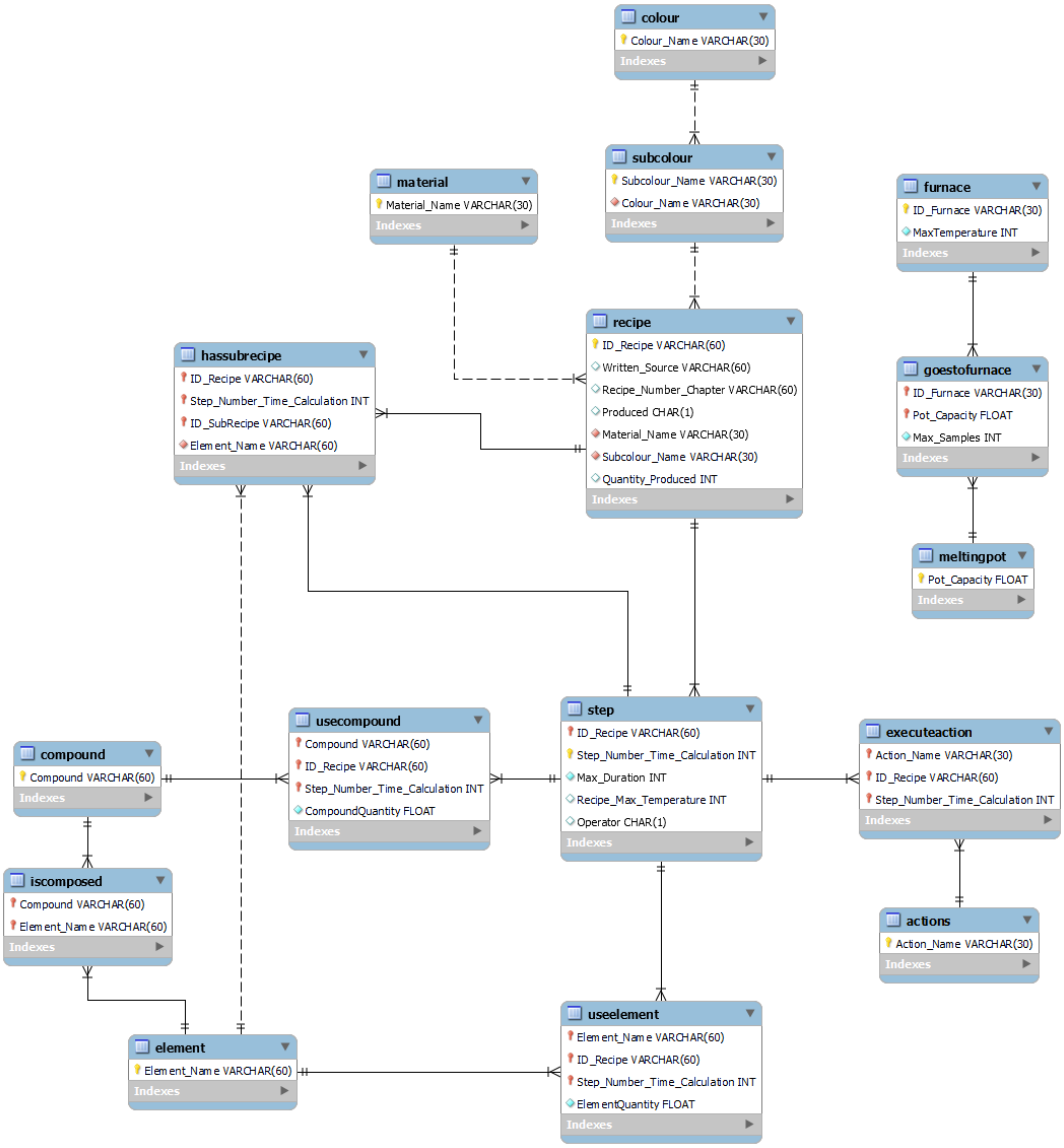


Figura B.1: Modelo relacional

C

Código ASP da primeira versão do planeamento de receitas

Listagem C.1: Programa ASP recipesV1.lp - primeira versão do planeamento

```
%geral
1{assign(IDRecipe,StepNumber,F,D,T):furnace(F,FurnaceMaxTemp,_),
    day(D), RecipeMaxTemp<=FurnaceMaxTemp, F!="Forno D (redondo)
    ", F!="Forno C (4)"}1 :- step(IDRecipe,StepNumber,_,
    RecipeMaxTemp,Action,_,T),Action!="Calcination", Action!="
    Annealing".
%forno D
1{assign(IDRecipe,StepNumber,"Forno D (redondo)",D,T) : furnace
    ("Forno D (redondo)",FurnaceMaxTemp,_), day(D), RecipeMaxTemp
    <=FurnaceMaxTemp}1 :- step(IDRecipe,StepNumber,_,
    RecipeMaxTemp,"Calcination",_,T).
%forno C
1{assign(IDRecipe,StepNumber,"Forno C (4)",D,T) : furnace("Forno
     C (4)",FurnaceMaxTemp,_),day(D),RecipeMaxTemp<=
    FurnaceMaxTemp}1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,"
    Annealing",_,T).

:-not assign(_,_,"Forno B (6)",D1,_),assign(IDRecipe,StepNumber
    ,"Forno A (Blue)",D1,T), step(IDRecipe,StepNumber,_,
    RecipeMaxTemp,_,_,T),RecipeMaxTemp<=1200, not assign(_,_,"
    Forno B (6)",D2,_),assign(_,_,"Forno A (Blue)",D2,_),D1<D2.

```

```
10 :-not assign(_,_,"Forno B (6)",D1,_),assign(_,_,"Forno A (Blue)
       ",D1,_), not assign(_,_,"Forno B (6)",D2,_),assign(IDRecipe2,
       StepNumber2,"Forno A (Blue)",D2,T2), step(IDRecipe2,
       StepNumber2,_,RecipeMaxTemp2,_,_,T2),RecipeMaxTemp2<=1200,D1<
       D2.
11
12 :- not assign(_,_,_,1,_).
13
14 :- assign(IDRecipe1,_,F,D,_), assign(IDRecipe2,_,F,D,_),
       IDRecipe1!=IDRecipe2.
15 :- assign(IDRecipe,_,F,D,T1), assign(IDRecipe,_,F,D,T2),T1<T2.
16
17 :- assign(IDRecipe,_,_,D1,T), assign(IDRecipe,_,_,D2,T), D1<D2.
18
19 :- hasSubrecipe(IDRecipe,StepNumber1,IDSubrecipe),assign(
       IDRecipe,StepNumber1,_,D1,_), assign(IDSubrecipe,StepNumber2,
       _,D2,_),D2>=D1.
20
21 :-S=#sum{P,IDRecipe,BN:potsUsed(IDRecipe,BN,P,_),assign(IDRecipe
       ,BN,F,D,_)},furnace(F,_,C),day(D),S>C.
22
23 numberDays(N):-N=#count{D:assign(_,_,_,D,_)}.
24 #minimize {N@2: numberDays(N)}.
25
26 :-numberDays(N),N<L,limit(L).
27
28 :- assign(_,_,_,D1,_), assign(_,_,_,D2,_),numberDays(N),D1<D2,
       D2-D1>=N.
```

D

Código ASP da segunda versão do planeamento de receitas

Listagem D.1: Programa ASP recipesV2.lp - segunda versão do planeamento

```
1 %geral
2 1{assign(IDRecipe,StepNumber,F,D,T):furnace(F,FurnaceMaxTemp,_),
    day(D), RecipeMaxTemp<=FurnaceMaxTemp, F!="Forno D (redondo)
    ", F!="Forno C (4)"}1 :- step(IDRecipe,StepNumber,_,
    RecipeMaxTemp,Action,_,T),Action!="Calcination", Action!="
    Annealing".
3 %forno D
4 1{assign(IDRecipe,StepNumber,"Forno D (redondo)",D,T) : furnace
    ("Forno D (redondo)",FurnaceMaxTemp,_), day(D), RecipeMaxTemp
    <=FurnaceMaxTemp}1 :- step(IDRecipe,StepNumber,_,
    RecipeMaxTemp,"Calcination",_,T).
5 %forno C
6 1{assign(IDRecipe,StepNumber,"Forno C (4)",D,T) : furnace("Forno
     C (4)",FurnaceMaxTemp,_),day(D),RecipeMaxTemp<=
    FurnaceMaxTemp}1:-step(IDRecipe,StepNumber,_,RecipeMaxTemp,"
    Annealing",_,T).
7
8 :-not assign(_,_,"Forno B (6)",D1,_),assign(IDRecipe,StepNumber
    ,"Forno A (Blue)",D1,T),step(IDRecipe,StepNumber,_,
    RecipeMaxTemp,_,_,T),RecipeMaxTemp<=1200,not assign(_,_,"
    Forno B (6)",D2,_),assign(_,_,"Forno A (Blue)",D2,_),D1<D2.
9
```

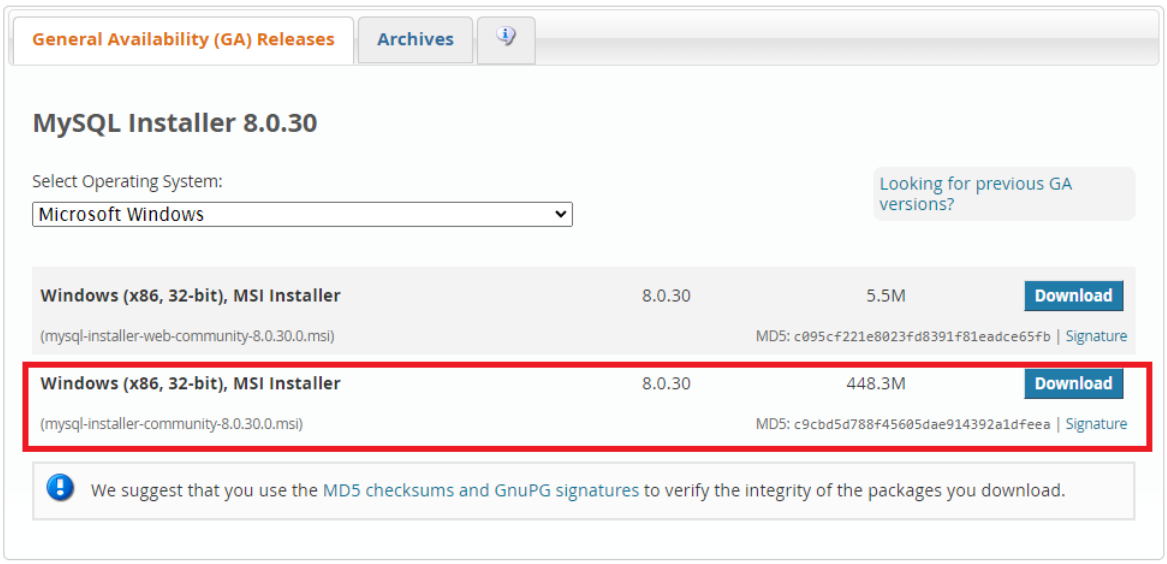
```
10 :-not assign(_,_,"Forno B (6)",D1,_),assign(_,_,"Forno A (Blue)
       ",D1,_),not assign(_,_,"Forno B (6)",D2,_),assign(IDRecipe2,
       StepNumber2,"Forno A (Blue)",D2,T2),step(IDRecipe2,
       StepNumber2,_,RecipeMaxTemp2,_,_,T2),RecipeMaxTemp2<=1200,D1<
       D2.
11
12 :- not assign(_,_,_,1,_).
13
14 %mesmo dia mesmo forno, com a duracao diferente
15 :- assign(IDRecipe1,BN1,F,D,_), assign(IDRecipe2,BN2,F,D,_),step
       (IDRecipe1,BN1,MaxDur1,_,_,_,_),step(IDRecipe2,BN2,MaxDur2,_,
       _,_,_),IDRecipe1!=IDRecipe2,MaxDur1<MaxDur2.
16
17 %mesmo dia mesmo forno, com temperaturas diferentes
18 :- assign(IDRecipe1,BN1,F,D,_), assign(IDRecipe2,BN2,F,D,_),step
       (IDRecipe1,BN1,_,MaxTemp1,_,_,_),step(IDRecipe2,BN2,_,
       MaxTemp2,_,_,_),IDRecipe1!=IDRecipe2,MaxTemp1<MaxTemp2.
19
20 %mesmo dia forno diferente, com a duracao igual e temp igual
21 :- assign(IDRecipe1,BN1,F1,D,_), assign(IDRecipe2,BN2,F2,D,_),
       step(IDRecipe1,BN1,MaxDur,MaxTemp,_,_,_),step(IDRecipe2,BN2,
       MaxDur,MaxTemp,_,_,_),IDRecipe1!=IDRecipe2,F1!=F2.
22
23 :- assign(IDRecipe,_,_,D1,T), assign(IDRecipe,_,_,D2,T), D1<D2.
24
25 :- hasSubrecipe(IDRecipe,StepNumber1,IDSubrecipe),assign(
       IDRecipe,StepNumber1,_,D1,_), assign(IDSubrecipe,StepNumber2,
       _,D2,_),D2>=D1.
26
27 :-S=#sum{P,IDRecipe,BN:potsUsed(IDRecipe,BN,P,_),assign(IDRecipe
       ,BN,F,D,_)},furnace(F,_,C),day(D),S>C.
28
29 numberDays(N):-N=#count{D:assign(_,_,_,D,_)}.
30 #minimize {N@2: numberDays(N)}.
31
32 :-numberDays(N),N<L,limit(L).
33
34 :- assign(_,_,_,D1,_), assign(_,_,_,D2,_),numberDays(N),D1<D2,
       D2-D1>=N.
35
```

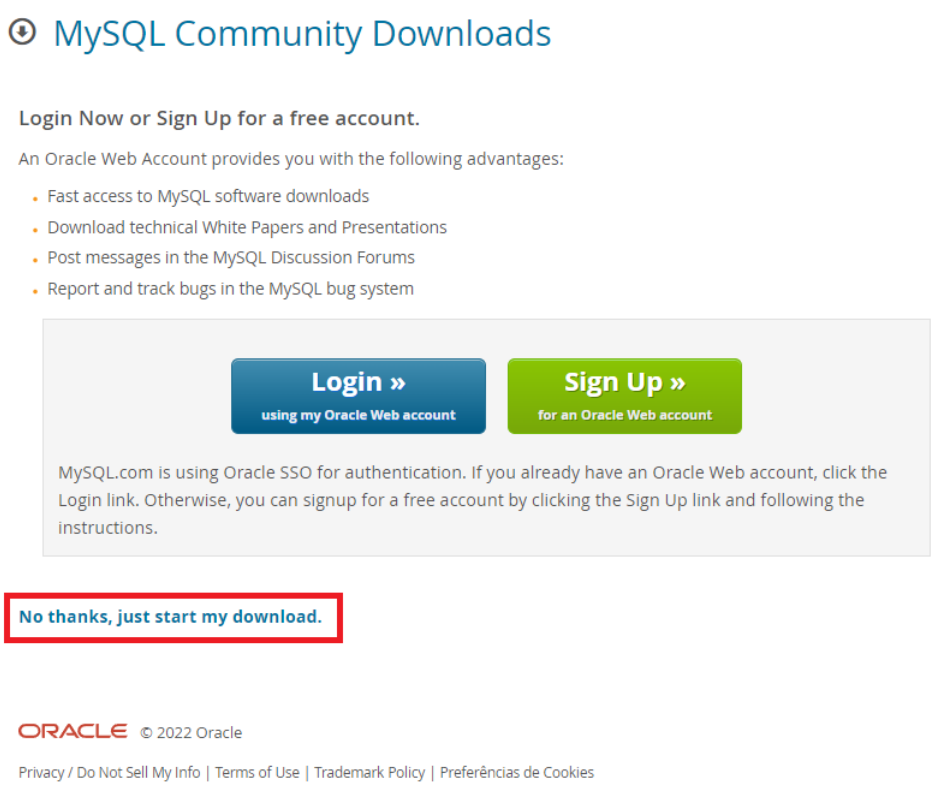
```
36 :-assign(IDRecipe1,_,F,D,_), assign(IDRecipe2,_,F,D,_), recipe(
       IDRecipe1,_,_,Colour1),recipe(IDRecipe2,_,_,Colour2),
       IDRecipe1!=IDRecipe2, Colour1!=Colour2.
37
38 :-assign(IDRecipe1,BN1,F,D,_), assign(IDRecipe2,BN2,F,D,_), step
       (IDRecipe1,BN1,_,_,"Stiring",_,_),recipe(IDRecipe1,
       WrittenSource1,_,_),recipe(IDRecipe2,WrittenSource2,_,_),
       WrittenSource1!=WrittenSource2,IDRecipe1!=IDRecipe2.
39
40 :-        assign(IDRecipe,BN,F,D,T1), assign(IDRecipe,BN,F,D,T2),
       step(IDRecipe,BN,_,_,_,_,T1),step(IDRecipe,BN,_,_,_,_,T2),T1<
       T2.
```

E

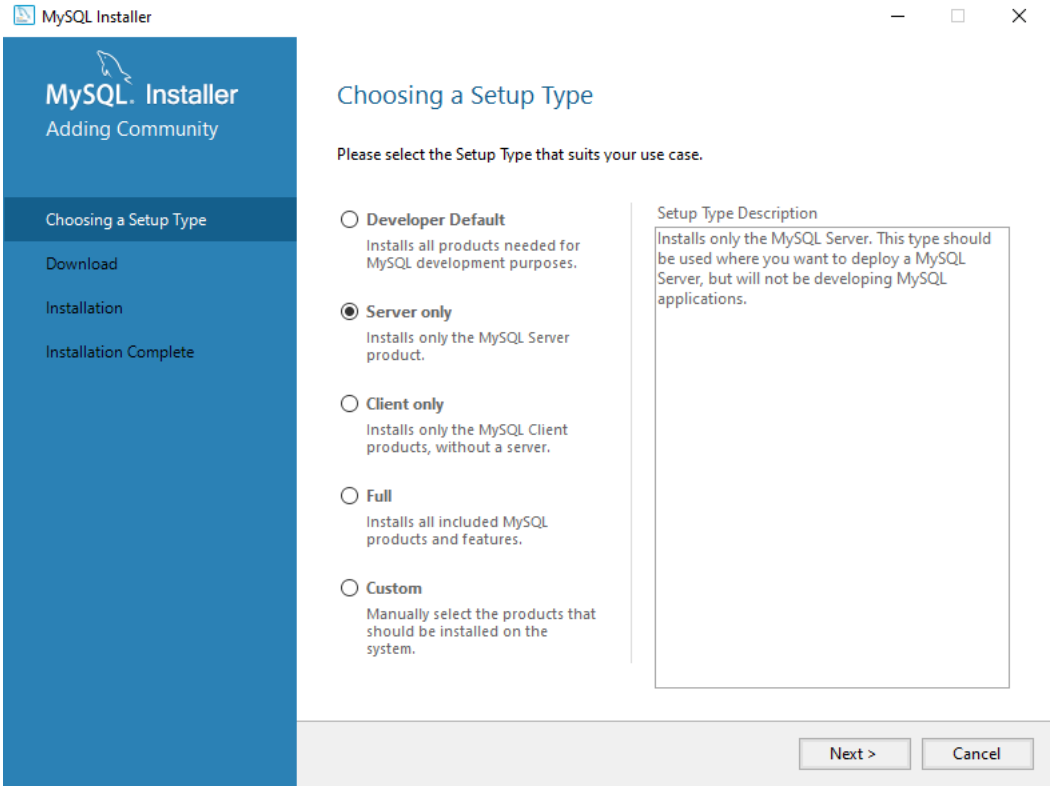
Instruções de instalação

Antes de executar a aplicação é importante que se tenha instalado o MySQL. O download do mesmo pode ser feito em: https://dev.mysql.com/downloads/installer/

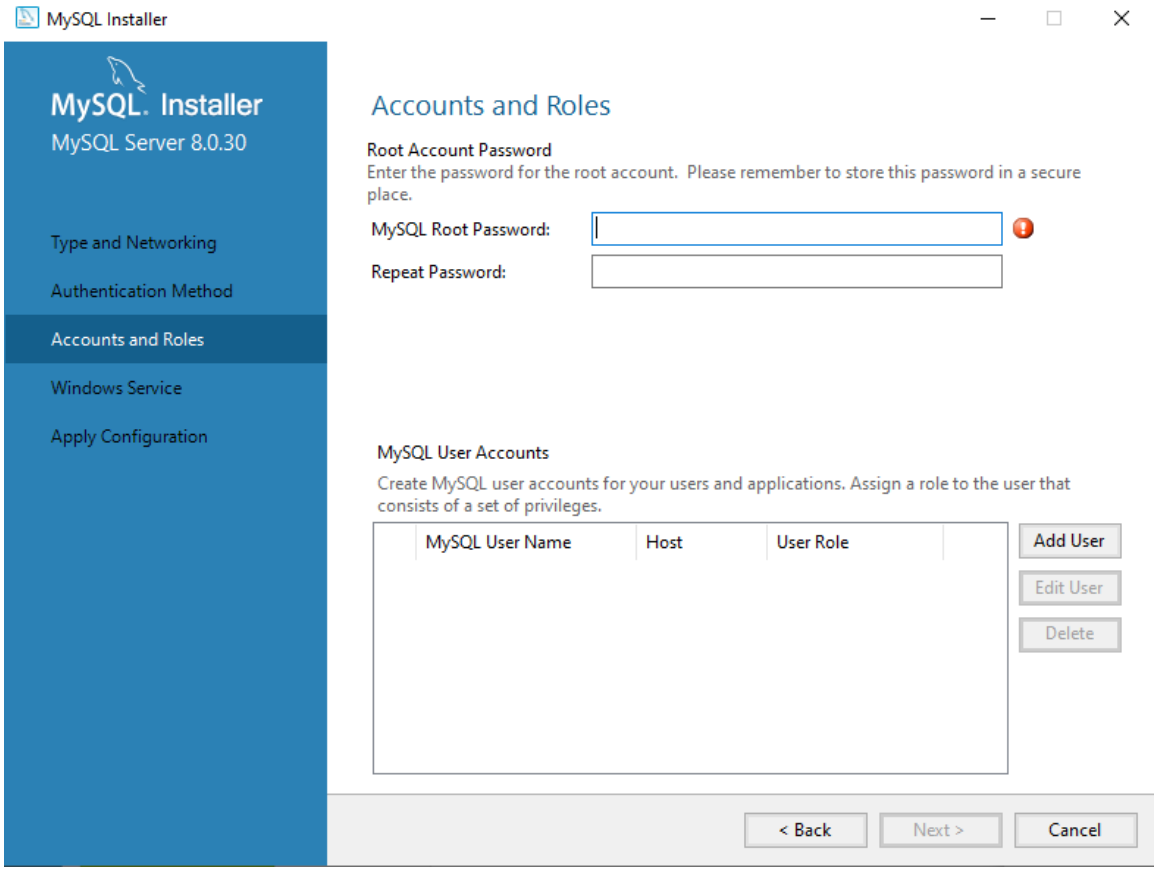




Depois do download estar concluído, escolher a opção **Server only**.



Os restantes passos são triviais, visto que só é necessário carregar no botão "Next". Quando aparecer o ecrã abaixo coloca-se a password desejada. Esta password será depois pedida quando se entrar pela primeira vez na aplicação.

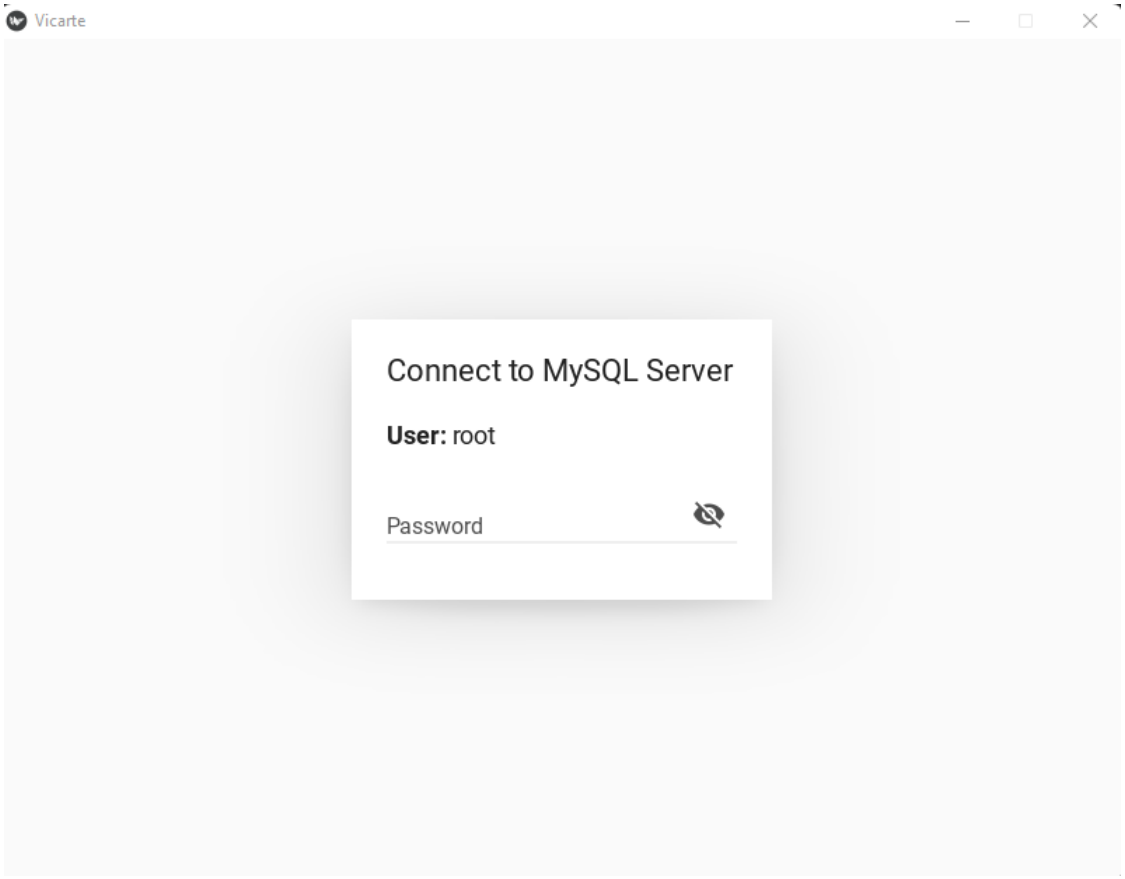


Depois de concluída a instalação do MySQL, basta transferir os ficheiros de dados (formato .csv) das receitas e dos fornos/cores, bem como o ficheiro .zip que contém a aplicação. O nome do ficheiro executável da aplicação é gui.exe .

Na aplicação, primeiro insere-se a password do MySQL.

Depois é apenas preciso importar primeiro os dados das cores e fornos, e de seguida, o das receitas. Desta forma, os dados ficam guardados na base de dados.

Quando os dados estiverem importados, pode começar-se a realizar planeamentos.


Vicarte
Connect to MySQL Server
User: root
Password


Vicarte
recipes
planner
Welcome!
Start Planning
Import Data
Change MySQL password