Universidade do Minho
Departamento de Informática

Mestrado em Engenharia Informática

Validação de Software Altamente Configurável

Pedro Manuel Pereira de Lemos

Orientado por:

Professor Doutor Pedro Rangel Henriques

Supervisionado por:

João Paulo Macedo da Cunha, *Principal Engineer* na Critical Software S.A.

Braga, 19 de Junho de 2012

Abstract

Evolution in application areas led to an increasing complexity and scale of software systems. In this context, recent configurable systems emerged to provide an efficient solution to cope with the frequent changes in systems requirements.

Configurable systems make applications more flexible and adaptable to each specific needs, but configurable systems are also error prone. Easily wrong configurations can be loaded and this may lead to undesirable and erroneous behavior.

Assuming that the main program is fully tested and accepted as correct, those imported configurations need validation to guarantee the absence of errors in the final application. Making application systems more reliable, it is possible to deliver quality products.

Formal method thechniques on testing are the classic approach to software validation and verification. These approaches are specially tuned for classic programs, and fail when applied to configurable systems. This was the motivation for this MSc. Degree project.

This master work gave rise to three distinct outcomes; a theoretical contribution, concerned with the problem study and the proposal of an effective methodological approach for such systems validation; a practical contribution consisting in a tool that implements that methodology; and finally a pragmatic contribution, with the application of that tool to a concrete case study.

Keywords: Highly Configurable Systems, Formal Methods, Unit Testing, Acceptance Tests, Validation Environments, Software Certification, Business Rules Management Systems (`BRMS`).

Resumo

A evolução nas áreas de aplicação (quer em número quer em ambição dos problemas a resolver) conduziu a uma crescente complexidade e dimensão dos sistemas de software. Inerentemente, na procura de uma boa solução que ajude a dar resposta às constantes alterações de comportamento requerida para satisfazer os diversos utilizadores destes sistemas, emergem sistemas mais flexíveis, que se acomodam facilmente a diferentes necessidades - sistemas configuráveis.

Os sistemas configuráveis tornam as aplicações mais adaptáveis às necessidades de cada um, porém tornam possível a introdução de configurações erradas que podem levar a comportamentos incorrectos e indesejáveis.

Assumindo que o programa principal está devidamente testado e aceite como correcto, constata-se que as configurações importadas para a aplicação necessitam de validação de modo a ser garantida a ausência de erros. Conferindo aos sistemas aplicacionais dos dias de hoje uma maior fiabilidade, garante-se a entrega de "produtos" de qualidade.

Tradicionalmente as abordagens à validação e verificação de software caem sobre os métodos formais ou testes sobre o sistema. No entanto, ambas as abordagens dedicam-se essencialmente ao código das aplicações como forma de certificar e garantir a qualidade dos projectos de software.

Esta dissertação apresenta uma proposta para colmatar a falha introduzida por abstracção das aplicações, no que diz respeito à certificação de software altamente configurável, com particular ênfase sobre os objectivos de negócio passíveis de serem configurados[1] *em tempo de execução, de modo a garantir que as configurações empregues estão correctas e de acordo com os requisitos.*

A abordagem seguida neste trabalho deu resultados a três níveis distintos; um contributo teórico em que se estudou e analisou o problema em busca de uma abordagem metodológica eficaz para validação deste tipo de sistemas; um contributo prático, pelo desenvolvimento de um sistema que implementa a metodologia definida; e finalmente um contributo pragmático mostrado pela aplicação do sistema desenvolvido a um caso de estudo.

Palavras Chave: Software Altamente Configurável, Métodos Formais, Testes Unitários, Testes de Aceitação, Ambientes de Validação, Certificação de Software, Sistemas de Gestão de Regras de Negócio (`BRMS`).

[1] Por exemplo, através de inserção de ficheiros `.xml` ou `.csv`.

Agradecimentos

"In the end, we will remember not the words of our enemies, but the silence of our friends."

Martin Luther King Jr.

Muito tempo eu esperei por este momento — a possibilidade de agradecer alguém — provavelmente uma das melhores coisas que podemos fazer na vida! Além disso, esta é provavelmente a única parte que todos os meus amigos irão ler, portanto é melhor não me esquecer de ninguém! Durante o longo processo de desenvolvimento desta tese de mestrado praticamente todos os que me rodeiam de alguma forma acabaram por estar envolvidos, até mesmo sem darem por isso.

Em primeiro lugar eu gostaria de agradecer ao *Professor Pedro Henriques* pela constante presença, ajuda e motivação; por saber como me manter focado nos objectivos mesmo quando parecia impossível fazê-lo. Agradeço também pelo café antes das nossas reuniões matinais e pela boa disposição logo ao começar o dia. Mas mais importante ainda, agradeço pelo tempo e pela disponibilidade, pelas conversas inspiradoras e também pelas palmadinhas nas costas! Um profundo e sincero obrigado pela sua amizade.

Gostaria de agradecer também a alguém que tem tido um papel muito importante na minha vida profissional (e académica), o *João Paulo Cunha*, o mentor desta tese de mestrado, pelas *dicas* que sempre me foi dando no seu muito pouco tempo disponível e que se mostraram certeiras — não me vou esquecer nunca do "*avião que consegue evitar despenhar-se numa montanha*" — acho que ainda hoje penso se consegui entender correctamente o que me queria transmitir. Obrigado pelo excelente desafio!

Quero também agradecer à minha familia, em especial aos meus pais, *Manuel* e *Adelina*, e à minha irmã *Paula*, pela paciência para lidarem comigo todos os dias e por terem de me aturar de forma ininterrupta. E também à menina *Inês* (apesar de ela não poder ler isto), que com pouco mais de 3 anos, e com aquele seu sorriso malandro e aquele seu olhar puro me transmite uma enorme alegria.

Não me posso esquecer obviamente de todos os que comigo caminharam ao longo da licenciatura e do mestrado e que são hoje amigos para a vida. Obrigado *João Granja*, *Nuno Oliveira*, *Pedro Ribeiro*, *Fábio Lima* e *Carlos Pereira*, pelos momentos de diversão mas também por estarem disponíveis sempre que necessário. De entre

estes gostaria de destacar o *João Granja*, pela enorme amizade, pela força e motivação, por me ouvir e pelas sábias palavras de conforto nos momentos mais díficeis; e o *Nuno Oliveira*, pela sua constante boa disposição e por ser um amigo sempre presente.

Quero também agradecer a todos os meus colegas da `Critical Software S.A.`, pelo bom ambiente criado, pelo espirito de equipa e entreajuda e por todos os bons momentos criados durante e após o horário de trabalho; destacando a equipa do SCI: *Manuel Martins*, *Patrick Machado*, *Paulo Abreu*, *Sérgio Barros*, *Nuno Monteiro*, *Eva Martins*, *André Carmo*, *Andreia Melo*, *Luís Leitão*, *Luís Machado*, *Mário Sampaio*, *Pedro Fernandes* e *Sílvia Astorga* — trabalhar convosco é fantástico; ao *Pedro Silva*, pelos bons conselhos que sempre me deu durante todo o estágio curricular e por me iluminar com algumas brilhantes ideias; à *Marilyn*, pelo prontidão e disponibilidade em me explicar os procedimentos de validação e verificação executados nos projectos da `Critical`; ao *Pedro Ribeiro*, pelas dicas e opiniões sempre bem-vindas e também pela disponibilidade em me ter mostrado os modos de funcionamento do `WOW`; ao *Cerqueira* e ao *Paulo Martins*, por numa conversa durante uma viagem de comboio me terem elucidado sobre `BRMS` e me ajudarem a definir o rumo deste trabalho; e a muitos outros com quem todos os dias tenho o prazer de partilhar o mesmo espaço de trabalho.

E como não podia deixar de ser, quero agradecer a todos os meus amigos que me têm acompanhado ao longo dos anos e com quem tenho o prazer de partilhar grande parte dos meus dias: *Pedro Macieira*, *Célia*, *Teresa*, *Patricia Cunha*, *Patricia Silva*, *Diogo*, *Zé Pedro*, *Lili*, *Ritinha*, *Ana Rita*, *Helena*, *Alpoim*, *Paulo*, *Nelson*, *Ricardo Xavier*, *Maura Cunha*, *Jota*, e a todos os amigos do café de 6ªfeira, com destaque para o *Nuno* e a *Filipa*, pela sua amizade e pelos bons momentos de distracção sobre duas rodas, mas também por saber que posso contar com eles; e a todos os outros de quem me possa ter eventualmente esquecido... um sincero obrigado!

E finalmente, um especial obrigado à *Joana Andersen* por me deixar bem disposto e com um sorriso no rosto, por ser minha companhia todos os dias e, também pelo *boost* que provocou na minha vida — obrigado por seres tão genuína!

Conteúdo

Figuras

Tabelas

Códigos

Acrónimos

A

API Application Programming Interface, p. 56.

ASF Apache Software Foundation, p. 125.

B

BRMS Business Rule Management System, p. 32.

C

CIT Combinatorial Interaction Testing, p. 26.

CSV Comma-Separated Values, p. 33.

E

EA Enterprise Architect, p. 19.

ECS Enterprise Critical Solutions, p. 1.

G

GA Google Analytics, p. 20.

GWO Google Website Optimizer, p. 20.

I

IP Internet Protocol, p. 72.

J

JAR Java Archive, p. 63.

JDBC Java Database Connectivity, p. 63.

L

`LHS` Left Hand Side, p. 65.

M

`MEI` Mestrado de Engenharia Informática, p. 1.

P

`POC` Proof of Concept, p. 55.

`POJO` Plain Old Java Object, p. xiii.

Q

`QAS` Quality Assurance Systems, p. 1.

R

`RHS` Right Hand Side, p. 66.

`RV` Regras de Validação, p. x.

S

`SPAE` Software Product Assurance Engineer, p. 22.

`SPL` Software Product Line, p. 31.

V

`VP` Visual Paradigm, p. 18.

`V&V` Verificação e Validação, p. 13.

W

`WOW` Work Orders Web, p. 34.

X

`XML` Extensible Markup Language, p. 33.

Para a Joana, para o Jó, para os meus pais e irmã,
por serem a luz do meu caminho.
Não há palavras que transmitam plenamente a minha gratidão.

Capítulo 1

Introdução

"Não sigas por caminhos feitos. Abre antes o teu caminho e deixa um trilho!"

Muriel Strode

Este documento foi escrito no contexto de um trabalho de mestrado na área tecnológica de *Enterprise Critical Solutions* (ECS)[1] e *Quality Assurance Systems* (QAS)[2].

A dissertação referida no parágrafo anterior enquadra-se no *Mestrado de Engenharia Informática* (MEI) leccionado pelo Departamento de Informática da Universidade do Minho e surge da realização de um projecto de estágio em colaboração com a Critical Software S.A. — Plataforma Eleitoral.

Neste capítulo é dado a conhecer o contexto a partir do qual surge a necessidade deste estudo, bem como os principais objectivos e metas definidas. Além disso descreve-se a estrutura utilizada para o restante documento.

1.1 Contexto

Nos dias de hoje quase tudo o que existe no mundo tem um lado informático, por muito pequeno que possa ser. Quase tudo o que nos rodeia é controlado por sistemas informáticos; desde o despertar até ao final do dia interagimos com uma enorme quantidade de sistemas: internet e serviços web, sistemas de transportes, comunicações, saúde, segurança, entre muitos outros.

Estes sistemas têm vindo a crescer não apenas em escala mas também em complexidade. E, aliado a esse aumento de complexidade dos sistemas informáticos temos a configurabilidade, que é também cada vez maior! [CDS07]

Este último factor prende-se com a necessidade de mudança e adaptação dos sistemas modernos a novas realidades, de modo a satisfazer os requisitos dos utilizadores e/ou clientes de forma eficiente e com eficácia [KMS03, ASM03, RSMM03]. De tal forma que as empresas de software, por pressão do mercado, são forçadas

[1] Soluções Críticas Empresariais

[2] Sistemas de Garantia de Qualidade

a atingirem os objectivos propostos — eficácia — utilizando menos recursos (e.g. tempo, dinheiro) — eficiência — de forma a se manterem competitivas [Dru02].

De acordo com `Frederick P. Brooks`[3], Engenheiro de Software e Cientista da Computação:

> "*The hardest single part of building a software system is deciding precisely what to build. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.*" [Bro95]

Sendo por isso a construção de **sistemas de software configuráveis** uma mais valia para superar a constante evolução e mudança de requisitos. Mas, no entanto, a possibilidade de configurar o software à medida, torna o software mais vulnerável, na medida em que permite que sejam introduzidos erros [ASM03].

No contexto acabado de descrever, constata-se que, cada vez mais, o software é composto por:

- **código** - que contém os `requisitos imutáveis` do negócio;
- **configurações** - que contêm os `requisitos` do negócio `que podem sofrer alterações`.

Esta característica vem criar um *overhead* adicional no contexto do desenvolvimento dos sistemas aplicacionais na medida em que a verificação e validação destes sistemas não é totalmente coberta pelas metodologias usualmente utilizadas [CG09, CGT09, CDS07, Qu09, CDS08, LTL+08]. Não só precisamos de efectuar testes ao código, mas também às configurações que são passadas como *input* ao programa.

Definição 1. ***Configurações:*** *são tipicamente opções que têm um conjunto de valores possíveis que podem ser seleccionados no contexto de uma determinada aplicação [NN09], por forma a* `personalizá-la` *[Qu09], produzindo um sistema aplicacional específico [CDS07]. Diferentes sistemas aplicacionais poderão ler ou carregar as suas configurações de diferentes formas. Através de* `flags` *numa linha de comandos, ficheiros de configuração, variáveis de ambiente, ou ainda através de disponibilização de painéis de configuração [NN09].*

A fim de construirmos não só o software de forma correcta — **verificação**, mas também construirmos o software correcto[4] — **validação**, não podemos restringir os nossos testes ao código da aplicação. Se queremos ter a certeza de que o nosso software é *bem comportado*, precisamos assegurar que as configurações importadas são também testadas, e que irão funcionar de acordo com as espectativas do cliente/utilizador.

[3]Homepage: `http://cs.unc.edu/~brooks/`

[4]software right vs. right software

Neste sentido importa deixar claro que este trabalho de mestrado se foca portanto na "validação" de software; e não na "verificação". Os dois conceitos foram já apresentados e é bastante importante compreender as diferenças entre ambos. No Capítulo 3 - `Verificação & Validação de Software` encontram-se as definições de ambos os conceitos: **verificação** (Definição 2) e **validação** (Definição 3).

1.2 Objectivos

Considerando o que acima foi dito e tendo por lema "*construir software altamente configurável, garantindo que os requisitos são cumpridos e que as configurações que os exprimem definem o comportamento esperado*" é fácil identificar os objectivos do trabalho de mestrado aqui definidos:

- estudar as tecnologias & metodologias existentes actualmente utilizadas na validação e certificação de software, em específico no que diz respeito a aplicações altamente configuráveis;
- desenvolver uma estratégia de validação do comportamento do software resultante da aplicação de configurações;
- analisar ferramentas/linguagens de restrição existentes para especificar os requisitos que se querem validar;
- desenvolver uma aplicação que integre a estratégia de validação do comportamento do software com a linguagem de especificação que define as regras que têm de ser validadas, e que certifique o cumprimento (ou não) dessas mesmas regras;
- analisar a viabilidade da solução encontrada face às abordagens actuais e em que contribui para o aumento da certificação de software.

Para clarificar, note-se que, aceitando que o código está correcto, o que se quer é ver que a execução desse código controlado pelas configurações cumpre os requisitos.

Tradicionalmente, as abordagens à validação e verificação do software caem sobre os métodos formais, ou sobre os testes aplicacionais [WF89, CG09].

No entanto, tais abordagens centram-se essencialmente no código das aplicações, e não nas configurações, como forma de certificar e garantir a qualidade dos projectos de software.

Pelo contrário, o trabalho que se pretende desenvolver nesta tese, assume que o software base está correcto (em termos de código) e dedica-se à certificação do software altamente configurável, de modo a garantir que as configurações empregues no mesmo produzem os efeitos pretendidos.

Assim, pretende-se estudar as metodologias e frameworks dedicadas à validação de software e testes aplicacionais e, desenvolver (com base nesses principios) um sistema (protótipo) que verifique se as configurações agregadas (*ficheiro de configuração*) a uma determinada aplicação foram bem definidas e cumprem as restrições do negócio que se lhe impõem, não provocando `danos` nem prejuízos para os seus utilizadores.

Para tal, o protótipo desenvolvido deverá permitir a introdução de regras, através de linguagens de restrição(`Constraint Languages` - exemplo: JML, OCL), que uma determinada aplicação (configurável) terá obrigatoriamente de cumprir.

A ideia é serem os utilizadores a identificar essas regras. Estes utilizadores, com base na sua experiência sobre o negócio, identificarão mais facilmente os problemas nas configurações [NN09]; o que resultará, com o tempo, em regras mais *refinadas* e, consequentemente numa validação cada vez mais *apurada*.

De um modo geral o protótipo, com base nas *Regras de Validação* (`RV`) (regras de negócio) introduzidas, deverá indicar se uma dada aplicação após a fase de configuração, cumpre essas mesmas regras. Ou seja, se a configuração introduzida na aplicação não invalida os pressupostos da mesma.

1.3 Contributos

Deste trabalho de mestrado resultaram os seguintes contributos:

- **contributo teórico** — pela proposta de uma metodologia eficaz para a validação deste tipo de sistemas;
- **contributo prático** — pelo desenvolvimento de uma ferramenta que implementa a metodologia definida;
- **contributo pragmático** — pela aplicação da ferramenta desenvolvida a um caso de estudo concreto.

1.4 Tese

A tese defendida nesta dissertação consiste em afirmar que, em sistemas altamente configuráveis, o processo de validação passa por três etapas:

1. **validar o código da aplicação:** garantindo que este se encontra correctamente testado e aceite;
2. **validar as configurações:** garantindo que estas se encontram correctamente definidas e são válidas na aplicação;
3. **validar a aplicação configurada:** ou seja, validar que as configurações (que definem regras de negócio) importadas no contexto da aplicação se encontram de facto correctas e definem o comportamento pretendido.

E mais ainda, que é possível construir um sistema validador através do suporte:

(i) à definição de regras de negócio a validar num dialecto específico;

(ii) à geração do modelo de domínio da aplicação;

(iii) à captura do conjunto de factos do sistema a validar;

(iv) à implementação de um algoritmo de condição-reacção em que se verificam as regras despoletadas pelos factos e se reagem em consonância.

1.5 Estrutura do Relatório

Após este capítulo de introdução, o documento encontra-se estruturado da seguinte forma:

O Capítulo 2, `O Problema`, fornece uma descrição detalhada do problema. Nele indicamos a origem do problema, e relação com os objectivos; proposta de solução e principais (sub)problemas a resolver.

O Capítulo 3, `Verificação & Validação de Software`, apresenta o processo de uma forma geral no contexto do desenvolvimento de software com qualidade. Nele apresentamos as definições de cada um dos conceitos bem como as tarefas envolvidas, explicando a distinção entre ambos.

O Capítulo 4, `Abordagens Típicas na Validação de Software Altamente Configurável`, apresenta as abordagens tipicamente seguidas para validar software altamente configurável.

O Capítulo 5, `Descrição de Configurações`, começa por apresentar algumas das várias abordagens possíveis para o desenvolvimento de sistemas de software configuráveis; passando posteriormente por apresentar diferentes tipos de sistemas configuráveis; e termina apresentando a abordagem que foi tida para o caso de estudo deste trabalho de mestrado.

O Capítulo 6, `Classificação de Erros no Software Configurável`, é dedicado à identificação dos erros que ocorrem tipicamente numa aplicação configurável por má definição das regras de negócio nas configurações; bem como a estratégia de validação que se irá seguir de modo a evitar estes mesmos erros.

O Capítulo 7, `Arquitectura da Solução - Validador`, apresenta a arquitectura que satisfaz as necessidades do `Validador` — numa primeira fase através de uma visão mais geral e superficial, e posteriormente detalhando-se cada um dos componentes que o integram.

O Capítulo 8, `Desenvolvimento do Sistema de Validação`, demonstra a viabilidade da solução através da implementação de uma Prova de Conceito.

O Capítulo 9, `Avaliação da Solução`, pretende expor sumariamente o problema que temos em mãos no âmbito da `Plataforma Eleitoral`, bem como avaliar o resultado da solução aplicada nesse mesmo contexto.

O Capítulo 10, `Conclusão e Trabalho Futuro`, dá esta dissertação por concluída. Nele são destacados os principais resultados atingidos, bem como algumas sugestões de trabalho futuro.

O Apêndice A, `Regras de Validação para o Caso de Estudo`, expõe uma lista de regras de validação com origem no caso de estudo — Plataforma Eleitoral.

O Apêndice B, `Ficheiros de Configuração - Autárquicas`, lista (parcialmente) os ficheiros utilizados pela Plataforma Eleitoral para configurar umas eleições Autárquicas.

O Apêndice C, `Conteúdo do ficheiro - Rules.drl`, mostra o conteúdo do ficheiro *Rules.drl* utilizado pelo Validador na Prova de Conceito.

O Apêndice D, `JBoss Drools`, descreve de uma forma ampla algumas das principais potencialidades do Drools.

O Apêndice E, `Leis de DeMorgan`, enumera as primeiras e segundas leis de DeMorgan.

Capítulo 2

O Problema

"A prudent question is one-half of wisdom."

Francis Bacon

O que se pretende portanto resolver com este trabalho de mestrado acenta, como já foi referido, na necessidade de `certificar software altamente configurável`.

A mudança e adaptação dos sistemas informáticos, aliada à necessidade de se satisfazerem os requisitos de negócio do cliente e que conduziu ao emergir desta nova caracteristica, que é a configurabilidade, introduz também novas vulnerabilidades nestes sistemas.

Acontece que, apesar de ser possivel a existência de um *backoffice* para gerir a aplicação, não será esse *backoffice* que permitirá validar as configurações introduzidas. Relembre-se que, **é fundamental manter o carácter configurável** destas aplicações. Assim sendo, não é razoável adicionar essa validação das configurações à aplicação, pois estaria-se a incutir comportamento especifico na lógica da aplicação.

Nesta dissertação pretende-se cobrir aplicações cujas regras de negócio são definidas pelas configurações importadas (através de ficheiros ou utilizando uma outra interface criada para o efeito) em tempo de execução, com a agravante de serem conhecidas, muitas vezes, pouco tempo antes de serem *postas à prova*. De tal forma que não existe tempo para que sejam executadas verificações manuais ou testes sobre essas configurações. Por essas razões, o risco de ocorrerem resultados inesperados por introdução de *bugs* provenientes das alterações efectuadas nas configurações ser **altamente crítico** [BB08].

A Plataforma Eleitoral, utilizada como caso de estudo nesta tese, é um exemplo de um sistema aplicacional que recorre à configuração dos requisitos de cliente utilizando ficheiros de configuração, os quais são carregados para o sistema permitindo adaptar a aplicação a uma ou outra eleição, consoante as necessidades. Tal tarefa é realizada já em tempo de execução, portanto com a fase de implementação fechada e, qualquer falha num desses ficheiros pode causar danos irreversíveis no processo eleitoral.

Pelas razões já referidas, o objectivo passa por automatizar a validação de configurações resultantes da importação de vários ficheiros de configuração, que possam existir, e que definem o negócio e a lógica de uma aplicação. Esse processo de validação da aplicação não deve diminuir ou restringir o seu comportamento e flexibilidade, oferecendo garantias no cumprimento das restrições e requisitos estabelecidos pelo utilizador/cliente. Simultaneamente interessa documentar e fundamentar a metodologia aplicada e estudar em que medida é aplicável noutros sistemas semelhantes.

O que se pretende realizar, não é validação individual dos vários ficheiros de configuração existentes (tal tarefa deve ser efectuada pela própria aplicação) muito menos interessa *olhar* para os ficheiros ou para o código da aplicação[1]. Interessa sim, validar as configurações definidas no contexto da aplicação face a um conjunto de regras definidas com o intuito de assegurar o "correcto" comportamento da mesma.

2.1 A Origem

Pode-se dizer que uma aplicação de software, que siga um modelo de desenvolvimento mais tradicional (com base no modelo `waterfall` de Royce, W. [Roy70]), visível na Figura 2.1, composto por uma série de tarefas como análise e captura de requisitos, concepção da solução, implementação e teste [Roy70, Tur96, SZ07, GV05, DBC88], se encontra validada se [ABD+04]:

- cumpre com os requisitos não só de negócio mas também os requisitos técnicos, tidos em conta no processo de concepção e desenvolvimento;
- o comportamento observado resultado de determinado `input` é o esperado.

Ora, sendo o processo de validação e teste dirigido pela metodologia de desenvolvimento adoptada [Das07, BKS03], quando se fala de **sistemas configuráveis**, os items acima referidos por si só, não são suficientes para garantir que a aplicação se encontra validada.

Exemplo 1. *Tome-se, como exemplo, o caso de um sistema aplicacional de emissão e processamento de facturas. Sobre a aplicação é possível configurar uma série de propriedades ou características dos bens ou serviços a serem facturados aos clientes. Agora, imaginemos que, por engano, nas configurações que foram inseridas os valores das taxas (ou mesmo do IVA[2]) foram erradamente introduzidos e quando o erro foi detectado já as facturas haviam sido enviadas aos clientes, não havendo forma de as recuperar. Estes e outros erros na emissão e processamento de facturas são frequentes e podem incrementar os custos de tais operações para as empresas.*

Ora isto acontece não porque a aplicação esteja mal concebida, mas sim porque a configuração estava errada!

[1] considerado como testado e aceite.

[2] IVA: Imposto sobre o Valor Acrescentado.

Acontece que nos sistemas configuráveis o processo de desenvolvimento de software é composto, não só pelas tarefas já referidas, mas também por uma tarefa complementar de **configuração**, o que provoca uma alteração no ciclo de vida destas aplicações [Gao10, PS08].

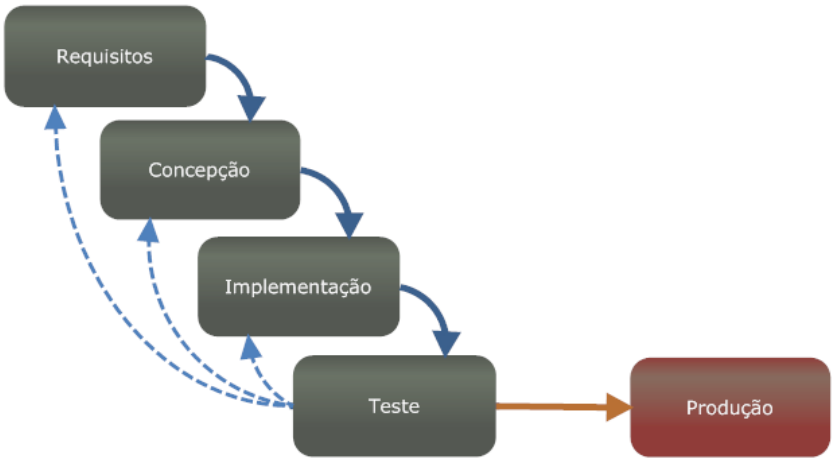

Figura 2.1: Ciclo de Vida Standard do Software

Esta alteração transforma o Ciclo de Vida Standard apresentado na Figura 2.1, num ciclo de vida com um novo componente que introduz um novo tipo de vulnerabilidades nas aplicações, como se ilustra na Figura 2.2.

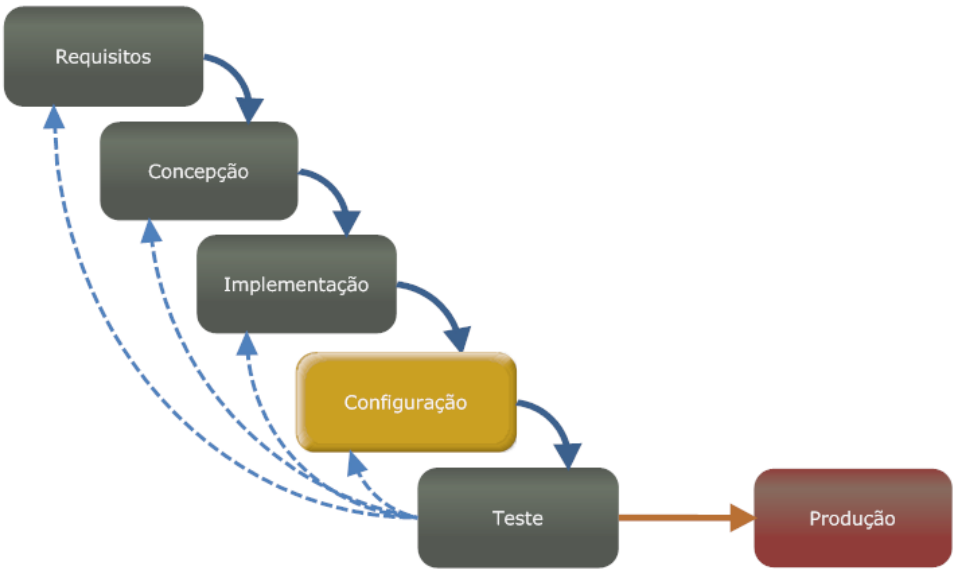

Figura 2.2: Ciclo de Vida do Software Altamente Configurável

Mas cuja vulnerabilidade é objecto de estudo desta dissertação, e a qual se pretende resolver através da validação adicional das aplicações após a configuração.

Validação essa que é díficil de concretizar manualmente devido às inúmeras configurações possíveis que um pequeno conjunto de variáveis permitem gerar [NN09,

CDS07]; tornando impraticável verificar que todos os pressupostos se mantêm; não só porque consome muito tempo, como também é altamente propenso a erros [CDS07].

Assim sendo, quer-se transformar o ciclo de vida já apresentado na Figura 2.2 resolvendo a vulnerabilidade através da introdução de uma "camada" de protecção que diminua o risco que as configurações trazem para os sistemas altamente configuráveis.

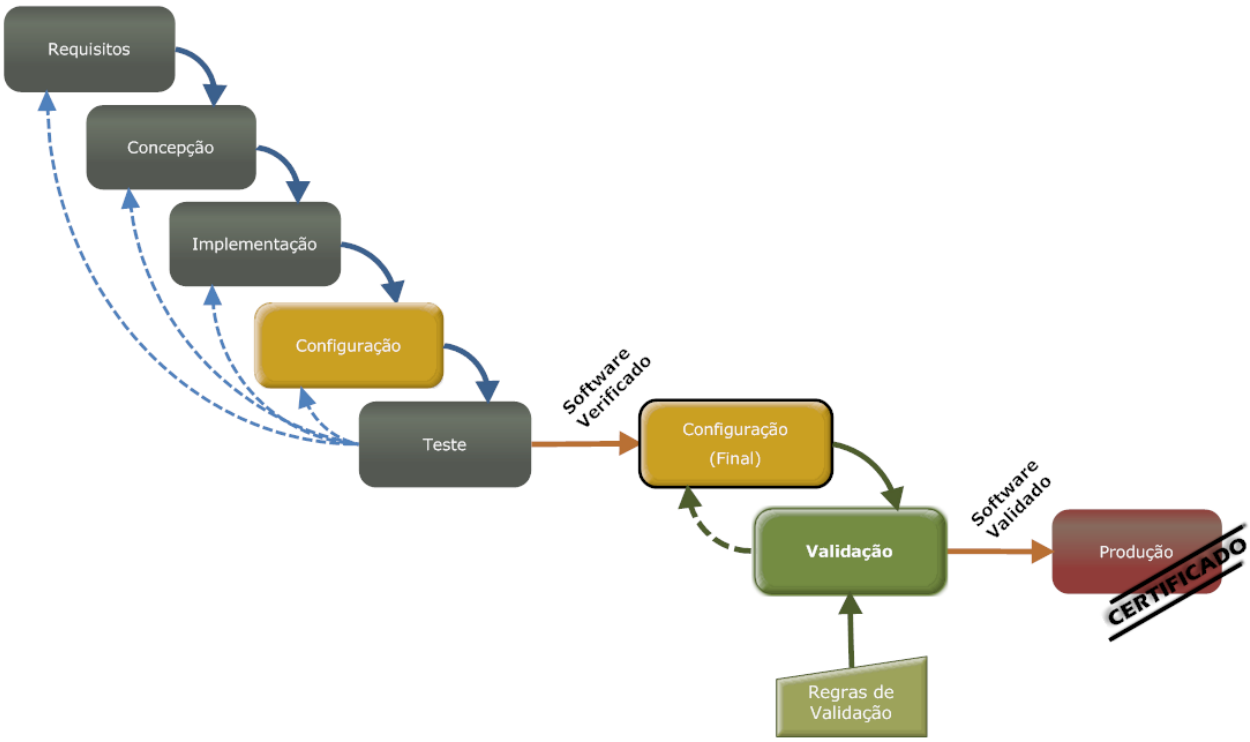

Figura 2.3: Ciclo de Vida do Software Altamente Configurável *Protegido*

A protecção apresentada na Figura 2.3 consiste num processo de validação do software, a ser executado após a verificação do mesmo. Tal processo deverá ser executado sempre que haja a necessidade de se alterarem as regras de configuração da aplicação *standard*. E, contará com um conjunto de `RV` definidas pelo cliente/utilizador da mesma. O resultado deste processamento deverá, tendo como base as `RV` fornecidas, indicar se a aplicação *standard* configurada é válida.

Tendo em conta o exemplo apresentado (sistema de controlo de stocks de cadeia de supermercados), imagine-se que alguém coloca no ficheiro de configuração todos os produtos como indisponíveis para venda. E agora, imagine-se que o cliente definiu uma regra de validação que indica que se um determinado produto existe, então pelo menos 'um' tem de estar disponível para venda. Então devemos ser capazes de identificar essa falha na configuração, evitando problemas maiores.

2.2 A Proposta

O que se propõe é que, após o processo usual de configuração da aplicação *standard*, seja executada uma "prova dos nove" (validação) que permita validar se as configurações importadas no contexto da aplicação cumprem o conjunto de assumpções e pressupostos definidos, usualmente, pelo cliente ou utilizador (em ambos os casos requer que tenham conhecimento claro do modelo de negócio), tal como se ilustra na Figura 2.3.

Para se poder concretizar tal operação, é necessário dar resposta a um conjunto de *sub-problemas*, os quais serão abordados nesta dissertação. São eles os seguintes:

- como identificar a informação da aplicação *standard* alvo necessária para validar as assumpções e pressupostos (`RV`) definidos;
- como definir e interpretar as `RV` a serem utilizadas no processo de validação;
- como aplicar as `RV` sobre a aplicação *standard* configurada por forma a tirar ilações sobre o cumprimento (ou não) das mesmas.

Identificados os principais problemas que terão de ser resolvidos, pode-se resumir este trabalho de mestrado numa questão:

> *Como validar que as configurações, que definem regras de negócio, aplicadas num sistema altamente configurável, estão de acordo com a lógica pretendida pelo cliente/utilizador, aumentando a confiança nessas configurações, sem que para tal se perca flexibilidade aplicacional?*

Capítulo 3

Verificação & Validação de Software

"Quality is everyone's responsibility."

W. Edwards Deming

Produzir software com qualidade é cada vez mais um factor importante no desenvolvimento de aplicações informáticas. Mas a noção de qualidade não é tão simples como aparenta, podendo ser vista de várias perspectivas e entendida de formas diferentes.

Existe, no entanto, um *standard* estabelecido pelo ISO9001-00, juntamente com as linhas orientadoras para aplicação ao software [II04], que definem qualidade como sendo "o grau para o qual um conjunto de características inerentes fazem cumprir requisitos".

Neste sentido, a *Verificação e Validação* (V&V) surge como um conjunto de processos de gestão de qualidade do software, que se tornou importante com o aumento da complexidade destes sistemas, e que aplicado em várias fases do ciclo de vida do software [ABD+04, WF89] se esforça por garantir, de um modo geral:

- que a qualidade é incorporada no software;
- que os requisitos especificados são implementados;
- que a implementação do software satisfaz as necessidades dos utilizadores.

O desenvolvimento de software tem sofrido bastantes alterações ao longo dos tempos: antes o software era composto por poucos módulos realizando pequenas simples tarefas; agora é composto por muitos módulos que desempenham tarefas mais complexas. Esta evolução provocou também uma alteração no processo de V&V do software, passando de um processo informal e realizado pelo próprio engenheiro de software, para um processo formal indispensável para a garantia de qualidade [MD08, Tra99] composto por tarefas aplicáveis em diferentes etapas do ciclo de vida do software e, executado por vezes por entidades ou organizações independentes da equipa de desenvolvimento.

A importância da V&V para a certificação do software reconhece-se porque, enquanto que os testes de sistema permitem revelar apenas a existência de erros ou defeitos na aplicação, os processos de V&V, pelo contrário, pretendem dar maior confiança sobre a ausência dos mesmos [Tra99, Rak01, MFRP06], embora não se possa garantir a sua ausência.

3.1 Dicotomia: Verificação vs. Validação

Verificação e Validação são actividades muitas vezes confundidas entre si. No entanto, são actividades com limites e objectivos bem definidos e bastante distintos que, de um modo geral, se complementam.

Em termos simples, podemos dizer que **verificação** acenta na tentativa de garantir que o produto é construído correctamente, enquanto que **validação** acenta na tentativa de garantir que é construído o produto correcto.

Definição 2. ***Verificação:*** *processo de avaliação de um sistema, ainda em desenvolvimento, de forma a garantir o cumprimento das condições estabelecidas no início de uma determinada fase do seu ciclo de vida. [Rak01, Das07, WF89, Ei90]*
Responde à pergunta: ***"O sistema satisfaz a especificação?"***

Definição 3. ***Validação:*** *processo de avaliação de um sistema, normalmente no final do seu desenvolvimento, de forma a garantir o cumprimento dos resquisitos especificados. [Rak01, Das07, WF89, Ei90]*
Responde à pergunta: ***"A especificação satisfaz as espectativas do cliente?"***

Quando se fala em verificação de software, estamo-nos a referir a um conjunto de actividades como revisões e reuniões para discutir e avaliar documentação, planos, código, requisitos e outras especificações, que em conjunto pretendem demonstrar que a implementação segue a arquitectura definida.

Pelo contrário, quando se fala em validação, referimo-nos a um conjunto de actividades que tipicamente envolvem testes ao próprio produto, que em conjunto pretendem demonstrar que o produto construído é o correcto, isto é, que o comportamento do produto não só é consistente com os requisitos definidos, como vai de encontro às necessidades do cliente.

Apresenta-se agora, como exemplo, uma situação que faz parte do quotidiano de qualquer um, de forma a mostrar as diferenças entre ambos os processos. Após o exemplo aqui dado, deverá ser mais fácil distinguir quando se fala de verificação ou de validação.

Exemplo 2. *Imaginemos que temos definida a seguinte propriedade no funcionamento de um elevador:*

> *Se um utilizador pressionar o botão de chamada do elevador no piso* ***i****,*
> *o elevador deverá chegar rapidamente a esse mesmo piso.*

Ora, a propriedade acima referida pode ser validada, mas não verificada ("rapidamente" é um quantificador subjectivo).
No entanto, se tivermos a propriedade:

> *Se um utilizador pressionar o botão de chamada do elevador no piso* ***i****, o elevador deverá chegar a esse mesmo piso no espaço de* ***30 segundos****.*

Esta propriedade tanto pode ser validada como verificada (30 segundos é um quantificador bastante preciso).

3.2 Processo de Verificação & Validação

O principal motivo que leva a que este processo seja bastante utilizado, deve-se ao facto da sua importância junto do cliente ou utilizador final da aplicação desenvolvida — o processo de `V&V` fornece um grau de confiança adicional de que o dito software está apto para servir o propósito para o qual foi concebido. Não significando, no entanto, que o software se encontra completamente livre de defeitos, mas que se encontra bem o suficiente para lhe ser dado o uso previsto.

O processo de verificação deve acontecer antes do processo de validação, pois não faz sentido algum validar que um produto faz o que é suposto antes de se verificar que o produto foi construído correctamente.

O processo de `V&V`, é um processo que se foca na qualidade dos produtos de software[1], sendo composto por inúmeras técnicas tanto de verificação como de validação, e que são aplicáveis em diferentes fases do ciclo de vida do desenvolvimento do software.

Nos próximos parágrafos as atenções estão centradas em dar-se uma resposta à seguinte pergunta: *Que técnicas utilizar no processo de* `V&V` *no contexto de um determinado projecto de software?*

Consoante o tipo de utilização, as expectativas dos utilizadores e o nível de criticidade do software para uma organização, determina-se o grau de confiança que será necessário garantir para a aplicação a desenvolver.

Por exemplo, se estivermos a comparar o desenvolvimento de um editor de texto (e.g. `vim`) com uma aplicação *web* do tipo "loja online", é fácil perceber que a segunda aplicação requer atenção para coisas como a segurança, a protecção dos dados e a disponibilidade do serviço, em muito devido não só ao tipo de aplicação mas também devido à natureza crítica dos dados com que irá trabalhar. De tal forma que, para que os utilizadores se sintam à vontade em utilizar a segunda aplicação, há a necessidade de garantir confiança em mais níveis (para cada uma das características referidas) do que na primeira aplicação, na qual as preocupações acima referidas não existem.

[1] Enquanto que *Quality Assurance* se foca na qualidade dos processos de desenvolvimento de software.

Por forma a dotar o software com o grau de confiança exigido existem uma enorme série de factores a serem tidos em conta quando se está a efectuar o planeamento das tarefas de V&V que serão necessárias executar no decorrer do desenvolvimento de um sistema aplicacional.

Como já foi referido, existem inúmeras técnicas possíveis para se fazer V&V. Não podemos, nem devemos, no entanto aplicar todas as técnicas existentes. Pois o facto de utilizarmos mais técnicas não fará com que o software final fique com mais qualidade. É necessário, consoante as situações determinar, quais as técnicas mais indicadas a aplicar por forma a atingirmos os objectivos que se pretendem. Para além disso deve-se salientar que nem todas as técnicas são aplicáveis em qualquer situação. E mais importante ainda, o facto de que aplicar estas técnicas de V&V, tal como outra técnica ou processo do desenvolvimento de software, necessita de tempo e, terá custos adicionais para o cliente (caso exista um).

Uma das etapas mais importantes no planeamento do processo de V&V é a decisão das técnicas (e ferramentas) a serem utilizadas no decorrer do desenvolvimento de um projecto de software. Antes de mais, é necessário perceber que esta não é uma tarefa fácil. O tipo do software alvo e as suas características têm impacto directo na selecção das técnicas e ferramentas — se se trata de uma aplicação orientada aos objectos (e.g. java, c#), se é uma aplicação distribuída, se é uma aplicação web ou móvel, entre muitas outras propriedades — vai fazer com que se decida o que utilizar. Os factores mais relevantes dizem respeito ao **tempo** existente para a execução do projecto e mais concretamente a porção de tempo que será possível dedicar às tarefas de V&V; de modo intrínseco temos o **custo** da execução dessas tarefas, tanto em termos económico-monetários como em termos de esforço (recursos-humanos); o nível de **desempenho, segurança e fiabilidade** exigidos, bem como o **risco** associado ao projecto são variáveis a ter em conta; outros factores como o **tamanho da equipa** do projecto ou o ***know-how*** da mesma influenciam de certo modo o referido planeamento; mas o factor primordial a ter em conta é sem dúvida o **cliente** do projecto e as características do último.

3.2.1 Principais Objectivos

O processo de V&V tem como principais objectivos:

- garantir a ausência de defeitos no sistema avaliado;
- avaliar se o sistema se encontra em conformidade para ser colocado em produção;
- instituir confiança no sistema que foi desenvolvido.

Mas, para além dos objectivos acima apresentados, que estão directamente relacionados com o sistema em desenvolvimento, temos, quando nos encontramos num ambiente de desenvolvimento empresarial, razões como:

- praticar custos de suporte e manutenção mais baixos;
- aumentar a satisfação dos clientes;
- aumentar a eficiência na utilização dos recursos de engenharia de software.

Para sistemas "não críticos", os procedimentos de `V&V` são frequentemente postos de parte ou executados apenas parcialmente em algumas fases de desenvolvimento, por serem bastante custosos[KN96]; já para **sistemas críticos** estes procedimentos são absolutamente necessários para garantir a construção de um sistema de elevada fiabilidade [Tra01]. Importa agora definir o que é um sistema crítico:

Definição 4. ***Sistemas Críticos:*** *são sistemas de software em que a existência de defeitos pode ter impacto dramático sobre a vida humana, sobre o meio ambiente ou sobre o património (bens activos). Caracterizam-se pelos seus elevados padrões de fiabilidade, disponibilidade, segurança e protecção dos dados/procedimentos que executam [HCL07, RS99, Kni02]. São exemplos os sistemas de controlo de tráfego aéreos; sistemas de controlo de semáforos num cruzamento; o sistema de ABS (Anti-lock Brake System) existente nos automóveis; sistema médico de processamento de imagens utilizado para fazer diagnósticos ou planear tratamento de um paciente; entre outros.*

3.3 Técnicas e Ferramentas

Nesta secção entraremos em detalhe nomeadamente no que diz respeito às técnicas e ferramentas usadas para o processo de `V&V`.

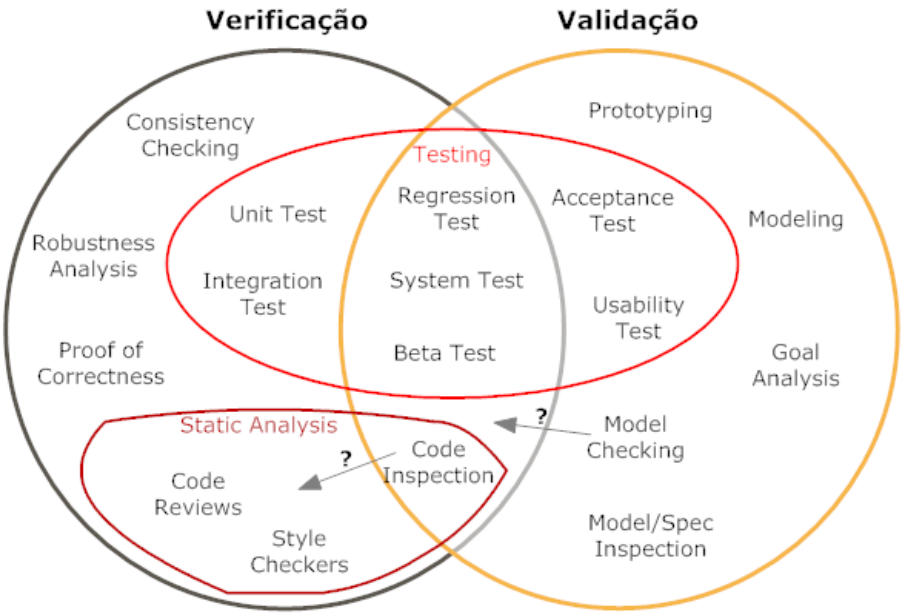

Figura 3.1: Técnicas de Verificação & Validação — adaptado de [Eas08]

Pelo que podemos observar na Figura 3.1, existem técnicas que se aplicam unicamente na verificação (ou na validação), mas existem também um conjunto de técnicas que são comuns aos dois processos (localizadas na intersecção das duas circunferências) — que estão identificadas por serem na sua maioria técnicas de teste. Para além disso note-se que ambos os conjuntos se encontram bastante populados. Mais ainda se realçam na imagem as técnicas que dizem respeito a tarefas de testes que são não só as técnicas mais conhecidas mas também as mais aplicadas.

3.3.1 Técnicas e Ferramentas de Verificação

As técnicas de verificação tradicionalmente utilizadas são, de um modo geral, os **testes**, as **técnicas formais** apoiadas em especificações, e técnicas com base em **análises estáticas**. Em seguida, apresenta-se uma listagem um pouco mais detalhada das técnicas usadas no processo de verificação, agrupadas em duas categorias:

- **Verificação Formal** - são técnicas matemáticas baseadas em especificações formais do que é pretendido. Deste grupo fazem parte técnicas como: *Proof of Correctness*.

  Como linguagens formais para especificação temos: VDM-SL e Z. No que diz respeito a ferramentas disponíveis temos: VDMTools[2], Overture[3], SpecBox[4] para o **VDM-SL**; CZT - Community Z Tools[5] para o **Z**.

- **Testes** - este grupo pode ser dividido em dois subgrupos:

  - **Testes Dinâmicos** - composto por técnicas de teste que envolvem a execução do sistema ou de um componente. Fazem parte deste grupo as seguintes técnicas: *Unit Test* (testes unitários), *Integration Test* (testes de integração), *Regression Test* (testes de regressão), *System Test* (testes de sistema), *Beta Test* (testes "beta").

    No que diz respeito a **ferramentas** disponíveis temos, para as técnicas referidas: JUnit[6], TestNG[7] para os testes unitários; WinRunner[8] para testes de regressão, JSystem[9] para testes de sistema; entre muitas outras.

  - **Testes Estáticos** - composto por técnicas suportadas em *Static Analysis* (Análises Estáticas), que não envolvem a execução do sistema. Fazem parte deste grupo as seguintes técnicas: *Robustness Analysis*, *Consistency Checking*, e ainda os *Style Checkers*, as *Code Reviews*, *Code Inspections*.

    Em respeito a **ferramentas** disponíveis para as técnicas referidas, temos: ferramentas UML como o *Visual Paradigm* (VP)[10] e o *Enterprise Architect*

[2] VMDTools — http://www.vdmtools.jp/en/
[3] Overture — http://www.overturetool.org
[4] SpecBox — http://www.adelard.com/services/SoftwareTools/SpecBox/index.html
[5] CZT — czt.sourceforge.net
[6] JUnit — http://www.junit.org/
[7] TestNG — http://testng.org/
[8] WinRunner — http://c2.com/cgi/wiki?WinRunner
[9] JSystem — http://www.jsystemtest.org/
[10] VP — http://www.visual-paradigm.com/

(EA)[11] para *Robustness Analysis*; UML/Analyzer[12], cpntools[13] (Petri Nets), MCC (plugin para o Poseidon) para *Consistency Checking*; PMD[14], FindBugs[15], Checkstyle[16] para *Static Analysis*; entre muitas outras.

Figura 3.2: Técnicas de Verificação no Ciclo de Vida do Software

Na Figura 3.2 apresentamos a distribuição das técnicas de verificação aqui referidas pelo ciclo de vida do software, tendo como base o ciclo de vida já apresentado na Figura 2.1.

3.3.2 Técnicas e Ferramentas de Validação

Quando o objectivo é a Validação, as técnicas tradicionalmente utilizadas são: as **inspecções** e **análises** a modelos, **prototipagem**, e o mais comum — os **testes**. Tal como para as técnicas de verificação, iremos agora também apresentar as técnicas mais usadas agrupadas em quatro categorias:

- **Testes Dinâmicos** - técnicas que envolvem, como já referido, a execução do sistema ou componente. Deste grupo fazem parte as seguintes técnicas: *Acceptance Test* (testes de aceitação), *Usability Test* (testes de usabilidade), *Regression Test* (testes de regressão), *System Test* (testes de sistema), *Beta Test* (testes "beta").

  Em respeito a **ferramentas** disponíveis para as técnicas referidas, temos a exemplo: Arbiter[17] e o FitRunner[18] para os Teste de Aceitação; Userfly[19], Sil-

[11] EA — http://www.sparxsystems.com/products/ea/index.html
[12] UML/Analyzer — http://www.alexander-egyed.com/tools/uml_analyzer_tool.html
[13] cpntools — http://wiki.daimi.au.dk/cpntools/cpntools.wiki
[14] PMD — http://pmd.sourceforge.net/
[15] FindBugs — http://findbugs.sourceforge.net/
[16] Checkstyle — http://checkstyle.sourceforge.net/
[17] Arbiter — http://arbiter.sourceforge.net/index.html
[18] FitRunner — http://fitrunner.sourceforge.net/
[19] UserFly — http://userfly.com/

verback[20], Chalkmark[21], UserVue[22], *Google Website Optimizer* (GWO)[23], *Google Analytics* (GA)[24], entre muitas outras disponíveis para os Testes de Usabilidade; para os Testes de Regressão é possível utilizarem-se ferramentas de automação como as que já foram referidas; para as restantes técnicas, podem ser aplicadas ferramentas, como as já referidas para os testes automatizados; entre muitas outras.

- **Validação Formal** - categoria existente também nas técnicas de verificação. Deste grupo fazem parte as seguintes técnicas: *Model Checking*.

  Para esta técnica, deixa-se aqui, a referência para algumas das muitas **ferramentas** existentes: NuSMV2[25], Bandera[26], SLAM[27].

- **Dependability Analysis (Análises de confiança)** - categoria constituída pelas técnicas: *Goal Analysis* (análise de objectivos), *Code Inspection*, *Modeling*, *Model/Spec Inspections*.

  Em respeito a ferramentas para este conjunto de técnicas temos: VP e o EA para *Modeling* e *Model/Spec Inspections*; IBIS[28], XATI[29] e CodeSurfer[30] para *Code Inspections*; e de um modo geral para todas as técnicas (incluindo para *Goal Analysis*) temos os processadores de texto como o Microsoft Word e o Acrobat Reader.

- **Prototyping (Prototipagem)** - por ser uma técnica bastante utilizada, destacam-se aqui diferentes tipos de protótipos: *protótipos de apresentação* (para explicação e demonstração de funcionalidades ao cliente), *protótipos de exploração* (usados para determinar problemas, recolher necessidades, clarificar objectivos), *protótipos experimentais* (para explorar viabilidades técnicas e testar a aplicabilidade de tecnologias), *protótipos evolutivos* (protótipos operacionais capazes de mostrar o desenvolvimento do sistema).

  Em termos de **ferramentas** de prototipagem podemos destacar algumas como: Gliffy[31], Balsamiq[32], Pencil[33].

Na Figura 3.3 apresenta-se a distribuição das técnicas de validação referidas, distribuidas pelo ciclo de vida do software, tendo como base o ciclo de vida apresentado na Figura 2.1.

[20] Silverback — http://silverbackapp.com/
[21] Chalkmark — http://www.optimalworkshop.com/chalkmark.htm
[22] UserVue — http://www.techsmith.com/uservue.asp
[23] GWO — https://www.google.com/analytics/siteopt/splash
[24] GA — http://www.google.com/analytics/
[25] NuSMV2 — http://nusmv.irst.itc.it/
[26] Bandera — http://bandera.projects.cis.ksu.edu/
[27] SLAM — http://research.microsoft.com/en-us/projects/slam/
[28] IBIS — http://cdg.di.uniba.it/index.php?n=Research.IBIS
[29] XATI — http://www.tol.oulu.fi/i3/tools/
[30] CodeSurfer — http://www.grammatech.com/products/codesurfer/overview.html
[31] Gliffy — http://www.gliffy.com/
[32] Balsamiq — http://www.balsamiq.com/
[33] Pencil — http://www.evolus.vn/Pencil/

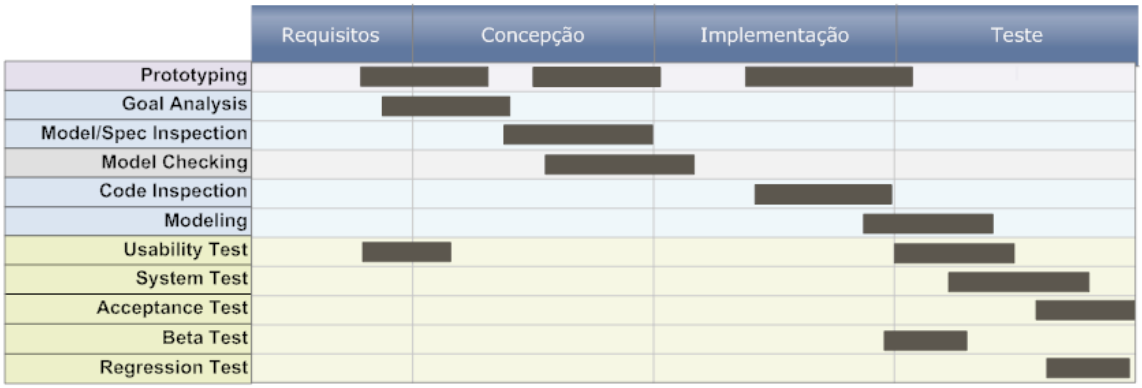

Figura 3.3: Técnicas de Validação no Ciclo de Vida do Software

3.4 Desenvolvimento Académico vs. Empresarial

Nesta secção o que se pretende é mostrar que a aplicação das técnicas de `V&V` é abordada de forma diferente dependendo se nos encontramos num ambiente de desenvolvimento tipicamente académico, ou num ambiente de desenvolvimento empresarial.

Quando nos encontramos num `ambiente académico` temos, a maioria das vezes, liberdade para seleccionar as técnicas e ferramentas mais adequadas para abranger todas as fases do ciclo de vida do desenvolvimento da aplicação por forma a tirar o maior partido dos processo de `V&V`.

No entanto, quando num `ambiente empresarial`, a primeira questão que se coloca é: "*quanta verificação e validação é suficiente para atingirmos os objectivos pretendidos?*". É óbvio que, quanto mais verificação e validação aplicarmos melhor se encontrará a aplicação desenvolvida. No entanto, o que é imprescindível saber, num ambiente de desenvolvimento empresarial é quando é que o custo e o tempo do projecto pesam mais do que os benefícios da execução das técnicas de `V&V`, o que varia de projecto para projecto.

Nos parágrafos seguintes pretende-se de certo modo dar um *overview* sobre a forma como os processos de `V&V` são postos em prática num ambiente empresarial — neste caso em específico na Critical Software S.A. — sendo dado especial atenção ao processo de validação.

No início de um novo projecto de software, é logo na fase de planeamento que se decide o esforço que vai ser gasto para os processos de `V&V`, tendo em consideração o cliente e os critérios estabelecidos para o projecto. É de notar que, apesar de nem todos os projectos/clientes exigirem a execução de métodos de `V&V`, estes são, por norma, sempre executados; e não apenas a pedido do cliente.

Os objectivos que se pretendem atingir são, tal como já referido neste mesmo capítulo, o de garantir qualidade nos projectos aumentando o grau de confiança e satisfação junto do cliente.

No conjunto de factores tidos em conta para decidir quais as técnicas a serem utilizadas temos: o cliente, que é sempre o mais importante; o nível do conhecimento do negócio; se há tempo e dinheiro.

Em termos do processo de validação, as técnicas a que é mais frequente recorrer são:

- Prototipagem (*Prototyping*);
- Testes de Aceitação (*Acceptance test*);

No que diz respeito à técnica de prototipagem, esta é realizada na fase de requisitos, sendo usada sempre que hajam requisitos que possam suscitar dúvidas na equipa de desenvolvimento. Quanto aos testes de aceitação são realizados em todos os projectos à excepção de situações extraordinárias em que o próprio cliente pretenda ele mesmo fazer os testes de aceitação, caso contrário são sempre realizados internamente.

Em termos de ferramentas, usam-se o Balsamiq[34], para desenhar protótipos e *mockups*; e o EA[35], para auxiliar na construcção de uma matriz de rastreabilidade entre os testes de aceitação e os requisitos da aplicação. Para a definição dos testes de aceitação existe um standard interno que indica como estes devem ser definidos bem como o procedimento a seguir na sua execução.

O processo de validação costuma portanto ser aplicado nas fases de Análise de Requisitos e de Validação, sendo a sua execução da responsabilidade dos "Gestores Técnicos" e dos *Software Product Assurance Engineer* (SPAE)'s do projecto.

Para terminar, é de salientar o facto de ser indispensável, para qualquer projecto, a execução de técnicas de validação pela importância que têm na garantia do cumprimento dos objectivos estabelecidos e do aumento da satisfação do cliente — sendo essa a melhor justificação para o tempo gasto neste processo.

3.5 Sumário

É fundamental deixar explícito que ambos os processos de verificação e validação do software são importantes para a construcção de software com qualidade. Mais ainda, importa salientar que são processos que contêm técnicas que se complementam umas às outras em vários aspectos, permitindo uma grande liberdade no modo de se fazer V&V.

Os processos de V&V estão cada vez mais evoluídos, não só no que diz respeito aos procedimentos — acompanhados por melhor documentação e suporte para as mais variadas situações permitindo a sua aplicação em inúmeros contextos — mas também no que diz respeito às ferramentas disponíveis — que são cada vez mais em maior número e com melhores resultados.

[34] http://www.balsamiq.com/
[35] http://www.sparxsystems.com/products/ea/index.html

A abundância de técnicas e ferramentas para executar estes processos é tal que podemos seleccionar as que melhor se adequam ao nosso contexto.

No entanto, mesmo seguindo os processos mais robustos de `V&V`, e utilizando as ferramentas mais eficazes, é possível que estas não cubram tudo. Pois, apesar da evolução tanto dos procedimentos como das ferramentas, os próprios sistemas também evoluem.

Este trabalho de mestrado surge da procura de um procedimento de validação, e ferramenta auxiliar, para colmatar uma "falha" na abordagem no contexto de aplicações altamente configuráveis.

Capítulo 4

Abordagens Típicas na Validação de Software Altamente Configurável

"Stay committed to your decisions, but stay flexible in your approach."

Anthony Robbins

Tal como Anthony Robbins refere, devemo-nos comprometer com os nossos objectivos/decisões, mas devemos ser flexíveis na abordagem que nos conduz até eles. Assim acontece também quando pretendemos validar software altamente configurável, assunto já mencionado no Capítulo 1.1 e que consiste no tema central desta tese. Existem inúmeras abordagens possíveis tendo em conta um objectivo em comum — validar software altamente configurável. Tais abordagens serão apresentadas neste capítulo, explicitando-se também o porquê de não satisfazerem os nossos objectivos.

De forma geral, as abordagens que serão aqui apresentadas, preocupam-se em investir recursos predominantemente na avaliação de um pequeno conjunto de configurações estabelecidas como sendo importantes, e sobre as quais se executam à *posteriori* técnicas de validação (como os `Testes de Aceitação` ou os `Testes de Regressão`), já referidas no capítulo anterior.

Todos os métodos utilizados para validar software altamente configurável deparam-se com o mesmo problema: as configurações individuais geram inúmeras possibilidades de combinação de configurações tornando praticamente impossível cobri-las a todas. A forma como este problema é endereçado passa por:

- seleccionar: reduzir o conjunto de configurações a testar às configurações que são mais importantes;
- prioritizar: ordenar as configurações para validar primeiro as configurações mais importantes.

Sendo que diferem entre si essencialmente na forma como seleccionam e/ou prioritizam as configurações que serão alvo de validação. É esta a principal diferença encontrada no processo de validação de um sistema altamente configurável e um outro sistema.

As abordagens seguidas para resolver o problema que surge no contexto das aplicações altamente configuráveis pode ser resolvido portanto através de técnicas como:

- ***Source Code Analysis*** [NN09] — técnica que pretende identificar e explorar a estrutura da aplicação(**white-box**) por forma a reduzir esforço de análise e testes, através da partilha dos resultados entre configurações relacionadas para que, testes e análises realizadas a uma determinada configuração possam ser aplicadas noutras. Esta técnica baseia-se no facto de que a maioria das configurações partilham quantidades significativas de código estando portanto relacionadas.

- ***Lifting*** [PS08] — é uma técnica que permite integrar a informação de configurações de uma aplicação de modo a facilitar a execução de técnicas como o `Model Checking` no contexto de sistemas altamente configuráveis. Sendo essencialmente usada para o processo de verificação, é no entanto útil para o *Model Checking* a qual é também uma técnica de Validação.

- ***Configuration Aware Prioritization*** [Qu09] — técnicas de prioritização cientes da existência de configurações na execução de testes de regressão. O desenvolvimento desta técnica surge da procura por uma alternativa às já existentes que se foque nas configurações das aplicações.

- ***Combinatorial Interaction Testing*** (`CIT`) [CDS07, RSM$^+$, CDS08] — método que visa seleccionar configurações para a execução de testes. Tendo como objectivo encontrar o menor conjunto possível de configurações que satisfaça uma condição. São inúmeras as variantes existentes: a técnica referida em [CDS07], *covering arrays*, baseia-se numa estrutura matemática que pode ser construída utilizando diferentes algoritmos e ferramentas, sendo aqui apresentada uma solução para lidar com restrições das configurações no âmbito das técnicas `CIT`; a técnica referida em [RSM$^+$] preocupa-se com o facto das interacções existentes entre configurações serem limitadas e que portanto pretende eliminar tais interacções das configurações a serem testadas. Existem técnicas que seguem abordagens **white-box** e outras **black-box**.

Importa agora definir o que são abordagens *white-box*, como as referidas acima, e em que diferem, por oposição, das abordagens *black-box*. Assim sendo temos [YMC01, BPS03]:

Definição 5. ***White-Box:*** *é uma abordagem que permite ver o sistema como se de uma "caixa transparente" se tratasse, permitindo ter conhecimento da estrutura interna da aplicação — abordagem estrutural.*

Definição 6. ***Black-Box:*** *é uma abordagem em que, pelo contrário, vê o sistema como uma "caixa negra", não havendo portanto conhecimento da estrutura interna da aplicação — abordagem funcional.*

Some call this "gray-box"or "translucent-box"test design, but others wish we'd stop talking about boxes altogether.

4.1 Discussão

Existem muitas abordagens para validar software altamente configurável. Algumas das abordagens seguidas para suportar a validação de configurações foram já aqui referidas, no entanto, nenhuma responde em concreto ao desafio a que nos propusemos nesta dissertação: validar aplicações altamente configuráveis dando ênfase aos objectivos de negócio passíveis de serem configurados em tempo de execução.

O que pretendemos não é uma abordagem *white-box*, mas sim uma abordagem *black-box*. Pois estamos interessados mais no carácter funcional da aplicação do que na sua estrutura. Para além disso, a maioria das abordagens, bem como as referidas, centram-se num problema de que as configurações de uma determinada aplicação geram inúmeras combinações sendo impraticável validar todas. E é por isso que investem esforço na selecção e prioritização das configurações de modo a que o conjunto de configurações a ser testado e validado seja menor (podendo haver configurações a que é dado menos atenção). Pelo contrário, o que pretendemos é centrarmo-nos na validação das configurações que serão usadas pela aplicação em determinada instância num determinado momento.

A maioria das abordagens, como foi visto, centra-se, após selecção e/ou prioritização, na validação das configurações de modo isolado. Quer-se com isto dizer que não é uma validação contextual, como pretendemos neste trabalho de mestrado.

A abordagem pretendida nesta dissertação encontra-se mais direccionada para a realização de **testes de aceitação** (*Acceptance Tests*) sendo portanto uma abordagem *black-box* ou seja, com interesse no carácter funcional da aplicação, e em que as configurações são validadas num contexto de execução real.

Capítulo 5

Descrição de Configurações

"With great power comes great responsibility."

Spider-Man - Comic SuperHero

A utilização de configurações (ver Definição 1 no Capítulo 1) no contexto de sistemas de software, de forma a possibilitar que estes sistemas sejam adaptados às necessidades específicas de cada utilizador, é algo que conta já com algum histórico [NN09].

Dotar software de tal capacidade é uma mais valia para os seus utilizadores, no entanto é necessário entender que a liberdade de configurar uma aplicação pode trazer complicações se estas não forem correctamente utilizadas — tal como o Spider-Man disse "com grande poder vêm grandes responsabilidades".

O que se pretende com este capítulo é:

- mostrar algumas das abordagens existentes no que diz respeito a construir software configurável;
- descrever a forma de especificar as configurações;
- introduzir alguns sistemas que podem ser considerados tipicamente configuráveis;
- mostrar a abordagem seguida no caso de estudo desta dissertação para lhe incutir o carácter configurável.

5.1 Abordagens ao desenvolvimento de Software Configurável

De entre as abordagens tidas ao longo dos tempos no que diz respeito ao desenvolvimento de sistemas configuráveis podemos referenciar as seguintes:

- **Hierarchical Design** — a noção de sistemas configuráveis não é recente. Já em 1968, Edsger W. Dijkstra refere nas conclusões do artigo [Dij68], que um sistema com uma arquitectura hierarquica a vários níveis poderá ser facilmente adaptado através de expansões de configurações — a organização do sistema em vários níveis hierárquicos em que cada nível utiliza apenas funcionalidades dos níveis abaixo permite que seja construída uma abstracção em cada nível escondendo os detalhes dos níveis inferiores suportando assim expansão das funcionalidades.

- **Program Families** — um pouco mais tarde, em 1976, é David L. Parnas que discute metodologias de programação para o desenvolvimento de "*famílias de programas*", ou programas que possuem variações entre si mas que possuem um conjunto de propriedades comuns — um pouco como os sistemas configuráveis. No artigo [Par76] são abordados alguns métodos para a construcção deste tipo de sistemas, nomeadamente o método:

  - Stepwise Refinement — introduzido por Dijkstra, consiste no refinamento gradual de uma aplicação, deixando para estágios mais tardios a implementação de certos operadores e operandos. Permitindo de certa forma que tal abstracção de operadores e operandos permita uma variedade de implementações.
  - Module Specification — consiste na divisão dos programas em módulos e, em "esconder" em módulos informação que não é comum aos restantes programadas da família em que se insere.

- **Object Oriented Design** — mais recentemente, o aparecimento das linguagens orientadas aos objectos como o Java, introduziram abordagens na concepção de sistemas que favorecem de certo modo a configurabilidade/-variabilidade das mesmas. Para tal contribui o facto de ser uma linguagem dotada de artifícios como: herança, abstracção, polimorfismo e encapsulamento [Sch07, NK02], que permitem modelar e construir uma arquitectura extensível, parametrizável e modular. Ou seja, que facilitam a construcção de uma aplicação que suporte variabilidade, que seja configurável.

Para tal, existem alguns mecanismos que se podem seguir de modo a ajudar a concretizar tais objectivos, como sejam a utilização de:

- Herança como mecanismo de adaptação: dada uma classe e sua implementação, uma sub-classe pode introduzir alterações ao comportamento por omissão (o da superclasse) consoante necessário;
- Plugins por forma a introduzir variabilidade;
- Estruturas compostas;
- Encapsulamento sobre o que varia no negócio.

Muitas das boas práticas no desenho de uma arquitectura de software referidas, são parte integrante dos Padrões de Desenho de Software — ou **Design**

Patterns, como são mais conhecidas — e que fornecem *boas* soluções para problemas recorrentes [AMC+03, GHJ+93, GHJV95], ajudando portanto na construcção de software configurável.

- ***Software Product Lines*** (SPL) — "Linhas de Producção de Software" são uma técnica para a concepção de sistemas de software semelhantes, que partilham um conjunto de características entre si e que são desenvolvidos a partir de um ponto em comum [SM10, Kru00, Don08]. Um pouco como é aplicado nas linhas de produção fisicas, o que é pretendido é que o software seja *montado* com artefactos reutilizáveis contribuindo assim para que os vários produtos gerados nesta mesma linha de producção (SPL) sejam consistentes [Wit09].

  A principal diferença encontra-se na mudança de estratégia: do desenvolvimento tradicional de um sistema específico e individual passamos para o desenvolvimento de uma área de negócio, abrindo assim espaço para o suporte a uma variedade de produtos de software [LSR07].

  Este tipo de abordagem pretende que, um sistema de software "modelo" com características gerais e comuns, se possa ajustar às necessidades especificas de um determinado cliente, através de configurações sobre a SPL, possibilitando dessa forma a reutilização desse mesmo sistema em diferentes cenários [SS09, LYL09].

  Desta forma, um sistema configurável pode ser visto como uma SPL, em que cada instância resultante da aplicação de diferentes configurações, é um produto diferente gerado com origem nessa mesma SPL.

  É importante notar a existência de diferentes estratégias no que diz respeito às técnicas de variabilidade (ver Figura 5.1) que podem ser utilizadas numa SPL: *adaptation* — em que ajustamos ou escolhemos as configurações quer seja através de um ficheiro ou parametrização em tempo de execução; *replacement* — "completamos" a nossa aplicação com um de vários componentes que foram implementados especificamente para aquela arquitectura; *extension* — em que adicionamos novos componentes à aplicação como plugins. No decorrer deste trabalho sempre que falamos de **configurações** estamos de facto a referirmo-nos à aplicação de qualquer uma das técnicas acima referidas.

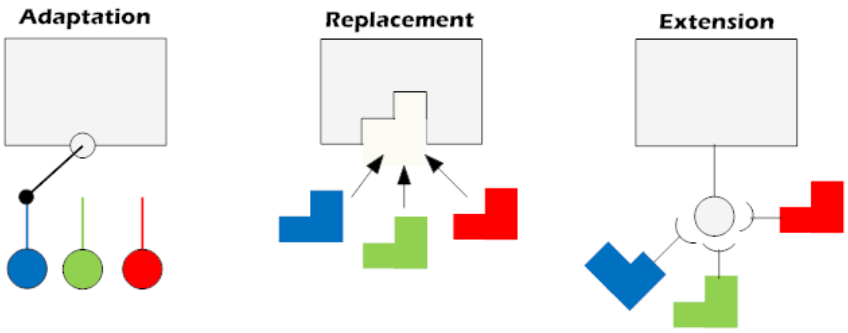

Figura 5.1: Técnicas para realizar Variabilidade numa Arquitectura [LSR07]

- ***Business Rules Management System*** (BRMS) — são sistemas de gestão de **regras de negócio** como sejam requisitos e regras condicionais, que expoẽm a lógica de negócio sob a forma de um conjunto de regras permitindo gerir e adaptar o comportamento e lógica de decisões de um sistema aplicacional de modo facilitado [HCWC09a, DMM07, TR07], que em muito se deve à capacidade de separar a lógica de negócio do código da aplicação [HCWC09b].

  Entendendo-se por **Regra de Negócio** algo que expressa uma política de negócio ou restrição sobre um determinado domínio [DMM07][1].

  Um BRMS não é uma ferramenta, mas sim um conjunto de ferramentas incluindo: motor de regras, repositório dessas mesmas regras, interface com o utilizador entre outras; e que pretendem flexibilizar a gestão das regras de negócio aplicadas num sistemas aplicacional principalmente quando tais regras tendem a ser voláteis [HCWC09a, HCWC09b].

  Existem inúmeros BRMS *Open Source* disponíveis como: **Drools** [Nea10] ou JbossRules (Red Hat), **Jess** [Fri10] (Sandia National Laboratories), **OPSJ** [PST10] (Production Systems Technologies) entre outras.

  Uma aplicação altamente configurável pode ser vista como um BRMS na medida em que as configurações sejam vistas como regras de negócio que definem o comportamento da aplicação em causa.

Cada uma das abordagens implementa diferentes mecanismos, podendo utilizar diferentes representações para as configurações. Seja qual for a abordagem escolhida, gerir e manter as configurações é essencialmente um processo *ad-hoc*. Consoante o sistema, as configurações poderão ser lidas, gravadas e processadas de forma diferente.

5.2 Especificar Configurações

Existem inúmeras formas possíveis para se especificarem configurações. A arquitectura utilizada para o efeito pode variar consoante o tipo de aplicação em que se inserem (e.g.: se é uma aplicação *web* ou uma aplicação para *desktop*, se é uma aplicação distribuída ou centralizada), consoante a linguagem de desenvolvimento utilizada (e.g.: linguagem orientada a objectos como Java ou é uma linguagem imperativa como o C), também consoante os objectivos pretendidos para essas configurações (e.g.: se pretendemos carregar essas configurações uma única vez ao arrancar do programa, ou se queremos que seja possível a alteração dos valores em tempo de execução), entre outros factores.

As formas mais comuns são a utilização de:

- ***flags* 'via' linha de comandos** — são basicamente opções (ou *argumentos*) que podemos enviar para uma aplicação ao invocá-la, podendo ser tantos quantos os que sejam necessários, dependendo das funcionalidades da aplicação. A

[1] Exemplo: "Se um território é círculo eleitoral, então deverá haver atribuição de mandatos nesse território".

sua utilização não é aplicável em todos os contextos, mas linguagens como o C, C++ ou Java permitem interpretar os argumentos passados por linha de comandos. Um exemplo de uma aplicação configurável através de *flags* é o comando **grep** para sistemas Unix.

- **ficheiros de preferências** — os quais incluem as configurações a serem carregadas pela aplicação. É usual a utilização de ficheiros *Extensible Markup Language* (`XML`) [W3C], ficheiros *Comma-Separated Values* (`CSV`) ou ficheiros Properties.

  Se as configurações que pretendemos carregar possuem uma estrutura mais complexa, o ideal é utilizar ficheiros `XML` pelo facto de facilitar a representação da informação de qualquer tipo. Por outro lado se o que pretendemos é representar listas de valores, então é possível que os ficheiros `CSV` sejam suficientes permitindo para de forma simples representar essa informação. Se as configurações que pretendemos importar são uma lista de pares do tipo chave/valor, então o ideal é utilizar ficheiros Properties que seguem exactamente essa estrutura. Se pretendermos, podemos obviamente definir a estrutura do nosso ficheiro da forma que nos for mais conveniente. Um exemplo de um sistema configurável através da utilização de ficheiros é o servidor aplicacional JBoss. O caso de estudo desta dissertação — Plataforma Eleitoral — utiliza esta técnica como principal forma de especificação das configurações.

- **variáveis de ambiente** — são um conjunto de pares chave/valor definidos de forma dinâmica num sistema operativo e que pretendem maioritariamente guardar valores que dizem respeito a propriedades globais relacionadas com o sistema operativo em execução, necessárias para execução de outras aplicações, como sejam a localização de uma determinada aplicação no sistema. Um exemplo de um sistema configurável que utilize variáveis de ambiente é a Java Virtual Machine. O caso de estudo utiliza uma única variável de ambiente, que tem como objectivo indicar a directoria do sistema onde os ficheiros de configuração se encontram.

- **painéis de preferências** — é uma forma mais "simpática" de expor a lista de configurações possíveis aos utilizadores através da introdução ou edição dos dados de forma controlada num formulário de um determinada aplicação. Podendo incluir validação automática dos valores introduzidos e/ou comportamento específico. O formulário serve maioritariamente como ambiente de controlo para preenchimento dessas configurações, as quais são normalmente guardadas posteriormente em um ficheiro ou base de dados. Um exemplo de um sistema configurável que utiliza painéis de preferências para edição das configurações é o *browser* Firefox (ver a página `about:config`).

- **tabelas de uma base de dados** — é uma forma que, directa ou indirectamente, é utilizada para persistir configurações de uma aplicação de forma a que estas não se percam. É normalmente um recurso utilizado por uma grande parte dos sistemas para, de forma indirecta, após interpretar os formulários ou fazer a leitura dos ficheiros, guardar as configurações do sistema. É exemplo da utilização desta abordagem o caso de estudo desta dissertação.

- **variáveis em memória** — sendo menos utilizada devido ao carácter volátil das variáveis em memória, é por vezes usada juntamente com outras formas (como sejam uma base de dados), por ser de mais rápido acesso. Surge algumas vezes associada ao *design pattern*: Singleton, o qual tem como objectivo garantir a existência de apenas uma instância de uma classe especifica — neste caso a classe que contenha as configurações. O caso de estudo desta dissertação utiliza também esta abordagem.

5.3 Exemplos de Sistemas Configuráveis

Após o que já foi dito sobre sistemas configuráveis, é importante introduzir alguns sistemas que são altamente configuráveis, e identificar algumas das suas aplicações após terem sido configurados:

- **Plataforma Eleitoral** — o caso de estudo deste trabalho de mestrado é um exemplo de um sistema configurável, o qual utiliza uma variável de ambiente que aponta para vários ficheiros de preferências utilizados para especificar as configurações. As configurações contidas nestes ficheiros são posteriormente guardadas utilizando-se tabelas de uma Base de Dados e variáveis em memória. O sistema foi já utilizado no âmbito de algumas das eleições realizadas em Portugal, nomeadamente nas:

  - Europeias de 2009 [dAID10c];
  - Legislativas de 2009 [dAID10d];
  - Autárquicas de 2009 [dAID10b].

  Para a utilização da Plataforma Eleitoral em cada uma das diferentes eleições referidas foram necessárias apenas ajustar as configurações com as regras de negócio específicas de cada um dos Actos Eleitorais.

- ***Work Orders Web*** (`WOW`) — é uma ferramenta de gestão de "ordens de trabalho" (*aka Work Orders*) — `Trouble Ticket System` — que permite agilizar o processo das *work orders* desde o momento do pedido até à sua completude [CSW10].

  É um sistema altamente configurável na medida em que permite o total controlo do fluxo de negócio e dos processos associados de forma distribuida. Para além disso é configurável a sua integração com vários serviços internos e externos: Web, e-mail ou SMS's.

  Todas as configurações da aplicação são introduzidas/alteradas através de um painél de preferências — o "Menu de Administração". Uma vez nesse ecrã o utilizador (Administrador) poderá configurar os `Tipos de Pedidos`, as `Prioridades`, os `Mapas de Estados`, as `Transições` aplicadas em cada uma das `Acções` possíveis de aplicar nos diferentes estados; os `Formulários` e os `Campos` a preencher; os `Relatórios` e as `Listagens`; bem como os `Utilizadores` e os `Grupos` a que pertencem, entre muitas outras características.

Este sistema é utilizado actualmente por vários clientes/parceiros da Critical Software como:

- Vodafone;
- grupo Portucel Soporcel;
- Banco Caixa Geral de Depósitos.

A arquitectura utilizada nas aplicações tem sofrido bastantes alterações de modo a adaptar-se cada vez melhor e facilitar a construcção e utilização de sistemas configuráveis.

É importante entender neste contexto que o *design* das configurações pode influênciar a relação das configurações entre si, e consequentemente influenciar a dificuldade com que se pode fazer a validação e teste das mesmas [NN09].

5.4 Caso de Estudo

Com tantas possibilidades para abordar a construcção de sistemas de software configuráveis, interessa perceber qual a abordagem concreta que foi seguida no caso de estudo desta dissertação de modo a construir uma aplicação de elevada configurabilidade.

Ora, o primeiro desafio que surgiu na construcção da aplicação foi o de **decidir o que seriam (ou não) configurações**. Ora o que se deve tornar parametrizável depende muito do nível de flexibilidade com que se pretende dotar a aplicação. Esta tarefa implica elevados níveis de conhecimento das regras de negócio do sistema a desenvolver. Para o caso da Plataforma Eleitoral, foram estudados os diferentes actos eleitorais e levantados todos e quaisquer pontos de variabilidade para que o sistema suportasse tal mudança de contexto.

No que diz respeito à **forma utilizada para suportar tais alterações de contexto** da aplicação, como já foi referido na secção anterior, recorreu-se à utilização de ficheiros para especificar essas mesmas regras — nomeadamente ficheiros `XML` e `CSV` (tabela 5.1). Dos muitos ficheiros necessários para configurar a Plataforma Eleitoral, existem ficheiros opcionais e ficheiros obrigatórios; existem relações de dependência e ordem entre os vários ficheiros; e ficheiros com uma sintaxe própria.

Os ficheiros `CSV` são utilizados quando pretendemos definir dados sem grande estrutura entre si, tais como listagens simples, à base de sequências de valores separados por vírgulas(‘,’) (Listagem 5.1). Por outro lado se as configurações possuem uma complexidade elevada, com elementos e atributos relacionados entre si de forma hierárquica (não linear), então os ficheiros `XML` são os mais indicados (Listagem 5.2).

Tabela 5.1: Ficheiros de Configuração da Plataforma Eleitoral.

Nome do Ficheiro	Descrição
configuration.xml	Ficheiro inicial e com configurações mais genéricas, como horas de abertura e fecho do acto eleitoral, algoritmos de cálculo a serem carregados, configurações para as afluências, relatórios e opções de exportação de resultados.
territory_hierarchy.csv	Configuração da hierarquia territorial a usar.
electoral_act.xml	Listagem das eleições a decorrer no acto eleitoral.
timezones.xml	Diferenças horárias existentes entre os territórios.
affluences.csv	Territórios alvo de afluências.
territory.csv	Definição dos territórios alvo do acto eleitoral.
reference_subscribers.csv	Listagem do número de inscritos de referência por território.
constituencies.csv	Listagem dos círculos eleitorais definidos.
candidates_list.csv	Candidatos de cada partido para cada círculo eleitoral.
options.csv	Opções de voto de cada um dos círculos eleitorais.
political_parties.csv	Listagem dos partidos políticos intervenientes no acto eleitoral.
users_details.csv	Listagem de logins/passwords de acesso à aplicação.
users.csv	Utilizadores a serem gerados para posterior acesso à aplicação.
supporters.csv	Listagem com apoiantes de cada opção do acto eleitoral.
old_results_data.csv	Listagem com resultados de eleições anteriores.

Listagem 5.1: Excerto de ficheiro CSV

```
1 Region,Territory_ID,Description,[Parent_Region],[Parent_ID]
2 GLOBAL,990000,Resultados Globais,,
3 FOREIGN,800000,Estrangeiro,GLOBAL,990000
4 LOCAL,500000,Portugal,GLOBAL,990000
5 LOCAL,010000,Aveiro,LOCAL,500000
6 LOCAL,020000,Beja,LOCAL,500000
7 LOCAL,030000,Braga,LOCAL,500000
```

Listagem 5.2: Excerto de ficheiro XML

```
1 <configuration>
2   <affluence insertion_time="01:00" definitive_close_time="true">
3     <hours>
4       <hour>2009-06-07T12:00:00</hour>
5       <hour>2009-06-07T16:00:00</hour>
6     </hours>
7   </affluence>
8 </configuration>
```

No que diz respeito às responsabilidades de cada actor envolvido no sistema, salienta-se o facto de:

- ser da responsabilidade do cliente o correcto preenchimento dos vários ficheiros de preferências com as configurações a serem carregadas pela aplicação; Sendo,

no entanto, aceitável que seja o administrador a preencher os ficheiros de preferências desde que o cliente forneça todos os dados necessários;

- ser da responsabilidade do administrador da aplicação garantir que tais ficheiros são correctamente usados pelo sistema.

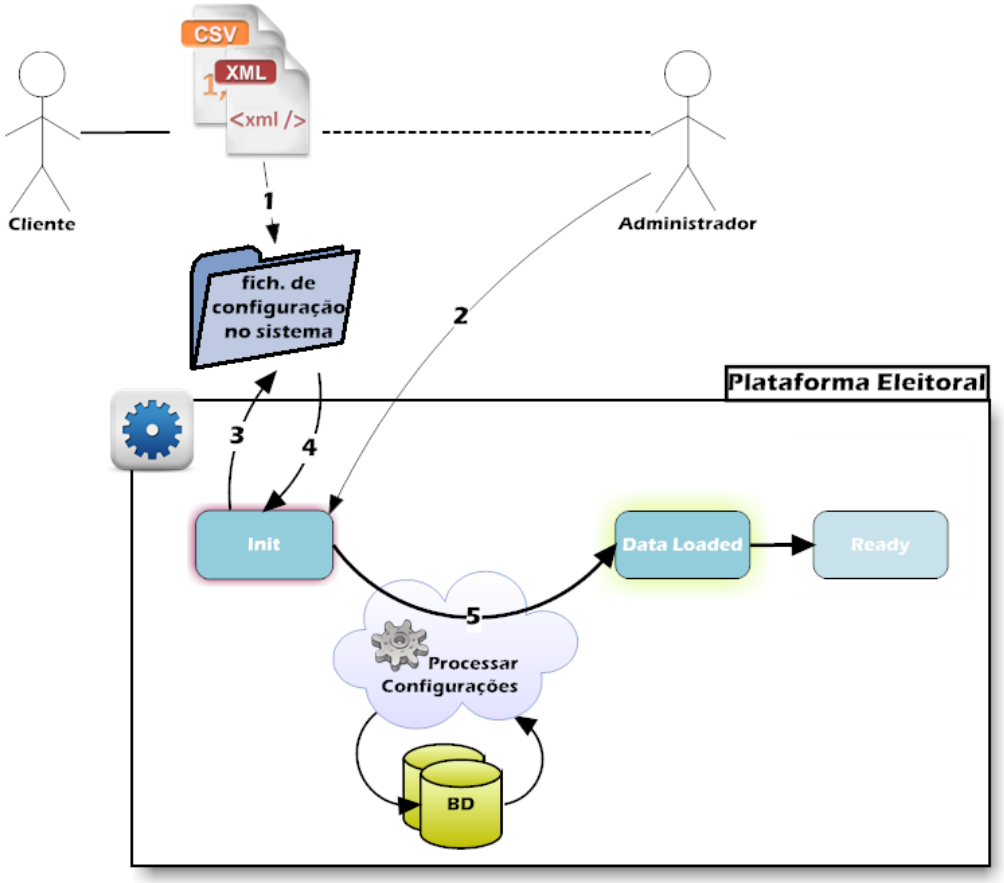

Figura 5.2: Carregamento das Configurações para a Plataforma Eleitoral

Na Figura 5.2, é possível verificar-se como é que o processo de configuração da Plataforma Eleitoral funciona, bem como quem é responsável por fazer que tarefas.

Descrevemos em seguida, com um pouco mais de detalhe, as diferentes etapas representadas:

1. O cliente, em conjunto com um administrador do sistema preenche os ficheiros de preferências necessários pela aplicação, colocando-os na pasta que será posteriormente utilizada pela Plataforma;

2. O administrador dá ordem para que carregamento dos dados seja efectuado de modo a configurar a Plataforma;

3. O sistema, através de uma variável de ambiente, reconhece qual a directoria do sistema operativo onde os vários ficheiros de preferências necessários se encontram;

4. O sistema inicia o processo de carregamento dos ficheiros;

5. O sistema processa os vários ficheiros existentes na directoria e gera as configurações que serão posteriormente utilizadas durante a utilização da Plataforma. Nesta etapa existem dados de configurações que são gravados na Bases de Dados da Plataforma Eleitoral.

Os acessos às Bases de Dados são sempre efectuados através do sistema, pelo que ninguém deverá efectuar alterações sobre a mesma de modo a não causar inconsistências de dados para com as regras de negócio.

Finda as etapas descritas, o sistema encontra-se configurado e (praticamente) pronto para utilização.

Para uma melhor percepção do modo do funcionamento global do caso de estudo desta dissertação, apresenta-se na Figura 5.3 o **diagrama de estados** da Plataforma Eleitoral.

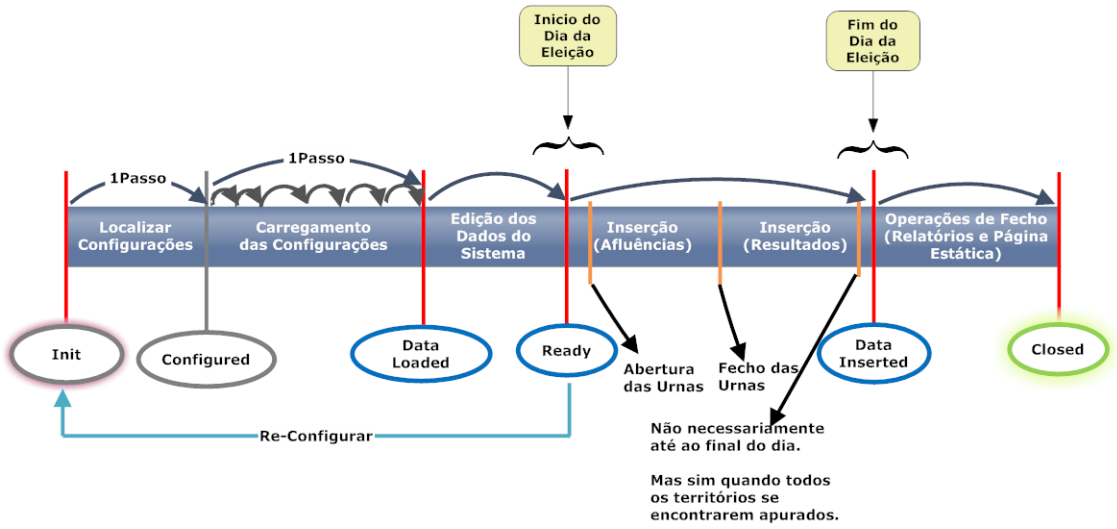

Figura 5.3: Diagrama de Estados da Plataforma Eleitoral

A execução da Plataforma Eleitoral inicia-se com a configuração da aplicação e carregamento de todas as configurações necessárias (transição de *Init* para *Data Loaded*); em seguida há a possibilidade de serem feitas pequenas correcções a esses mesmos dados e marca-se a aplicação como configurada e pronta a ser usada (transição de *Data Loaded* para *Ready*); neste estado e, enquanto não existem quaisquer dados introduzidos na aplicação há sempre a possibilidade de re-configurar a aplicação com novas configurações (transição de *Ready* para *Init*).

A partir do momento em que a aplicação se encontra no estado *Ready*, a aplicação terá em consideração as regras de negócio configuradas. É durante esta fase que ocorrem as inserções de afluências e de resultados das votações, que são o `core` do negócio. De notar que a partir desta fase as configurações foram completamente carregadas e os ficheiros não são mais utilizados. Assim que todos os territórios tenham os dados inseridos é possível fazer a transição para o estado *Data Inserted*.

Nesta última fase são possíveis a geração de relatórios e da página estática da eleição; Feito isto transitamos para o último estado da nossa aplicação, o estado *Closed*, onde termina a execução da mesma.

Capítulo 6

Classificação de Erros no Software Configurável

"Testing can only demonstrate the presence of defects, never their absence."

Dijkstra

Antes de abordarmos o tema principal deste capítulo é importante relembrar o objectivo deste trabalho de mestrado – o que se pretende é validar que as configurações utilizadas no contexto de uma aplicação definem os requisitos ou regras de negócio que o cliente deseja.

Com o desenvolvimento do trabalho desta tese pretende-se fornecer ao cliente uma forma de este poder definir um conjunto de regras que pretende ver validadas e que irão ajudar a garantir a ausência de erros de negócio introduzidos por erros nas configurações.

É com base nessas regras de negócio que construímos as regras que se pretendem validar, às quais chamamos de `Regras de Validação(RV)`. Podemos encontrar no Anexo A um conjunto exemplo dessas regras para o caso de estudo deste trabalho de mestrado.

O objectivo das `RV` é o de definir restrições sobre o negócio, que permitam validar se a aplicação cumpre a lógica pretendida pelo cliente. No caso de alguma destas regras não ser satisfeita, significa que as configurações que foram usadas contêm erros.

O objectivo é definir o maior número de `RV` de modo a podermos aumentar a confiança nas configurações utilizadas na aplicação, e diminuir possíveis erros lógicos durante a execução.

É importante notar que, tal como Dijkstra disse, os testes numa aplicação permitem apenas garantir a existência de erros e não a sua ausência. Também não conseguimos validar que as configurações se encontram 100% correctas; no entanto podemos garantir que não ocorrem determinados erros face às `RV` definidas.

6.1 Terminologia

Nesta secção pretendemos definir o conjunto de termos que iremos utilizar nas secções seguintes, e que servirão para que não surjam subjectividades nas definições textuais que serão apresentadas.

Assim sendo, temos:

- **atributo**: representa uma característica ou propriedade de uma entidade (ou objecto);

  Por exemplo *name* é um atributo da entidade *Territory*.

- **tuplo**: corresponde a um número pré-definido de atributos que tem como finalidade descrever um objecto (de negócio ou mesmo do mundo real) ou a informação desse objecto;

  Por exemplo os seguintes atributos: *region*, *level* e *description* definem um tuplo para o objecto *TerritoryHierarchy* (Hierarquia de Territórios).

- **relação**: corresponde à ligação/conexão que é possivel estabelecer entre dois ou mais tuplos numa entidade, ou entre entidades diferentes. Podem ser relações de dependência, de integridade, associação ou composição;

  Por exemplo: cada tuplo da entidade *Territory* está associado a um tuplo da entidade *TerritoryHierarchy*.

- **entidade**: corresponde a um conjunto de tuplos que possuem os mesmos atributos; os quais podem estabelecer relações entre si na mesma entidade, ou entidades diferentes.

  Por exemplo: *Territory*, *TerritoryHierarchy* ou *Affluences* são exemplos de entidades.

6.2 Configurações

6.2.1 O que podemos fazer com elas?

Já foram referidas neste documento (capítulo 5) diferentes formas de especificar configurações. O que não foi referido, foi o que é possível fazer com as configurações:

- definir valores de atributos;
- estabelecer relações;
- instanciar tuplos e as suas entidades;

6.2.2 Problemas que se podem introduzir?

É através das configurações que são "injectados" erros nas regras de negócio da aplicação. Já identificamos o que podemos fazer com as configurações, resta-nos agora enunciar o tipo de problemas que nos permitem introduzir na aplicação.

Assim sendo, e tomando cada um dos termos definidos como referência, iremos expor os problemas que ocorrem por ordem de complexidade, ou seja, partindo do termo mais elementar – o atributo – passando depois pelo tuplo e relação e terminando no mais complexo – a entidade [Oli08]:

A. Atributo

   1. Valor em falta;
   2. Violação de sintaxe;
   3. Violação de domínio;
   4. Erro ortográfico;
   5. Violação de unicidade;

B. Tuplo

   1 Violação de restrição de integridade – entre atributos de um mesmo tuplo;

C. Relação

   1. Existência de sinónimos:
      - entre o mesmo atributo de diferentes tuplos;
      - entre atributos de diferentes entidades;
   2. Existência de homónimos
   3. Violação de restrição de integridade:
      - entre o mesmo atributo de diferentes tuplos;
      - entre diferentes atributos de diferentes tuplos;
      - entre atributos de diferentes entidades;
   4. Violação de dependência funcional;
   5. Circularidade entre tuplos num auto-relacionamento;
   6. Tuplos duplicados:
      - na mesma entidade;
      - em entidades diferentes;
   7. Violação de integridade referencial;
   8. Violação da cardinalidade entre relações;

D. Entidade

   1. Heterogeneidade de sintaxes – de um atributo em diferentes entidades;
   2. Heterogeneidade de unidades de medida – de um atributo em diferentes entidades;

6.3 Tipos de Erros

As regras `RV` que definimos pretendem evitar que regras de negócio sejam mal definidas nas configurações. Nesta secção pretendemos apresentar erros que tipicamente acontecem na definição das regras de negócio aquando da construção das configurações para aplicações altamente configuráveis.

Os problemas causados pelas más configurações ao nível dos atributos, tuplos, relações e entidades (tipificados em 6.2.2) vão induzir erros nas regras de negócio, os quais se agrupam em seguida e, para os quais se apresentam alguns exemplos de `RV`, que têm como objectivo evitar erros desse mesmo tipo. Tem-se assim os seguintes tipos de erros:

- **Completude de Entidades** [Problemas: C7-C8] — quando a cardinalidade de uma relação de pertença não é respeitada.[1]

  Como exemplos de `RV` para este tipo de erros temos, por exemplo:

  - `Todos os territórios têm de ter pelo menos 1 opção de voto.`
  - `Todos os territórios que não são de inserção devem ter a si associados pelo menos 1 território no nível inferior (ou seja um "filho").`

- **Coerência de Dados** [Problemas: C1-C2, C4, D1-D2] — quando não existe conformidade entre valores de dois ou mais atributos de diferentes tuplos ou entidades.

  Como exemplos de `RV` podemos ter:

  - `No contexto da comparação de resultados actuais com os anteriores, um território actual (identificado pelo` *`id`* `e` *`região`*`), caso exista na lista dos territórios anteriores, deverá corresponder ao mesmo território.` Ou seja, para o mesmo 'id' e 'região', a descrição deverá ser a mesma em ambos os territórios.

- **Integridade de Dados** [Problemas: A1-A5] — quando determinado atributo de uma entidade possui valores não permitidos (ou inválidos).

  Note-se: Este tipo de erros são maioritariamente detectados directamente na inserção elementar de cada registo dos ficheiros de configuração. No entanto, algumas destas regras não estarão integradas directamente na aplicação para que, como já referido anteriormente, mantermos o carácter configurável da aplicação.

[1] Podendo indicar que: uma entidade A pode (ou não) estar associada a mais que uma entidade B; uma entidade A tem (ou não) de estar associada a pelo menos uma entidade B;

Como exemplos de `RV` poderiamos ter:

- `Territórios com nomes vazios (“ ”).`
- `Candidatos com nomes vazios (“ ”).`
- `Territórios com número de inscritos de referência igual a 0 (zero).`

- **Integridade Relacional** [Problemas: B1, C3, C5-C6] — quando há (valores de) atributos ou tuplos que violam restrições de negócio, quando comparados entre si, quer seja:
  - dentro do mesmo tuplo (apenas aplicável aos atributos);
  - entre tuplos numa mesma entidade;
  - entre entidades;

  Note-se: Maioritariamente, as validações que pretendemos fazer serão deste tipo de erros, pois dizem respeito a coisas que apenas se podem validar por cruzamento de dados (originalmente provenientes de dois ou mais ficheiros de configuração diferentes).

  Como exemplos de `RV` podemos ter:

  - `Nenhum território poderá ter como território no nível inferior (ou seja o “filho”) o mesmo território associado ao nível superior (ou seja o “pai”).` Ou seja, não podem existir territórios com ciclos.
  - `Todos os territórios devem ser alcançáveis.` Ou seja, partindo de cada um dos territórios definidos na hierarquia, devemos ser capazes de navegar até ao território *raiz*.
  - `O número de mandatos a atribuir num círculo eleitoral encontra-se estritamente relacionado com o número de inscritos de referência do(s) território(s) que lhe estão associados. Assim sendo, nenhum círculo eleitoral poderá atribuir mais mandatos do que um outro círculo eleitoral que tenha mais inscritos de referência do que o primeiro.`
  - `O número de candidatos efectivos de cada partido concorrente num círculo eleitoral de uma dada eleição deverá ser igual ao número de mandatos a atribuir nesse mesmo círculo eleitoral.`

Os grupos acima definidos pretendem de alguma forma **generalizar os problemas** que são comuns. Para além disso estes grupos, por representarem problemas semelhantes, agregam desde já as diferentes regras por “tipo de validação”.

6.4 Classes de Severidade dos Erros

A classificação por severidade dos diferentes tipos de erros possíveis depende em muito das características do sistema a que nos estamos a referir. As próprias definições das diferentes classes ou categorias dos erros dependem elas próprias do tipo de sistema a que se referem — um erro interpretado como sendo grave num sistema pode ser considerado um erro menor num noutro sistema diferente.

Nesta secção pretendemos apresentar quais as várias classes de severidade de erros que pretendemos associar às diferentes `RV` — dado que cada regra de validação pretende de certa forma evitar um determinado erro em específico.

O objectivo é ser possível, posteriormente, tomar diferentes acções consoante a severidade do erro que a `RV` pretende evitar.

Assim sendo, teremos as seguintes classes:

- **Catastrófico** — situação irreversível e que coloca as funcionalidades principais da aplicação em risco;
- **Severo/Grave** — Coloca algumas funcionalidades da aplicação em risco, não existindo *workaround*[2];
- **Maior/Moderado** — Coloca algumas funcionalidades da aplicação em risco, mas existem *workarounds*;
- **Menor** — Não afecta a execução da aplicação, mas poderão ser visíveis ao utilizador final;
- **Trivial** — Não afecta a execução da aplicação, sendo erros transparentes para o utilizador final.

6.5 Estratégia de Validação das Regras (`RV`) definidas

Após definição do conjunto de regras de validação (`RV`), a estratégia a seguir é que tais regras sejam, verificadas individualmente sobre a aplicação já configurada de modo a que possamos avaliar se essa mesma regra está (ou não) a ser cumprida.

Se todas as regras `RV` que o cliente definir passarem no teste de validação, então poderemos dar a aplicação como validada para as regras que o cliente definira. Tal como já referido anteriormente, é de todo o interesse que este conjunto de regras se vá expandindo e seja o mais completo possível para que possamos maximizar a confiança nas configurações em utilização pela aplicação.

A estratégia para validar cada uma das `RV` começará pela representação dessa mesma regra numa determinada linguagem (o mais simples possível), havendo posteriormente um mecanismo para interpretar e executar a regra na aplicação; esta execução deverá retornar um resultado (eventualmente um *booleano*) indicando se a regra está (ou não) a ser cumprida.

Este assunto será posteriormente explorado nesta dissertação, no entanto, é de esperar que as `RV` que correspondem ao mesmo tipo de erro tenham uma abordagem de resolução semelhante.

[2] *workaround*: caminho ou método alternativo para atingir determinado objectivo ou tarefa, utilizado de forma temporária para contornar ou superar um problema. (adaptado de [Dic10])

Capítulo 7

Arquitectura da Solução - `Validador`

"Everything you can imagine is real."

Pablo Picasso

Este capítulo vem apresentar a arquitectura da aplicação de validação do software altamente configurável; primeiro numa visão geral tendo em vista as necessidades da mesma como solução ao problema descrito nesta dissertação; e depois numa visão mais refinada identificando os vários componentes que a integram.

7.1 Visão Geral do Sistema

O Validador pode ser dividido em três fases: a fase de criação de regras: onde um utilizador com conhecimento do negócio da aplicação a validar introduz as regras num repositório; a fase de validação: onde as regras previamente introduzidas serão validadas no contexto da aplicação alvo; e a fase de apresentação de resultados: onde serão apresentados os resultados da validação de cada uma das regras executadas.

O intuito deste projecto é fornecer uma aplicação que permita validar, com base num conjunto de premissas ou regras `RV`, que uma determinada aplicação altamente configurável tem o seu negócio correctamente definido.

A Figura 7.1 mostra qual o ambiente envolvente e as interacções do `Validador` com os sistemas, ou entidades externas.

Em seguida descreveremos o comportamento do sistema — `Validador` — identificando as interacções entre os *actores* do sistema e as várias funcionalidades por eles desempenhadas.

Tais funcionalidades foram traduzidos num conjunto de casos de uso (Figura 7.2) que documentam os requisitos funcionais de alto nível. Um caso de uso descreve a forma como os utilizadores do sistema interagem com ele para efectuar as suas tarefas.

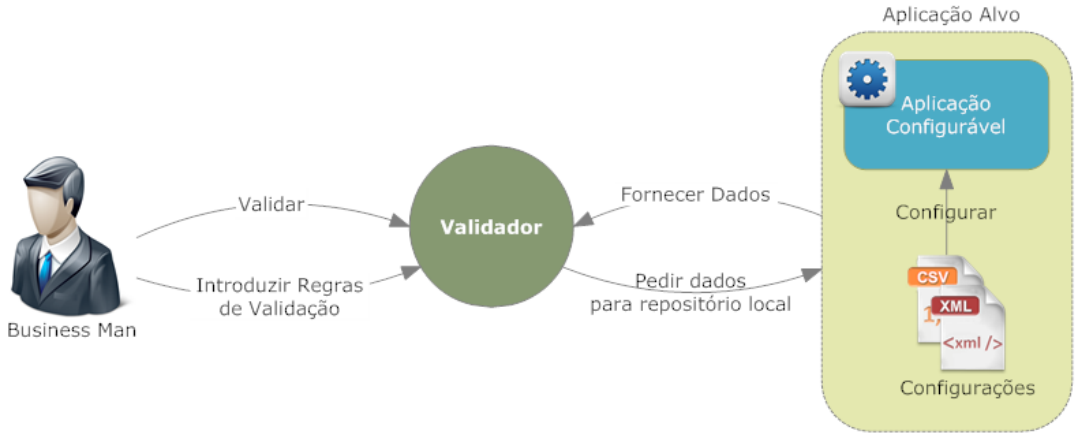

Figura 7.1: Visão Geral do Sistema

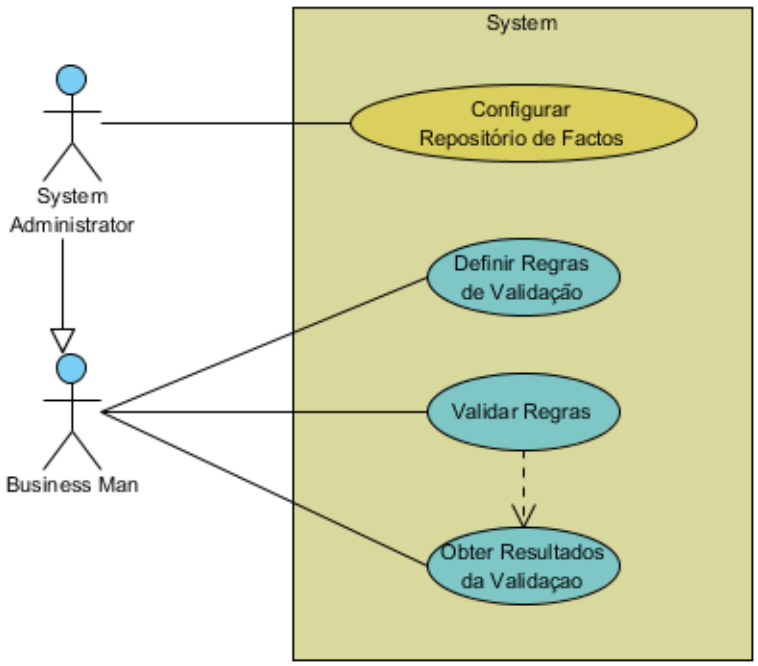

Figura 7.2: Diagrama de Use-Cases do Validador

7.2 Descrição do Processo do Validador

Um `Validador` é definido por um conjunto de regras a validar (regras `RV`); um conjunto de dados ou informações que se pretendem ver validadas (factos); e um processo de validação que valida cada uma das regras `RV` para o conjunto de factos existentes.

Em seguida apresenta-se uma descrição mais detalhada do comportamento que o `Validador` terá durante a sua execução.

O processo do sistema `Validador` inicia-se com a configuração do repositório de factos para que seja possível aceder à aplicação altamente configurável alvo de validação, a qual fornecerá todos os dados necessários para que possa ser validada.

Feito isto, um utilizador *expert* na área de negócio da aplicação alvo (aka: *business analyst*) define o conjunto de regras `RV` que se pretendem executar e validar.

Após a introdução dessas regras, o utilizador (*business analyst*) poderá ordenar a execução do processo de validação de cada uma regras que introduziu previamente.

Como resultado desta última acção, obteremos uma listagem de todas as regras juntamente com a informação que indica se a regra se encontra empregue, ou não, na aplicação alvo.

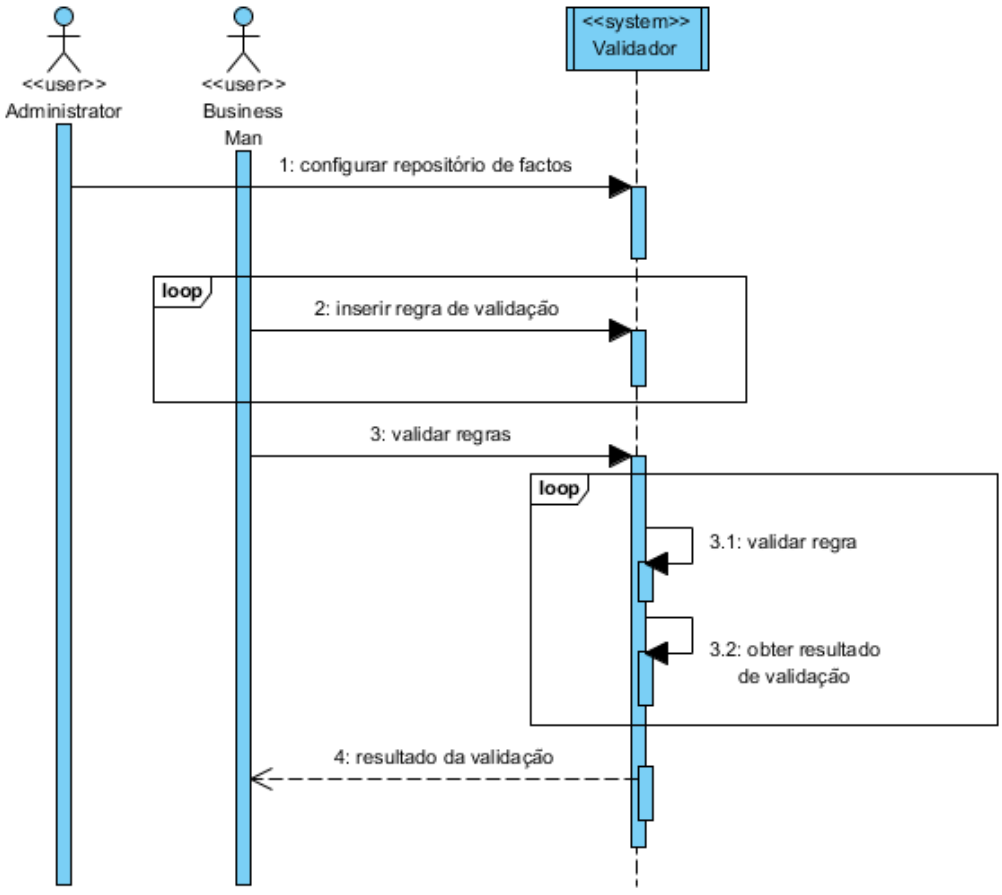

Figura 7.3: Modelo do Processo do Validador

Na Figura 7.3 apresenta-se o modelo de negócio para o Validador, sendo visíveis as diferentes fases que dele fazem parte.

7.3 Visão de Arquitectura

A arquitectura escolhida pode ser definida em três camadas que providenciam as funcionalidades necessárias ao processo de validação de uma aplicação altamente configurável.

Os três níveis funcionais necessários para o funcionamento do `Validador` são os seguintes:

- Configuração do repositório de factos;
- Inserção de Regras `RV`;
- Validação de Regras e Apresentação dos Resultados;

Na Figura 7.4 é apresentada uma visão geral da arquitectura para o `Validador`.

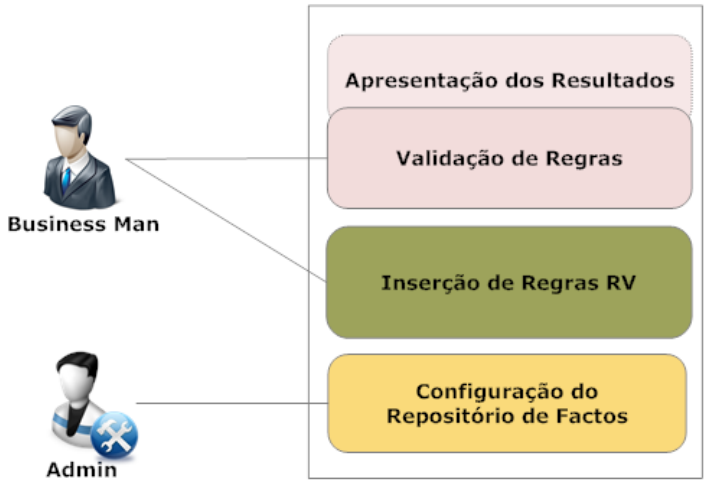

Figura 7.4: Visão de Alto Nível da Arquitectura

A camada de configuração do repositório de factos é responsável por configurar o acesso à aplicação alvo que se pretende validar de forma a tornar possível a comunicação entre o `Validador` e a "Aplicação Configurável" para que seja possível o acesso às informações e dados (factos) da última para o processo de validação das regras.

A camada de inserção de regras de validação (`RV`) é responsável por providenciar mecanismos para a inserção das regras (num repositório de regras), para serem posteriormente validadas. Esta camada irá fornecer uma interface para que o utilizador possa efectuar essa mesma introdução.

Por último, a camada de validação de regras e apresentação dos resultados é responsável por fornecer um mecanismo para ordenar a configuração e execução do processo de validação das regras previamente introduzidas no repositório. Cada uma dessas regras deverá, neste momento, ser processada e executada. Sendo os resultados posteriormente disponibilizados ao utilizador através de páginas *web* ou outros interfaces externos (e.g.: *webservices*).

7.3.1 Especificação da Arquitectura

Após a apresentação do modelo geral da estrutura do nosso sistema, pretende-se agora refinar o modelo previamente apresentado através da decomposição do sistema em **componentes** (elementos estruturais).

Para cada um dos componentes pretende-se apurar as suas responsabilidades, e identificar as inter-ligações existentes entre os diferentes componentes que compõem o sistema.

Apresentam-se de seguida a listagem desses componentes:

- **Knowledge Repository** (ou Repositório de Regras) — componente responsável por armazenar de forma centralizada todas as regras `RV` que se pretendem validar sobre a nossa aplicação altamente configurável.

- **Facts Repository** (ou Repositório de Factos) — componente responsável por armazenar os dados provenientes da aplicação alvo de validação e que são necessários para executar a validação das regras existentes no `Repositório de Regras`.

- **Integrator** (ou Integrador) — este componente, depois de correctamente configurado, é responsável por:

  - aceder à aplicação alvo de modo a obter dados de configuração da mesma;
  - converter os dados provenientes da aplicação para o modelo utilizado para a representação dos factos no `Repositório de Factos`;
  - gerar o Modelo de Factos.

  Na Figura 7.5 podemos observar as interacções deste componente.

- **Rules Manager** (ou Gestor de Regras) — componente responsável por toda a gestão das regras `RV`, mais especificamente no que diz respeito:

  - à introdução/edição das regras para o `Repositório de Regras` a partir de um ficheiro ou editor gráfico específico para o efeito;
  - ao controlo de acessos ao `Repositório de Regras`, necessário para posterior validação;

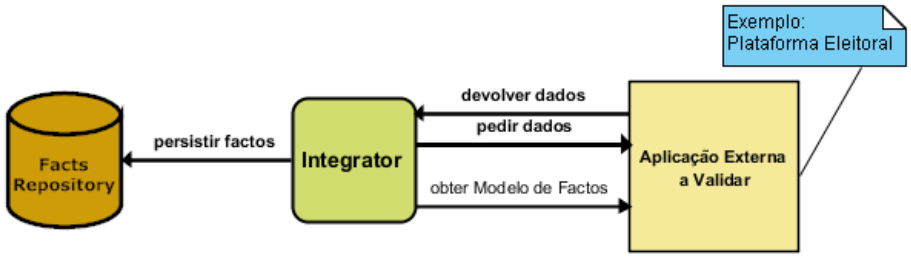

Figura 7.5: Componente: Integrator

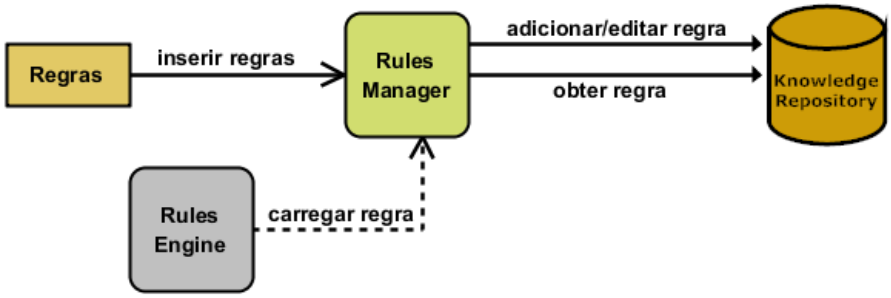

Figura 7.6: Componente: Gestor de Regras.

Na Figura 7.6 podemos observar as interacções deste componente.

- **Rules Engine** (ou Motor de Regras) — componente responsável por "decidir" o que fazer para validar uma determinada regra `RV` do `Repositório de Regras`. Da lista de responsabilidades deste componente fazem parte:
  - obter as regras `RV` a validar do `Repositório de Regras`, por intermédio do `Gestor de Regras`;
  - obter os factos necessários, para validar determinada regra, do `Repositório de Factos`;
  - receber ordem de execução do processo de validação dos factos contra as regras existentes;
  - inferir o resultado da validação de cada uma das regras `RV` do Repositório de Regras.

Na Figura 7.7 podemos observar as interacções deste componente.

- **Rules Processor** (ou Processador de Regras) — componente responsável por desencadear o processo de validação da aplicação altamente configurável, bem como de expor os resultados dessa validação para o utilizador. Para além disso é responsável por determinar como deve a validação ocorrer (e.g.: se devem ser validadas todas as regras do repositório, ou não.).

Na Figura 7.8 podemos observar as interacções deste componente.

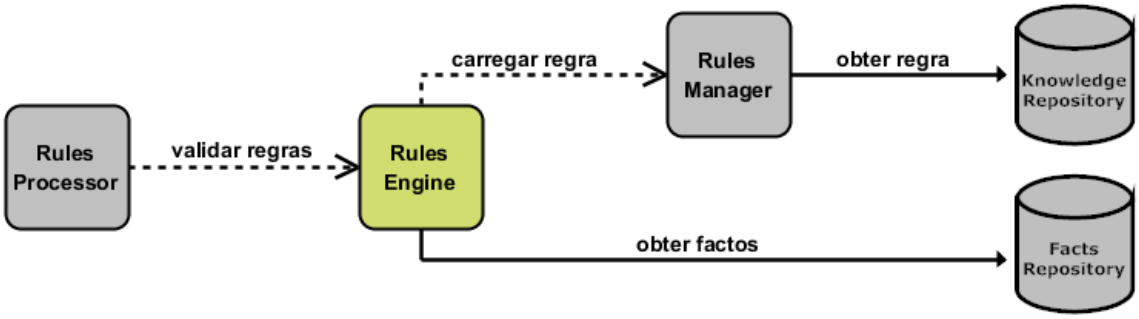

Figura 7.7: Componente: Motor de Regras.

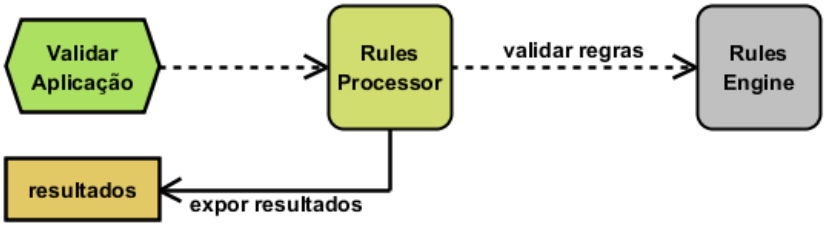

Figura 7.8: Componente: Processador de Regras.

Feito isto apresentamos em seguida na Figura 7.9 um diagrama de arquitectura conceptual com todas as interacções entre os componentes previamente introduzidos.

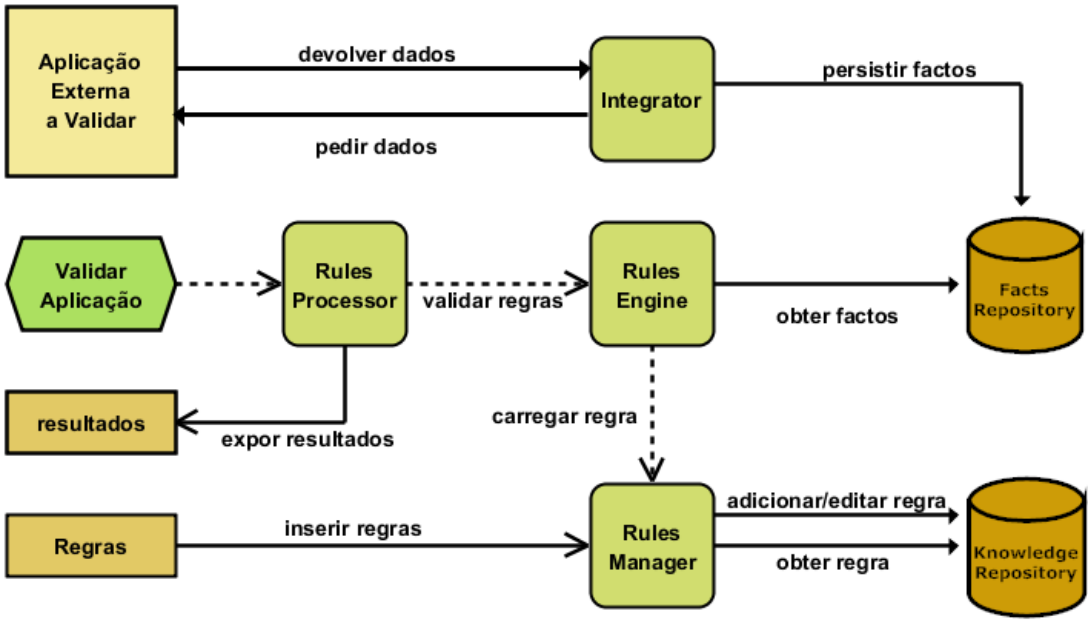

Figura 7.9: Arquitectura Conceptual do Sistema.

Capítulo 8

Desenvolvimento do Sistema de Validação

"A scientist builds in order to learn; an engineer learns in order to build."

Fred Brooks

Após o desenho da arquitectura do sistema validador, interessa-nos demonstrar a viabilidade da estratégia de validação definida, bem como da arquitectura já apresentada, através da especificação e implementação de uma *Prova de Conceito* (POC)[Dic11] utilizando Java [Ora11] como linguagem principal.

Para a realização da prova de conceito tem-se uma abordagem partida pelos diferentes componentes a desenvolver, identificados no modelo arquitectural. É importante, para cada componente: especificar o funcionamento do negócio; identificar ferramentas e linguagens a utilizar; anotar decisões técnicas na utilização das ferramentas; e mostrar detalhes sobre a concretização da sua implementação.

Como **ponto chave** para o desenvolvimento do sistema validador, optou-se pela **utilização de uma ferramenta** já referenciada no capítulo 5.1 como sendo um sistema de gestão de regras de negócio (BRMS) — o **Drools**.

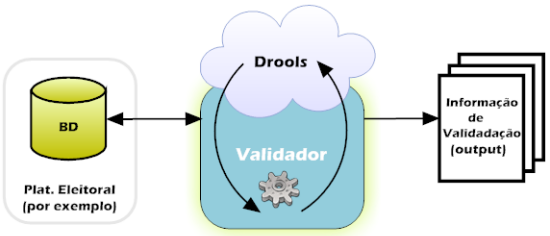

Figura 8.1: O Drools no Sistema de Validação.

Como se pode ver na Figura 8.1 o Drools surge como uma peça *core* da solução proposta, não só fornecendo um conjunto de ferramentas muito úteis ao desenvolvimento do sistema pretendido, mas também por servir como agregador dos diversos

componentes.

Ao longo do capítulo, serão fornecidos os detalhes necessários para adaptar e incorporar o Drools na solução, porém, para uma visão mais ampla das potencialidades desta ferramenta poderá consultar-se o Apêndice D.

Nas secções seguintes apresenta-se o modo pela qual o desenvolvimento de cada um dos componentes da arquitectura será endereçado. Desde o Repositório de Factos (secção 8.1) e de Regras (secção 8.2), passando pelo Integrador (secção 8.3) e pelo Gestor de Regras (secção 8.4) e finalizando com o Motor e Processor de Regras (secção 8.5).

8.1 Repositório de Factos

Na concepção do `Repositório de Factos` (Facts Repository), as questões que imediatamente se colocam são:

- que estrutura deverá ser usada para persistir os factos?
- como será populada a estrutura de dados do repositório?
- que tipo de informações irão os factos conter?

A ideia passa por encontrar o modelo/estrutura para suportar a representação, de forma inequívoca, das informações provenientes das aplicações altamente configuráveis — **Modelo de Factos**. Sendo que o modo como estas informações serão explicitadas deverá ser genérico o suficiente para que sejam independentes do negócio da aplicação alvo de validação.

Para além disso, as `RV` (cuja concepção será abordada mais tarde na secção 8.2) deverão ser definidas tendo em consideração este modelo, para facilitar posterior acesso a esta mesma informação.

Ora, acontece que o Drools[1] (em específico o Drools Runtime) possui uma *interface* — `org.drools.runtime.StatefulKnowledgeSession` — na sua *Application Programming Interface* (`API`), que permite estabelecer o diálogo com o seu motor de regras, mantendo o estado dessa sessão. Sendo essa mesma sessão que irá manter os nossos factos em memória.

Os mesmos factos que serão posteriormentes validados, são escritos usando Java, em que cada facto é uma instância da classe java.lang.Object, a qual representa o topo da hierarquia de classes desta linguagem (todas as classes são do tipo Object).

[1] **Versão:** Neste trabalho de mestrado foi utilizada a **versão 5.1.1** do Drools.

Listagem 8.1: Criar Sessão e Adicionar Factos ao repositório

```
1 StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();
2
3 /** Add FACTS to the FactRepository. */
4 final List<Object> electionFacts = FactsRepositoryFactory.factsConstructor();
5 for (final Object i : electionFacts) {
6    ksession.insert(i);
7 }
```

Desta forma, utilizando como suporte a referida `API` do Drools, todos os nossos factos serão carregados para a *sessão* (em memória) imediatamente no início do processo de validação e antes da execução de qualquer regra, como se pode ver pela Listagem 8.1.

A aplicação altamente configurável a validar fornecerá os **factos** (dados) necessários para popular o modelo definido para o repositório. O componente responsável por esse processo é o Integrador, que será abordado na secção 8.3 deste capítulo.

8.2 Repositório de Regras

No que diz respeito ao `Repositório de Regras` (Knowledge Repository), é importante ter em atenção que as regras `RV`:

- deverão ser capazes de exprimir qualquer regra que evite os problemas descritos no capítulo 6.2.2;
- deverão ser definidas/expressas de forma independente do modelo de negócio da aplicação;
- deverão ser interpretadas posteriormente aquando do *carregamento* dos factos que lhe estão associados para o `Repositório de Factos`;

Como tal, utilizamos mais uma vez as potencialidades do Drools para lidar com as regras a validar. Neste caso utilizamos a *interface*:

- `org.drools.definition.rule.Rule` — como suporte às nossas regras de validação;
- `org.drools.KnowledgeBase` — para o nosso repositório de regras.

Sendo que, o nosso repositório (`KnowledgeBase`) será preenchido a partir de um RulesFactory (Listagem 8.2), o qual adicionará as nossas regras de validação a partir de *resources* externos à aplicação (Listagem 8.3), os quais serão tipicamente ficheiros de extensão `.DRL` gerados ou escritos por um utilizador.

Listagem 8.2: Carregar o repositório de regras

```
1
2 // load up the knowledge base
3 final KnowledgeBase kbase = RulesFactory.readKnowledgeBase();
```

Listagem 8.3: Adicionar regras ao repositório a partir de resources .DRL

```
1
2  final KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
3  kbuilder.add(ResourceFactory.newClassPathResource("Rules.drl"), ResourceType.DRL);
4
5  final KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
6  kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
```

A mesma *sessão* já referida na secção anterior, e a qual estabelece o diálogo com o motor de regras, é criada a partir do `KnowledgeBase` (Listagem 8.4). Permitindo que sobre um repositório de regras, tenhamos diferentes *sessões*, ou seja diferentes repositórios de dados em simultaneo. Sendo que estas mesmas *sessões* podem ser actualizadas sempre que seja adicionada ou removida um regra do repositório [Bal09] para que as condições de uma possível nova regra sejam avaliadas imediatamente para cada facto existente.

Listagem 8.4: Criação da `KnowledgeSession`

```
1
2 ksession = kbase.newStatefulKnowledgeSession();
```

No caso concreto da nossa proposta de solução interessa-nos trabalhar unicamente com uma *sessão*, ou seja, um único repositório de dados. E, as nossas regras serão, tal como os factos, carregados para o repositório no início, não havendo alterações às regras (ou aos factos) durante o processo de validação.

8.2.1 Construir Regras de Validação

Para a construcção das regras `RV` utilizamos, mais uma vez, o Drools como base permitindo-nos expressar as regras de modo simples por um lado e por outro de forma independente, com a vantagem que o Drools fornece inúmeros mecanismos para a escrita das regras `RV` (mais detalhes podem ser encontrados no Apêndice D). Podendo utilizar-se:

- o `Drools Guvnor` [JBo11, Bro09] — o qual permite o uso:
  - de um editor gráfico (Figura 8.2);
  - de um editor textual (Figura 8.3);
  - de tabelas de decisão (Figura 8.4); as quais podem ser construídas previamente em Excel e importadas à posteriori para o Guvnor; ou então construídas dinâmicamente através de um formulário *web*.
- um simples editor de texto (como por exemplo o vi [CSU11, Gil11, vim11], ou o notepad++ [Ho11]);
- o Drools IDE para o Eclipse [Fou11];

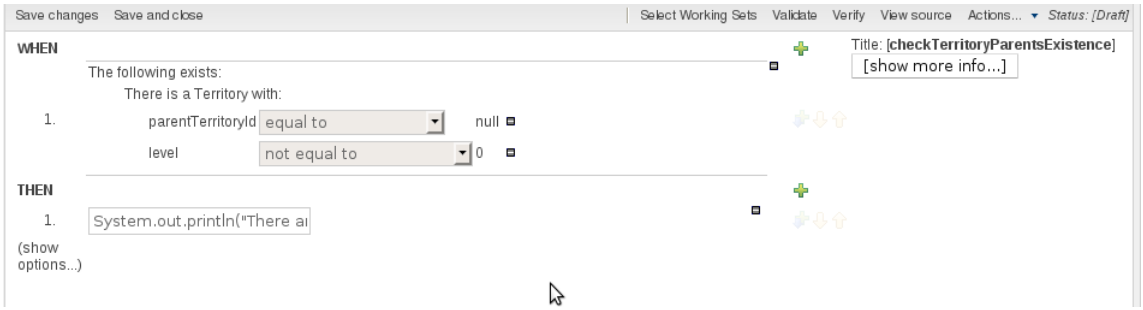

Figura 8.2: Drools Guvnor — editor gráfico.

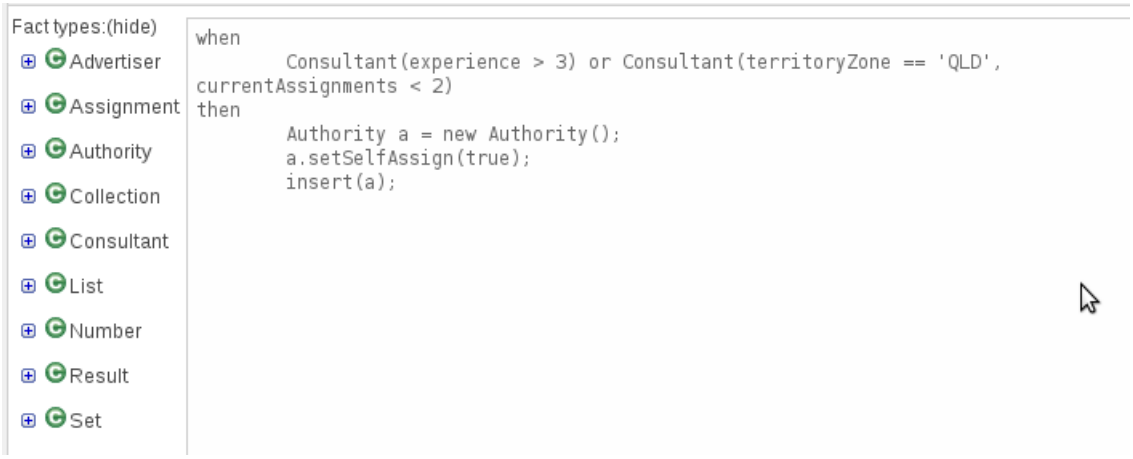

Figura 8.3: Drools Guvnor — editor textual.

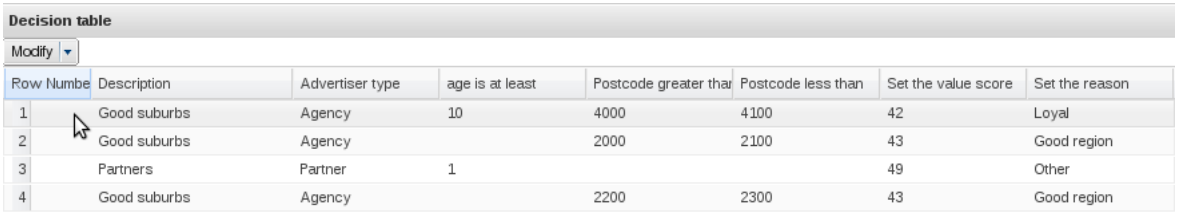

Figura 8.4: Drools Guvnor — tabela de decisão.

Indepentemente do método utilizado para escrever as regras, os dialectos suportados pelo Drools são [Bal09]:

- Java — o Drools suporta todos os tipos nativos de Java, incluindo expressões regulares;
- mvel [B+11] — uma linguagem de expressão para aplicações com base no Java mas que facilita não só a leitura das regras, mas também a escrita das mesmas especialmente para utilizadores que não conheçam o Java com linguagem de programação;

A forma como tais regras são expressas deve cumprir com a definição do nosso **Modelo-de-Factos**, o qual é gerado pelo `Integrator`, permitindo que as regras `RV` escritas sejam adaptáveis ao modelo de negócio da aplicação que pretendemos validar.

Formato das Regras: As regras devem seguir o seguinte formato [2]:

When `X` $\Rightarrow$ `Y`

Em que:

- **'X'** — representa a **acção**, e deverá ser escrita na negativa;
- **'Y'** — representa a **reacção**, e deverá ser, no contexto do nosso sistema validador, uma mensagem de excepção a ser devolvida ao utilizador.

Ora, todas as regras `RV` são usualmente descritas através de frases afirmativas, como podemos constatar pelo Apêndice A. Assim sendo, para podermos utilizar correctamente o nosso validador, precisamos de as **transformar** em frases na negativa — para essa transformação utilizaremos as Leis de DeMorgan.

De modo sucinto, as **Leis de DeMorgan** relacionam as operações lógicas: 'união' ($\vee$), 'intersecção' ($\wedge$) e 'negação' ($\neg$), permitindo-nos negar afirmações através da aplicação de regras de lógica [dO82]. (Para informações mais detalhadas pode-se consultar o Apêndice E).

Exemplo: A título de exemplo, pegaremos na seguinte `RV`:

- *Todos os territórios devem ter um número de eleitores inscritos de referência superior a zero (0)*;
  Formato: $\forall x \in A : p(x)$

Regra esta que, pretendemos:

1. negar — ficando no formato: $\neg[\forall x \in A : p(x)]$;

[2] Que é o mesmo que dizer: "Quando 'X' então 'Y'."

2. aplicar as leis de DeMorgan — neste caso em específico será aplicado o Teorema 3: $\neg[\forall x \in A : p(x)] \Leftrightarrow \exists x \in A : \neg p(x)$

Desta forma acabamos de obter a variável 'X' da nossa regra final; a qual, após traduzida para linguagem natural fica:

- `When X` ≅ *Quando existe um território pertencente ao conjunto de territórios, tal que os inscritos de referência não sejam superiores a zero(0)*; ⇒ `Y` ≅ *então atiramos uma mensagem de erro.*

Após esta conversão poderemos então escrever a regra para que possa ser posteriormente importada para o nosso sistema validador. Na Listagem seguinte podemos ver a regra expressa utilizando como dialecto principal o mvel:

```
1  rule "zeroReferenceSubscribers"
2      dialect "mvel"
3      when
4          $list : java.util.List( )
5                       from collect ( Territory(referenceSubscribers == 0 ||
6                                      referenceSubscribers == null) )
7          eval ($list.size() > 0)
8      then
9          System.out.println("- There are: " + $list.size() +
10                            " Territories with no Reference Subscribers:");
11         for(Territory $i : $list){
12            System.out.println("      - " + $i.pk.region + "-" + $i.pk.id);
13         }
14 end
```

8.3 Integrador

Este é um dos componentes mais importante para o bom funcionamento do sistema validador. Não sendo o componente *central*, requereu no entanto bastante esforço para definir uma arquitectura que garanta o carácter modular e adaptável; não só deste mesmo componente, mas de todo sistema validador.

Relembramos que o Integrador (Integrator) é o responsável por estabelecer a comunicação com a aplicação alvo de validação, e por conseguinte, o responsável:

- por obter as configurações a validar do mesmo — factos a serem carregados para o repositório de factos;

- pela **geração** do `Modelo de Factos` — importante para o suporte dos factos carregados, e também para auxilio na escrita das regras a serem processadas.

Assim sendo, para se conseguir o desenvolvimento do nosso Sistema Validador de modo a que seja aplicável a diversas e variadas aplicações configuráveis, o componente Integrador deverá ser capaz de se ajustar às diferentes formas de se especificarem configurações, tal como foi visto na secção 5.2.

No decorrer da modelação deste componente aplicou-se um ***Design Pattern*** muito conhecido — Factory Method: o qual delega as responsabilidades de instanciação de objectos para as sub-classes [GHJV95].

Definição 7. ***Design Patterns:*** *representam boas soluções para um problema comum (com contexto bem definido). Por já terem sido testadas facilitam a implementação, teste e manutenção das aplicações onde aplicadas; e para além disso definem um modelo standard que é reconhecido por outros developers, tanto para comunicação como documentação, melhorando o nível de programação de cada um [GHJV95, GHJ+93, AMC+03].*

Desta forma é-nos possível ter diferentes processos/implementações permitindo que aplicações com diferentes caracteristicas no que diz respeito às configurações possam ser tratadas de forma adequada. No diagrama da Figura 8.5 podemos observar a existência de dois métodos na *interface* `FactModel`:

- `generateFactModel()`: responsável pela geração das classes pertencentes ao modelo de factos;
- `getFacts()`: responsável por obter os factos sobre os quais recairão as acções de validação.

Estes métodos serão implementados por cada uma das sub-classes:

- `DBFactModel`: implementação para configurações existentes em bases de dados;
- `FileFactModel`: implementação para configurações provenientes de ficheiro;

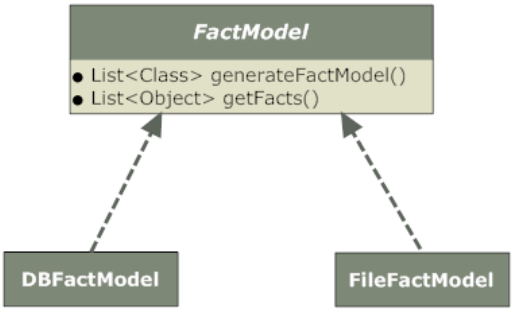

Figura 8.5: `FactoryMethod` aplicado no Integrador.

Para a prova de conceito desenvolvida e, tendo em consideração o caso de estudo — `Plataforma Eleitoral` — optamos pela implementação do `DBFactModel`; ou seja um Integrador preparado para aplicações cujas configurações se encontram numa base de dados.

Antes de mais, para podermos aceder as configurações de uma determinada aplicação, quando estas se encontram numa base de dados relacional iremos necessitar à partida da seguintes informações para estabelecer a ligação:

- *Java Database Connectivity* (`JDBC`) URL [3];
- *username* de acesso;
- respectiva *password*;
- *Schema*(s) da base de dados a explorar;
- class do *driver* e dialecto utilizado.

A primeira etapa a realizar pelo Integrador é a geração do Modelo de Factos. Para esse efeito recorreu-se à utilização do **Hibernate Tools** [JBo12b] e do **Maven** [Fou12]. Com estes torna-se relativamente simples a geração das classes (*Plain Old Java Objects* (`POJO`)) das entidades que representam o modelo de factos; basta para tal utilizar um *plugin* do Hibernate integrado no Maven com as configurações de acesso à base de dados — como podemos ver pela Listagem 8.5.

Listagem 8.5: Configurações de acesso a uma Base de Dados.

```
1
2 <appconfig>
3   <dbType>PostgreSQL</dbType>
4   <driver>com.postgresql.Driver</driver>
5   <jdbcConnectionString>jdbc:postgresql://localhost:5432/legislativas</
        jdbcConnectionString>
6   <databaseIP>localhost</databaseIP>
7   <databaseCatalog>legislativas</databaseCatalog>
8   <databaseUsername>postgres</databaseUsername>
9   <databasePassword>gammaray</databasePassword>
10  <sourceTarget>/home/pedro/validator-project</sourceTarget>
11  <projectName>Validator</projectName>
12  <topLevel>com.uminho</topLevel>
13  <schemaStrategy>PARTIAL</schemaStrategy>
14 </appconfig>
```

Após a geração do Modelo de Factos, interessa-nos exportar esse mesmo modelo num *package Java Archive* (`JAR`) para que fique disponível para utilização no editor de regras como o `Guvnor`, se assim se entender.

Posteriormente, e apenas após a geração do Modelo de Factos, procedemos à captura dos dados das configurações. Para o fazermos seguimos o seguinte algoritmo:

1. utilizando *introspecção* em Java, obter um *array* ou lista de todas as classes (`POJO`) geradas no modelo de factos;

[3] **URL:** Uniform Resource Locator.

2. iterar sobre cada uma dessas classes e construir uma *query* **HQL** na forma:

   "Select x from" + `Class.getName()`

3. coleccionar, com o auxilio do Hibernate todos os objectos devolvidos na execução das queries definidas no ponto anterior;

4. adicionar cada uma desses objectos ao `StatefulKnowledgeSession`, tal como mostrado na Listagem 8.1.

Com isto temos então o nosso Integrador preparado para funcionar com aplicações configuravéis cujas configurações se encontrem numa base de dados. Para qualquer outro caso, necessitaremos de implementar (caso não tenha sido ainda feito) uma outra classe concreta que implemente a *interface* `FactModel` — ver Figura 8.5 — com o comportamento desejado;

8.4 Gestor de Regras

O `Gestor de Regras` (Rules Manager) é um componente que tem como suporte base, o **Drools Guvnor**; o qual fornece todas as funcionalidades desejadas, e cuja reutilização evita esforços de implementação para o desenvolvimento deste mesmo componente.

Para começar é importante referir que a importação do **Modelo de Factos** gerado pelo Integrador, é suportado pelo Guvnor, bem como a exportação das regras `RV` construídas tendo como base esse mesmo modelo; as quais são posteriormente importadas pelo `Validador`, tal como se pode ver pela Listagem 8.3.

Para além destas funcionalidades básicas, o Guvnor fornece um conjunto de operações, tais como:

- controlo de acessos por *login* e *password*;
- controlo de versões das regras `RV` num repositório próprio;
- mecanismo de *deploy* das regras;
- importação e exportação dos dados;
- edição de regras utilizando diferentes formatos;
- mecanismo de testes unitários para as regras `RV`;
- suporte para compilação de regras permitindo a tradução das mesmas para uma linguagem mais perto de uma *linguagem natural* e que o Rules Engine consiga entender.

Quaisquer outras informações sobre o Guvnor poderão ser consultadas no Apêndice D ou, na bibliografia apontada.

8.5 Processador de Regras & Motor de Regras

Tal como os outros componentes, também estes têm como base as `API`'s do Drools.

O `Processador de Regras` (Rules Processor) é o componente do Validador que funciona como ponto de partida na medida em que é este que desencadeia o processo de validação — Listagem 8.6; e é também aquele que expõe os resultados para o utilizador — Listagem 8.7.

Listagem 8.6: Início do processo de validação

```
1
2   // Fire RULES!
3   ksession.fireAllRules();
```

Listagem 8.7: Exposição dos Resultados da Validação

```
1
2   // Get to know which rules were not executed! = Logging Working Memory
3   System.out.println("-----------------------------------------");
4   displayRules("Rules Validated (and satisfied):", ruleNames);
```

Mais ainda se deve ter em atenção que é este mesmo componente que define o modo como as regras são processadas e a interpretação que lhes é atribuída. Em que, segundo o que já foi apresentado neste capítulo, podemos afirmar que:

- Se nenhuma regra for despoletada, então isso significa que as condições (regras `RV`) fornecidas para validação foram satisfeitas.
- Se por algum motivo, uma regra `RV` não for satisfeita, então é impressa uma mensagem de erro.

No que diz respeito ao `Motor de Regras` (Rules Engine), importa explicitar o funcionamento do processo de validação de um regra `RV`, definido pela utilização da `API` do Drools.

Ora, este processo é implicitamente desencadeado a partir do momento em que o RulesProcessor despoleta o início do processo de validação. Internamente é utilizado um algoritmo `Rete` [For79, For82] [4], o qual procura fazer *match* entre as regras e os factos existentes em cada um dos respectivos repositórios de forma eficiente.

Algoritmo Rete: É implementado através da construcção de um grafo orientado, cujos nodos representam um ou mais testes encontrados na condição "*quando*" da regra — *Left Hand Side* (`LHS`). No nível mais baixo (nodos folha) da rede encontram-se os nodos que representam regras individuais.

Quando um conjunto de factos atravessa todo o caminho até às folhas, então significa que passou todos os testes no `LHS` de uma regra particular, tornando-se

[4] A palavra '*Rete*' tem origem no latim e significa *rede*.

assim uma **activação**. Isto significa que a condição "*então*" — *Right Hand Side* (`RHS`), pode ser executada (*fired*).

Na Figura 8.6 podemos observar o grafo construído, seguindo o Algoritmo de Rete, pelo `Drools` para a regra que demos como exemplo na secção 8.2.1. Todos os grafos têm sempre início num nodo branco (único) e, terminam num nodo preto, que representa uma regra do repositório; tendo o grafo tantos nodos pretos quanto o número de regras avaliadas pelo Algoritmo de Rete. A primeira fase do algoritmo analisa a cláusula '*quando*' de cada uma das regras e, identifica os factos (nodos a vermelho) que lhes estão associados. Feito isto, decompõe-se cada um dos factos nos vários elementos que lhe pertencem (nodos a azul). Tanto os factos como os seus elementos podem ser inter-ligados por nodos intermédios (nodos verdes e amarelos) que servem de *cache* permitindo optimizar o número de verificações necessárias para chegar até um nodo final. Quando atingimos os nodos pretos no fundo do diagrama, isso significa que as condições dessa regra foram satisfeitas e como tal a regra pode ser *despoletada*.

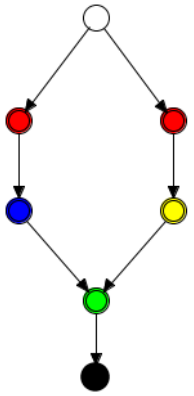

Figura 8.6: Grafo de Nodos seguindo o Algoritmo Rete.

Dado que no nosso caso em particular, os nossos factos são configurações e portanto não alteráveis em *runtime*, significa que cada regra é executada apenas uma e uma só vez. Para além disso, relembra-se que o `RHS` das nossas regras `RV` é tipicamente uma mensagem de excepção; o que significa que o despoletar da regra corresponde em termos básicos à execução de código Java.

Para além do que já foi referido, sobre o nosso Rules Engine foi adicionado um *Event Listener* — ver Listagem 8.8, o qual permite rastrear o estado de execução das regras bem como os resultados da validação. Estas mesmas informações, para além de funcionarem como `log` do Validador, são posteriormente transmitidas ao Rules Processor que se encarregará de expor os resultados.

Listagem 8.8: Adição de um *Event Listener*

```
1
2 // Setting handler for rules.
3 final TrackingAgendaEventListener listener = new TrackingAgendaEventListener(rules
      , ruleNames);
4 ksession.addEventListener(listener);
```

8.6 O Sistema Desenvolvido

O sistema desenvolvido assenta portanto nas seguintes linhas de acção:

- utilização do Java como linguagem de implementação;
- utilização do Jboss Drools como ferramenta *core*:
  - na concepção do `Repositório de Factos` e do `Repositório de Regras`;
  - na construcção das regras de validação quer pela utilização do Guvnor ou do Drools IDE;
  - no desenvolvimento de um motor para validação de regras;
- construcção de um Integrador de forma modular para geração de `Modelos de Factos` e captura de configurações, que suporte diferente aplicações altamente configuráveis; recorrendo para tal ao Maven e ao Hibernate Tools no desenvolvimento de um Integrador orientado para aplicações configuráveis com suporte em bases de dados;
- interpretação dos resultados tendo em consideração que todas as regras de validação devem ser escritas seguindo o mesmo principio:

  When `X` $\Rightarrow$ `Y`

  Sendo `'X'` a **acção** escrita na negativa e, `'Y'` a **reacção**, que deverá ser uma mensagem de excepção a ser devolvida ao utilizador.

Em seguida, e para um melhor entendimento do seu funcionamento, apresentamos um diagrama de actividades [VP11] com o fluxo de execução do Validador.

Como podemos observar pela Figura 8.7, tudo começa com a configuração do `Integrador` de modo a ajustar correctamente o Validador à aplicação configurável. Uma vez realizada essa configuração, é então possível a geração do modelo de factos que será útil para a escrita das regras. Após a geração do modelo podemos realizar a exportação do mesmo para o Guvnor onde escreveremos as regras que serão depois importadas de volta para o Validador e adicionadas ao *Repositório de Regras*; simultaneamente podemos capturar os factos à aplicação configurável a validar e adicioná-los ao *Repositório de Factos*. Feito isto e, após a sincronização das actividades temos tanto o repositório de regras como o repositório de factos preenchidos e portanto podemos ordenar o processo de validação (*'Fire All Rules'*) do qual resultará um output com o resultado dessa mesma validação, acção com a qual se encerram as actividades do sistema Validador.

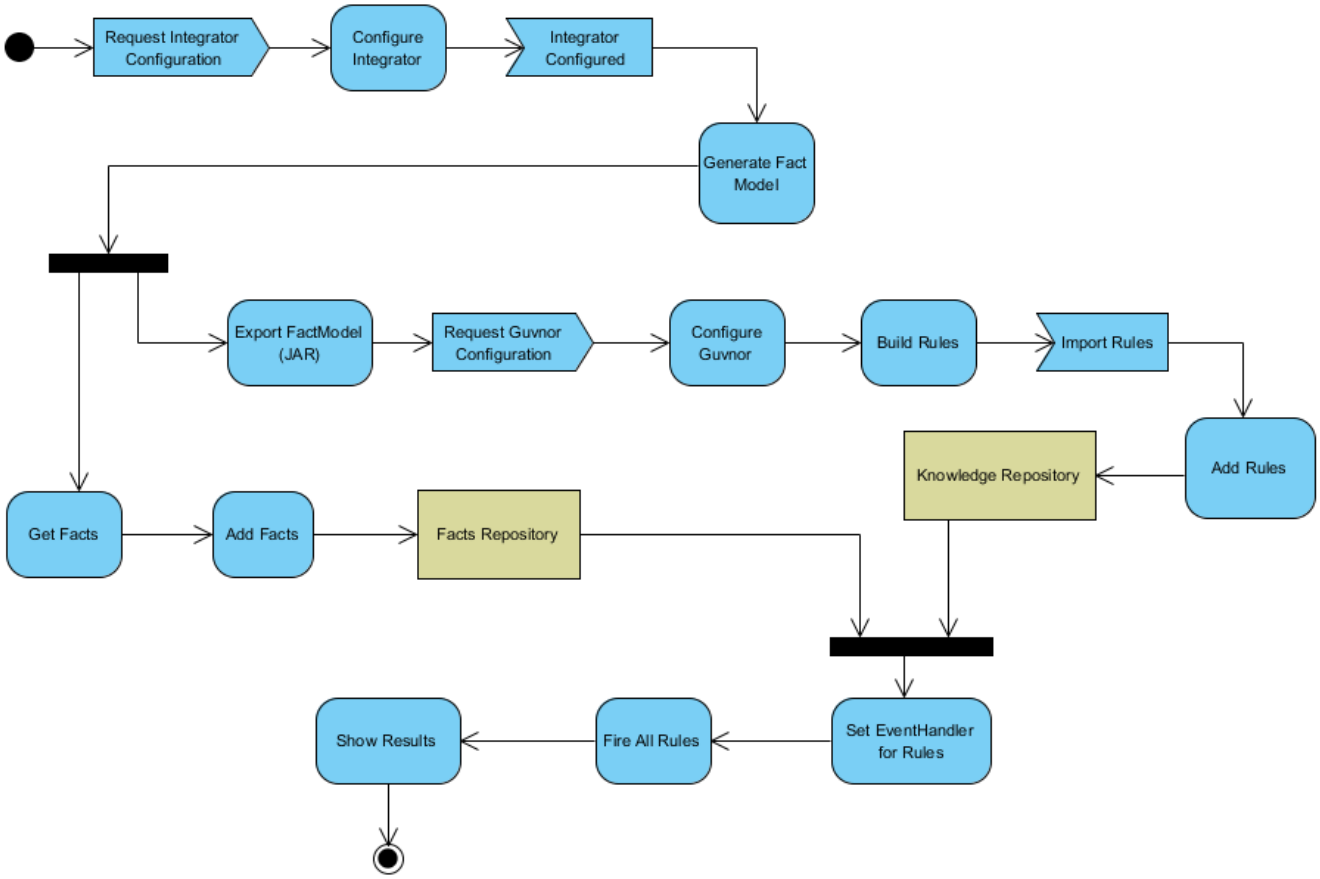

Figura 8.7: Fluxo de execução do Validador

Capítulo 9

Avaliação da Solução

"The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' (I found it!) but 'That's funny ...' "

Isaac Asimov

Este capítulo pretende servir como forma de validar a proposta no contexto concreto da `Plataforma Eleitoral`, ilustrando algumas das questões que se pretendem ver resolvidas, bem como avaliar o comportamento da solução adoptada na resolução de tais problemas.

De um modo geral, este capítulo reune um conjunto de informações que têm sido já apresentadas ao longo desta dissertação de mestrado e, pretende facilitar o entendimento do problema e da abordagem seguida para chegar até à solução, permitindo a sua validação prática.

9.1 O Caso de Estudo

A `Plataforma Eleitoral` é uma aplicação altamente configurável, cujo objectivo é o de se adaptar às mais diversas eleições através de alterações das regras de negócio da Administração Eleitoral a partir das configurações expressas em múltiplos ficheiros `CSV` e `XML`.

Pelo facto de as regras de negócio serem provenientes de ficheiros de configuração em separado, torna-se díficil de executar determinadas validações dessas mesmas configurações.

- Tome-se como exemplo as seguintes configurações relacionadas com a definição da hierarquia de territórios (Listagem 9.1), a definição da árvore de territórios (Listagem 9.2) e dos círculos eleitorais associados (Listagem 9.3). Todos os dados mencionados encontram-se estrictamente relacionados entre si, no entanto, são necessariamente definidos em separado.

Listagem 9.1: Excerto do ficheiro *territory-hierarchy.csv*

```
1 Region , Level , Description
```

```
2 GLOBAL,0 ,GLOBAL_RESULTS
3 LOCAL,1 , Pais
4 LOCAL,2 , Distrito
5 LOCAL,3 , Municipio
6 LOCAL,4 , Freguesia
```

Listagem 9.2: Excerto do ficheiro *territory.csv*

```
1 Region , Territory_ID , Description ,[ Parent_Region ] ,[ Parent_ID ]
2 GLOBAL,990000 , Resultados Globais , ,
3 LOCAL,500000 , Portugal ,GLOBAL,990000
4 LOCAL,010000 , Aveiro ,LOCAL,500000
5 LOCAL,010100 , Agueda ,LOCAL,010000
6 LOCAL,010101 , Agadao ,LOCAL,010100
```

Listagem 9.3: Excerto do ficheiro *constituencies.csv*

```
1 ElectionID , Region , TerritoryID , Mandates
2 AF,LOCAL,010101 ,5
3 AM,LOCAL,010100 ,12
4 CM,LOCAL,010100 ,10
```

No entanto, sempre que exista uma regra definida num ficheiro de configuração comum a todo o processo eleitoral (independente da eleição), que seja possível validar genericamente, então a decisão foi de incluir a sua validação na própria aplicação.

- Um exemplo disso é a validação que é efectuada sobre a definição dos territórios seleccionados para afluências. Tal validação é feita a nível da aplicação porque a legislação diz que qualquer território seleccionado para recolha de dados de afluências deverá ser sempre um território `LOCAL`, o que em termos práticos significa que deverá ser sempre uma freguesia, independentemente da eleição em causa.

Todas as restantes regras específicas foram até ao momento validadas manualmente, o que acarreta não só um custo mais elevado na execução de tais tarefas, como também não garante que não se cometam omissões ou enganos.

É particularmente devido a estas últimas regras de negócio enunciadas não poderem ser validadas de forma automática pela aplicação e pelo facto das mesmas permitirem que se introduzam na aplicação erros que podem ser catastróficos para a execução da mesma que a questão de se validarem as regras de negócio da aplicação surgiu como um desafio construtivo.

9.2 Validação

Após se analisar o problema, percebeu-se que para resolver a validação de tais regras, o importante é validar o resultado da importação das configurações para o contexto da aplicação (e não as configurações por si só que são definidas nos vários ficheiros). Assim sendo, e após termos a aplicação configurada, o processo a seguir para usar o sistema de validação aqui proposto passa por:

1. definir as regras de negócio de forma mais simples possível, utilizando um dos dialectos reconhecido pelo `Drools`;

2. processar cada uma dessas regras no contexto da aplicação (implica ter acesso ao modelo de domínio, e aos dados que configuram o negócio da aplicação);

3. obter o resultado de processar cada uma das regras;

Após o `passo 3`, a equipa de desenvolvimento deve ainda:

1. tirar ilações sobre os resultados;

2. corrigir erros (fora do âmbito de validação);

3. voltar ao `passo 2` da anterior enumeração.

Nos próximos parágrafos serão descritos todas as etapas percorridas para a **validação** da Plataforma Eleitoral, neste caso específico, configurada para umas eleições *Autárquicas* [dAID10a].

Sendo o objectivo primordial, o de testar os diversos componentes do sistema Validador, serão percorridas, e detalhadas o melhor possível, cada uma das fases de execução do mesmo, para uma representação o mais fiável possível de como seriam os passos descritos numa outra qualquer situação.

Assume-se portanto, como ponto de partida, a existência de uma instância da Plataforma Eleitoral já configurada para umas eleições *Autárquicas*[1], resultado do carregamento de **14 ficheiros de configuração** (os quais podem ser encontrados no Apêndice B).

9.2.1 Pré-Configurar Validador

Assim sendo, numa primeira fase do Validador, precisamos de configurar o funcionamento do Integrador para que se conecte à base de dados PostgreSQL utilizada pela plataforma. Para tal actualizámos o ficheiro de configuração geral do Validador (Listagem 9.4), o qual contém informações acerca da implementação concreta (*classe*) a ser utilizada pelo Integrador:

```
1 <package>com.uminho.validator.integrator.factory.DBFactModel</package>
```

bem como qual o ficheiro que contém as configurações necessárias por este:

```
1 <integrator-conf>integrator.conf</integrator-conf>
```

Para além disso, contém informações acerca do directório e do nome do ficheiro que será gerado e irá conter os factos:

```
1 <factModel>
2   <outputDir>/etc/validator/<outputDir>
3   <outputName>factModel</outputName>
4 </factModel>
```

[1] configurada com dados baseados na realidade.

Precisamos também de definir a configuração para o Integrador (Listagem 9.5) contendo as propriedades da conexão à base de dados, como o tipo da base de dados utilizada, o *driver* a utilizar bem como o endereço *Internet Protocol* (IP), porta e nome da base de dados; e ainda o utilizador e a sua password de acesso.

Listagem 9.4: Ficheiro `validator.conf`

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <integrator>
    <package>com.uminho.validator.integrator.factory.DBFactModel</package>
    <integrator-conf>integrator.conf</integrator-conf>
  </integrator>
  <factModel>
    <outputDir>/etc/validator/<outputDir>
    <outputName>factModel</outputName>
  </factModel>
  <rules>
    <importFile>/etc/validator/Rules.drl</importFile>
  </rules>
</configuration>
```

Listagem 9.5: Ficheiro `integrator.conf`

```
<?xml version="1.0" encoding="utf-8"?>
<appconfig>
  <dbType>PostgreSQL</dbType>
  <driver>com.postgresql.Driver</driver>
  <jdbcConnectionString>jdbc:postgresql://localhost:5432/autarquicas</
      jdbcConnectionString>
  <databaseIP>localhost</databaseIP>
  <databaseCatalog>autarquicas</databaseCatalog>
  <databaseUsername>postgres</databaseUsername>
  <databasePassword>gammaray</databasePassword>
  <sourceTarget>/home/pedro/validator-project</sourceTarget>
  <projectName>Validator</projectName>
  <topLevel>com.uminho</topLevel>
  <schemaStrategy>PARTIAL</schemaStrategy>
</appconfig>
```

9.2.2 Iniciar Validador

Depois de configurado o sistema validador, damos início à sua execução, o qual, utilizando as configurações indicadas gera o Modelo de Factos, e exporta-o para a directoria definida na configuração da Listagem 9.4.

Numa rápida análise ao JAR gerado, verificamos que contém **46** POJO's (entidades) geradas (Listagem 9.6). Estas classes, que definem o modelo de domínio da Plataforma Eleitoral, poderão ser utilizadas no editor de regras do Guvnor.

Listagem 9.6: Geração do Modelo de Factos (Log)

```
1 ==========================================================
2 ==                  VALIDATION SOFTWARE                 ==
3 ==========================================================
4 #1 - Generating Fact Model (JAR)
5         Fact Model generated:
6         > 46 entities found!
7         > JAR exported to: "/etc/validator/factModel.JAR"
```

9.2.3 Construir Regras RV

Para a construcção das regras RV, utilizaremos o Drools Guvnor, por ser aquele que oferece maior suporte e validação na construção destas mesmas regras[2].

O primeiro passo a efectuar é o *upload* do JAR gerado contendo o Modelo de Factos. Para tal devemos proceder da seguinte forma:

1. Abrir o Guvnor num *browser* (e.g.: *http://localhost:8080/guvnor/*);
2. Ir à listagem das '*Knowledge Bases*' e carregar em: '**Upload POJO Model JAR**' – Figura 9.1;
3. Configurar detalhes do novo arquivo a ser criado – Figura 9.2;
4. Seleccionar o ficheiro JAR gerado pelo validador e, carregar em '**Upload**' – Figura 9.3;

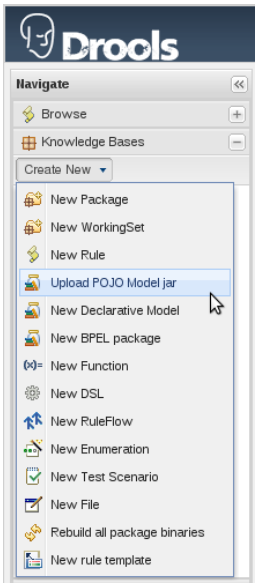

Figura 9.1: Upload POJO Model.

Após isto, podemos confirmar o carregamento do nosso modelo, bem como consultar o seu conteúdo (Figura 9.4).

Passamos então para a escrita das regras RV, não esquecendo de manter o formato já sugerido neste documento:

[2]Para alternativas consultar o Capítulo 8.2.1 - Construir Regras de Validação.

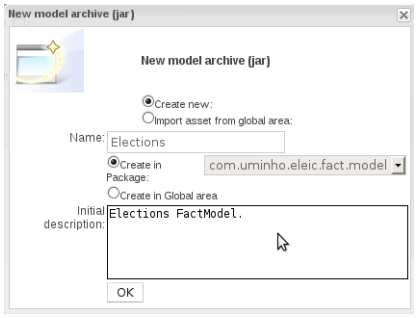

Figura 9.2: New Model Archive.

Figura 9.3: Selecionar ficheiro .JAR.

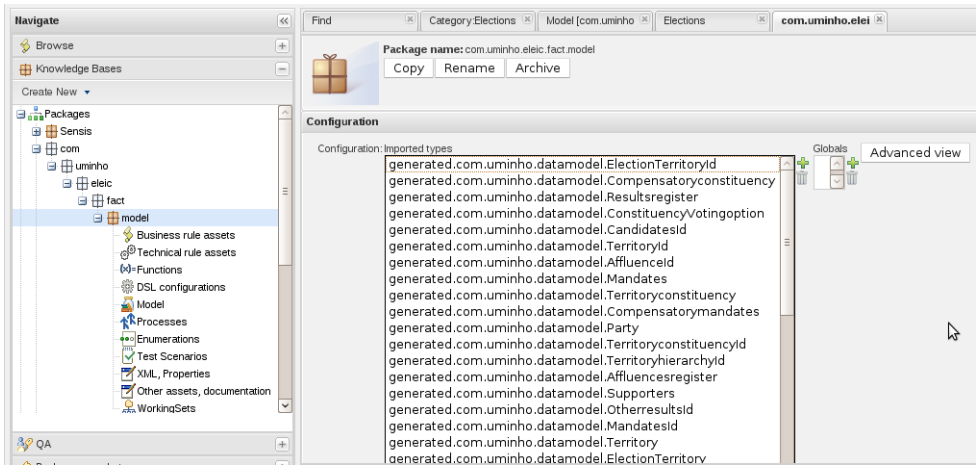

Figura 9.4: Listagem das entidades do Repositório de Factos.

When `X` $\Rightarrow$ `Y`

As regras que pretendemos validar serão primeiro apresentadas em linguagem natural, para uma maior facilidade de leitura e compreensão, e posteriormente utilizando quer o formato gráfico do Guvnor, quer o modo textual num dialecto técnico suportado pelo Validador.

De modo a sermos o mais abrangentes possíveis nas regras de validação escolhidas, tivemos em consideração os diferentes tipos de erros estudados no Capítulo 6.3 de modo a apresentarmos pelo menos uma regra para cada tipo de erro definido. Assim sendo, apresentam-se em seguida, indexadas por *tipo de erro*, algumas das regras de validação possíveis:

- Completude de Entidades:
  1. Todos os territórios têm de ter pelo menos 1 opção de voto;
  2. Todos os territórios que não são de inserção devem ter a si associados pelo menos 1 território no nível inferior;
- Coerência de Dados:
  1. No contexto da comparação de resultados actuais com os anteriores, um território actual (identificado pelo *id* e *região*), caso exista na lista dos territórios anteriores, deverá ter a mesma descrição.
- Integridade de Dados:
  1. Não podem existir territórios com nomes vazios (“ ”).
  2. Só pode existir um território *ROOT*, de nível 0 (zero) e com região `GLOBAL`.
  3. Não podem existir territórios com número de inscritos de referência igual a 0 (zero).
  4. Para a Assembleia de Freguesia, todo os círculos eleitorais deverão estar definidos no nível 4 (ao nível da freguesia).
- Integridade Relacional:
  1. Um território não pode ter um território de nível superior (ou seja um *pai*) que seja igual a si mesmo.

9.2.4 Transcrever Regras para o Validador

Após identificadas as regras, interessa-nos agora transcreve-las para uma linguagem suportada pelo Validador e, respeitando o formato já indicado (Se houver necessidade relembrar o procedimento a seguir para a transformação de cada uma das regras, deve consultar-se a Secção 8.2.1).

Apresentaremos, a título de exemplo, a introdução de uma regra RV utilizando o Guvnor. Para tal devemos proceder da seguinte forma:

1. Seleccionar a opção *Knowledge Bases* no menu principal (à esquerda) do Guvnor; escolher então a opção "*Create New*" e seleccionar: ***New Rule*** – Figura 9.5;

2. Introduzir nome, descrição e categoria (e.g.: *Elections*) da regra; bem como o formato que iremos utilizar para escrever a regra (e.g.: *Guided Editor*, *Text Editor*, *Spreadsheet* ou outro) – Figura 9.6;

3. Abrir o *formulário* para escrita da regra no formato previamente seleccionado – Figura 9.7; Escrever a regra nesse formato seleccionado [3].

Após a introdução da regra, de acordo com os passos acima, teremos algo semelhante ao visível na Figura 9.8.

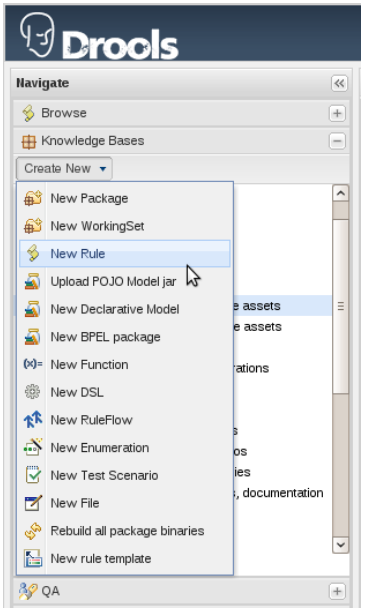

Figura 9.5: Criar Nova Regra.

Se optarmos pela escrita de todas as regras utilizando o Guvnor como suporte, poderemos depois exportar o código das regras definidas (Figura 9.9) gerando o ficheiro de extensão **.drl**.

O resultado da escrita das regras RV, enumeradas na Secção 9.2.3 acima, utilizando o dialecto mvel pode ser encontrado no Apêndice C, o qual reflecte o conteúdo do ficheiro **Rules.drl** que deverá ser colocado em local previamente definido pelo ficheiro validator.conf (Listagem 9.4) para que essas mesmas regras sejam importadas para o *Repositório de Regras*, para posterior validação por parte do Validador.

[3] Para mais detalhes de como o fazer consultar Secção 8.2.1.

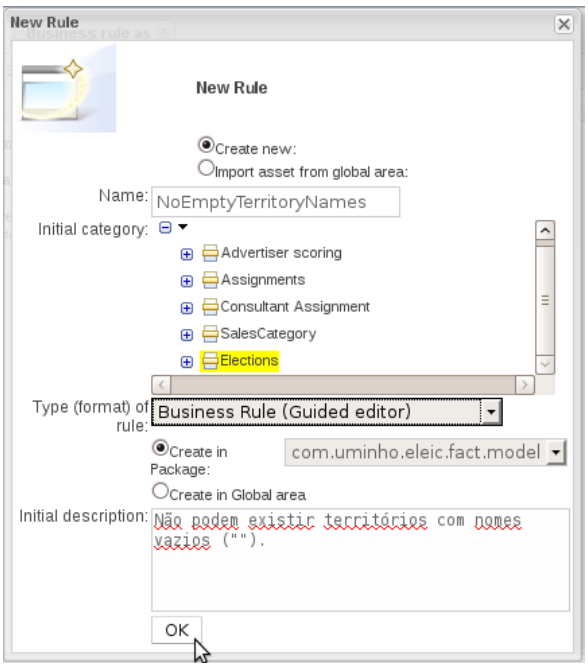

Figura 9.6: Introduzir dados para nova regra.

Figura 9.7: Formulário para introdução da nova regra.

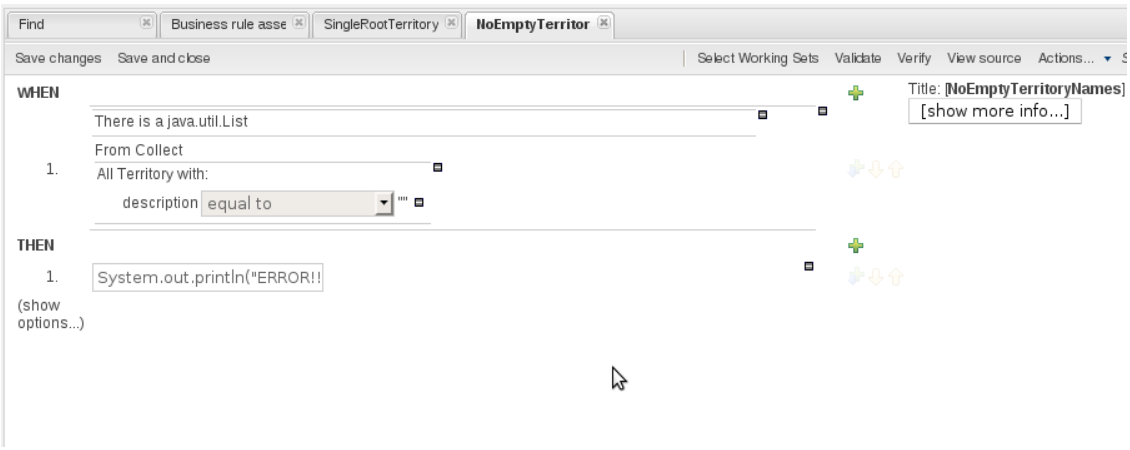

Figura 9.8: Nova regra finalizada.

Figura 9.9: Código fonte de uma regra (dialecto: `mvel`).

9.2.5 Ordenar Validação

Após a definição das regras `RV`, falta-nos unicamente ordenar ao nosso sistema Validador que efectue a validação da Plataforma Eleitoral para as regras definidas. Assim que tal ordem seja dada, o sistema encarregar-se-á de:

- adicionar as regras definidas no ficheiro de extensão **.drl** para o Repositório de Regras;
- capturar todos os factos existentes na aplicação para o Repositório de Factos;
- efectuar a **validação** das regras introduzidas para os factos capturados.

9.3 Análise do Resultado

Ao longo da execução do Validador são fornecidas, como *output* da ferramenta, diversas informações na consola da aplicação.

Entre tais informações, encontramos o número de factos recolhidos para cada `POJO` (Listagem 9.7), bem como o seu total. Mais ainda, é-nos transmitido quais os nomes das regras carregadas para o repositório (Listagem 9.8) e, após o processo de validação é-nos indicado quais as regras que se encontram validadas e satisfeitas (Listagem 9.9), bem como as regras que se encontram inválidas (Listagem 9.10), para que possamos averiguar se é um problema da regra definida ou, um erro nas configurações.

Listagem 9.7: Factos Capturados (Log)

```
#2 - Captured FACTS:
-------------------------------------------------
        ENTITY                          #Facts
> Affluence                             0
> AffluencesRegister                    0
> AffluencesStatus                      674
> BoycottedTerritories                  0
> BoycottedTerritoriesRegister          0
> Candidates                            1321
> Constituency                          4734
> Constituency_VotingOption             287
> Election                              3
> Election_Territory                    4631
> ElectoralAct                          1
> Options                               16
> Territory                             4590
> TerritoryConstituency                 4628
> VotingOption                          16
(...)
-------------------------------------------------
## TOTAL FACTS :                        32840
```

Listagem 9.8: Regras a Validar (Log)

```
#3 - Loaded Rules:
-------------------------------------------------
- constituencyLevelForCM
- singleRootTerritory
- checkTerritoryParentsExistence
- constituencyLevelForAM
- constituencyLevelForAF
- parentTerritoryMustBeDifferentFromItself
```

```
- zeroReferenceSubscribers
(...)
-----------------------------------------------------
## TOTAL RULES LOADED :                    10
```

Listagem 9.9: Regras Validadas e Satisfeitas (Log)

```
#4 - Rules Validated (and satisfied):
-----------------------------------------------------
- checkTerritoryParentsExistence
- constituencyLevelForAM
- constituencyLevelForAF
(...)
```

Listagem 9.10: Regras Falhadas e Detalhe (Log)

```
#5 - Failed Rules (Activated!)
-----------------------------------------------------
Activation created --> constituencyLevelForCM
Activation created --> zeroReferenceSubscribers
Activation created --> parentTerritoryMustBeDifferentFromItself
Activation created --> singleRootTerritory
(...)

DETAILED ERRORS:
-----------------------------------------------------
> There are: 2 ROOT Territories:
      - GLOBAL-990000
      - GLOBAL-030840
> There are: 1 Territories with Illegal Parent:
      - LOCAL-130000
> There are: 6 Territories with no Reference Subscribers:
      - GLOBAL-990000
      - LOCAL-500000
      - LOCAL-030000
      - LOCAL-130000
      - LOCAL-031200
      - GLOBAL-030840
> CM constituency shall be located at level 3!
      - instead is located at: 4
(...)
```

No fim dos elementos de avaliação produzidos, cabe ao utilizador fazer a sua interpretação, avaliar o significado dos resultados e, se necessário, corrigir as configurações e voltar a executar o Validador.

Capítulo 10

Conclusão e Trabalho Futuro

"If a man will begin with certainties, he shall end in doubts; but if he will be content to begin with doubts he shall end in certainties."

Francis Bacon

"Construir software altamente configurável, garantindo que os requisitos são cumpridos e que as configurações que os exprimem definem o comportamento esperado" é o lema deste trabalho, conforme formulado no Capítulo 1.

Neste trabalho de mestrado o principal objectivo era o de colmatar uma falha relacionada com certificação de software altamente configurável; falha essa introduzida por mecanismos de abstracção, com particular ênfase para mecanismos que definem objectivos de negócio em tempo de execução através de configurações; ou seja, pretende-se garantir a ausência de erros nas configurações de negócio através de validação dessas mesmas configurações.

Para atingir o referido objectivo decidiu-se seguir por uma abordagem alternativa aos tradicionais métodos formais ou mesmo testes aplicacionais. O trabalho desenvolvido centrou-se nas configurações em vez de se centrar no código aplicacional, assumindo-se à partida que o software base se encontra correcto. Dessa forma, toda a atenção cai sobre as configurações empregues e sobre o cumprimento (ou não) dos requisitos de negócio que essas mesmas configurações definem.

Assim sendo, no Capítulo 2 começou-se por analisar a origem do problema e por estudar quais as necessidades e *sub-problemas* que tinham de ser resolvidos para concretizar os objectivos estabelecidos, conseguindo-se resumir todo este trabalho de mestrado numa única questão:

> *Como validar que as configurações, que definem regras de negócio, aplicadas num sistema altamente configurável, estão de acordo com a lógica pretendida pelo cliente/utilizador, aumentando a confiança nessas configurações, sem que para tal se perca flexibilidade aplicacional?*

No Capítulo 3 explicita-se, o melhor possível, a dicotomia entre **verificação** e **validação** bem como as técnicas e ferramentas utilizadas por cada um dos processos. Percebeu-se então que são processos que se complementam. Apesar das suas ferramentas e técnicas estarem cada vez mais evoluídas, no caso dos sistemas de software altamente configurável, existem ainda algumas limitações. São essas limitações que a proposta, referida neste trabalho, pretende resolver através do desenvolvimento de uma ferramenta *auxiliar* de **validação**, a qual surge devido à evolução no modo como se tem vindo a desenvolver software.

Identificada concretamente a área de acção na qual o nosso problema se enquadra — validação — estudámos no Capítulo 4 as tecnologias e metodologias existentes para validar software altamente configurável. Tipicamente é efectuada uma validação isolada das configurações, após realização de uma selecção das mesmas devido ao elevado número de possibilidades; são abordagens que se regem essencialmente pelo teste das configurações seleccionadas e que podem não incluir as configurações que realmente nos interessam. Como tal não respondem por completo ao desafio a que nos propusemos nesta tese de mestrado — validação das configurações que serão usadas pela aplicação em determinada instância num determinado momento, ou seja validadas num contexto de execução real específico.

Para que seja possível o desenvolvimento de uma solução adaptável e modular, é imprescindível conhecer as técnicas existentes para realizar variabilidade numa arquitectura, bem como conhecer os diferentes modos utilizados para a especificação de configurações — estudo esse que foi realizado no Capítulo 5 e o que nos fez perceber que são inúmeras as formas de se especificarem configurações e que podem variar consoante a arquitectura, a linguagem de desenvolvimento ou mesmo dos objectivos de negócio da aplicação em questão.

Após a análise do problema e identificação clara de quais as questões a que é necessário dar resposta bem como qual o caminho a seguir para a sua concretização, apresentámos no Capítulo 6 como são compostas as configurações, e em função disso que tipo de problemas podem ser introduzidos nas aplicações; posteriormente apresentámos uma serie de classes de severidade de erros a que as Regras de Validação (`RV`) podem ser associadas e, finalmente desvendámos a estratégia de validação a ser usada para dar resposta ao problema central deste trabalho de mestrado.

Tendo em conta que existem inúmeras formas de se tornar uma aplicação configurável e que as configurações podem ser definidas de variadas formas, fica claro que a estratégia de validação a ser implementada no Validador deverá assegurar compatibilidade com o maior número possível de aplicações configuráveis. Para tal a nossa estratégia assenta em três etapas distintas, definidas o mais genericamente possível, que revelam claramente quais as nossas intenções:

1. definir e representar as regras `RV` numa determinada linguagem;
2. interpretar as regras definidas no ponto anterior;
3. executar a validação das regras no contexto da aplicação altamente configurável.

Definida a estratégia a usar, apresentámos no Capítulo 7 a arquitectura da solução, especificando claramente quais os limites do sistema a desenvolver bem como as funcionalidades necessárias para a execução do processo de validação.

O objectivo é evidentemente analisar as etapas da nossa estratégia de validação, identificando as funcionalidades necessárias e dependências existentes no fluxo de informação do processo que lhe está associado; de modo a agrupar as responsabilidades interligadas em elementos estruturais que representam os componentes da nossa arquitectura.

Como resultado obtivemos uma arquitectura (ver Figura 7.9) composta por:

- quatro componentes *core*: Integrator, Rules Engine, Rules Manager e Rules Processor;
- dois componentes para persistência de dados: Facts Repository e Knowledge Repository;

Examinando com detalhe o desenho desta arquitectura constatámos que apresenta algumas semelhanças com a arquitectura de sistemas de Business Rules Management System (BRMS) — os quais foram abordados no Capítulo 5.1.

Após o desenho da arquitectura, avançámos para o desenvolvimento do sistema validador, cujos detalhes de implementação podem ser encontrados no Capítulo 8, e onde demonstrámos a viabilidade da estratégia definida através da realização de uma Prova de Conceito (POC) sustentada pela arquitectura já apresentada.

Na abordagem ao desenvolvimento do sistema validador tomou-se como ponto de partida a utilização do **Drools** como ferramenta de suporte fundamental à concretização desta implementação. Este apesar de aplicado normalmente com outros propósitos[1], surge aqui fornecendo uma série de bibliotecas e utilitários, os quais são adaptados, consoante as necessidades, aos diferentes componentes da arquitectura. Para além desta ferramenta, também o Hibernate Tools e o Maven foram utilizados, em concreto no componente Integrador.

Findo o desenvolvimento do sistema validador, e relembrando os propósitos que o mesmo pretende atingir, interessa referir que:

- o componente Integrador foi sem sombra de dúvidas aquele que colocou maiores desafios; principalmente por ter a responsabilidade de identificar as configurações da aplicação altamente configurável, e também de as capturar para poderem ser usadas como `factos` para validação;
- cada facto recolhido pelo Integrador, independentemente da aplicação a que diz respeito, é nada mais do que uma instância da classe java.lang.Object, suportada pelo Drools, e que respeita o Modelo de Factos gerado para a aplicação sob análise;
- o Modelo de Factos (gerado automaticamente pelo Integrador) é um *package* de entidades `Java` que permitem representar inequivocamente a informação de uma qualquer aplicação configurável (factos), sendo imprescindível para a escrita das regras — as quais devem, obrigatoriamente, seguir o modelo.

[1] Drools: É utilizado normalmente em sistemas cuja lógica de negócio surge como um conjunto de regras, as quais adaptam comportamentos e lógica sem a necessidade de alterar código aplicacional

- os restantes componentes que compõem o Validador, como o `Gestor de Regras`, o `Motor de Regras` e ainda o `Processador de Regras`, foram adaptados a partir da `API` do Drools de modo a servir as funcionalidades pretendidas;

- as regras `RV` podem ser escritas com recurso a diferentes mecanismos, no entanto, nenhum dos mecanismos impõe o formato da mensagem definido na secção 8.2.1 — pelo que terá de ser o utilizador a garantir o cumprimento desse mesmo formato, para assegurar o bom funcionamento do Validador;

- é o Algoritmo de Rete, utilizado internamente pelo Drools, que está encarregue de fazer *match* entre as regras e os factos existentes em cada um dos respectivos repositórios.

No Capítulo 9 colocámos à prova a estratégia de validação desenvolvida, utilizando a solução implementada no contexto da `Plataforma Eleitoral`, que é caso de estudo neste trabalho de mestrado.

A preocupação passou por se testarem os diversos componentes que fazem parte do nosso sistema Validador; desde a fase de geração do modelo de factos e da escrita das regras `RV`, passando pelo preenchimento dos repositórios de regras e de factos até à fase em que se despoleta a validação das regras face aos factos existentes no repositório.

Para tal, utilizaram-se alguns exemplos não muito extensos, mas que permitiram testar a robustez da estratégia e do processo desenvolvido de forma mais realistica possível — a `Plataforma Eleitoral` foi configurada com umas eleições Autárquicas[2], tendo sido escritas algumas regras `RV` em concordância com o Modelo de Factos, gerado a partir da base de dados `PostgreSQL` utilizada pela aplicação. A partir daí foram capturados os factos da aplicação, sendo despoletado o processo de validação para as regras `RV` escritas. Nesta fase houve a preocupação de se testar os diferentes tipos de resposta esperados por parte do Validador desenvolvido: todas as regras a falharem a validação; seguido de apenas algumas regras a falharem a validação e finalmente, com todas as regras a passarem a validação (após a correcção das configurações utilizadas).

Com isto, foi possível ilustrar que a solução desenvolvida neste trabalho de mestrado permite validar as configurações (face a um conjunto de regras `RV` definidas) e também fazê-lo por um processo automatizado e reutilizável — permitindo que para o mesmo Modelo de Factos, se possa fazer variar tanto o Repositório de Regras como o Repositório de Factos consoante desejado. Contribui-se desta forma para a validação do comportamento do software resultante da aplicação de configurações, e para o aumento da certificação de software.

Analisando a **relação custo-benefício** da solução implementada, pode-se dizer que é evidentemente favorável dado que, aquilo que outrora teria de ser executado manualmente por um utilizador com experiência e conhecimentos de negócio da

[2]Notar que os dados utilizados são fictícios não só por uma questão de simplicidade, mas também para maior facilidade em abranger diferentes situações

aplicação sobre validação, pode agora ser efectuado com recurso ao nosso Validador; obviamente que, o *revés da medalha* é que há a necessidade de se efectuar uma ligeira configuração para conseguirmos gerar o Modelo de Factos, bem como proceder à escrita das regras `RV`; no entanto, uma vez escritas as regras, estas poderão ser usadas repetidamente em contextos semelhantes sempre que necessário, e com um esforço reduzido.

Em suma, respondendo à questão formulada que resume todo este trabalho de mestrado, podemos referir que para validar configurações de um sistema altamente configurável há a necessidade de se gerar um Modelo de Factos, o qual define a estrutura que será usada como base para suportar os factos recolhidos dessa mesma aplicação e que serão alvo de validação; bem como a necessidade de se escreverem um conjunto de regras de negócio , que definem o negócio do sistema a validar. Cada uma das regras `RV` definidas será posteriormente validada face aos factos capturados, utilizando um Motor de Regras. Como resultado dessa validação obtemos um relatório detalhado com uma listagem das regras que foram executadas, quais as que foram aprovadas e quais as que falharam. Todo este processo encontra-se dissociado do sistema altamente configurável sobre validação e como tal a flexibilidade aplicacional é mantida, mas permite-nos aumentar a confiança nas configurações na medida em que podemos verificar tantas regras quantas quisermos.

Do trabalho desenvolvido no Capítulo 9, surgem alguns aspectos que se identificaram como eventuais pontos de melhoria para o nosso Validador, os quais são abordados na secção seguinte.

10.1 Trabalho Futuro

Os objectivos a que nos proposemos no Capítulo 1 desta dissertação foram atingidos. No entanto existem aspectos que, apesar de não terem sido explicitamente abordados no decorrer deste trabalho, levantam questões que podem ser tomadas como futuras linhas de investigação; tais questões são identificadas e discutidas em seguida.

Um dos componentes principais da arquitectura do nosso Validador — o **Integrador** — foi construído a pensar na necessidade futura de expansão do mesmo. Ora, uma das principais responsabilidades deste componente, é a **geração do Modelo de Factos**, a qual é actualmente efectuada com base num modelo de dados (base de dados relacional), através da geração das entidades desse modelo, e que representam na sua essência o modelo de domínio da aplicação que pretendemos validar. No entanto, nem todas as aplicações têm o seu domínio mapeado em modelos de dados relacionais, e como tal a existência de uma forma mais abrangente para definição do Modelo de Factos a partir de outros suportes, como por exemplo a partir de estruturas em memória, seria uma mais valia para o Validador.

Da mesma forma que a geração do Modelo de Factos foi implementado para funcionar com aplicações cujo modelo de dados se baseia numa base de dados rela-

cional, também a forma como se identifica e capturam os factos (que são colocados no Repositório de Factos e que representam a informação que será validada) foi implementada para factos que se encontram num modelo de dados relacional. Seria portanto interessante, dotar o módulo Integrador da capacidade de **capturar factos**, não só a partir de uma base de dados relacional, mas também a partir da memória interna da aplicação, de ficheiro, ou de outro suporte qualquer, aumentando dessa forma o conjunto de aplicações que podem ser validadas utilizando o nosso sistema Validador.

No que diz respeito às **Regras** `RV`, que são actualmente escritas recorrendo às linguagens `Java` ou `mvel`, e interpretadas posteriormente pelo Drools, seria interessante a **criação de uma linguagem formal, mais próxima da linguagem natural**. Tais regras continuariam a cumprir a especificação definida pelo Modelo de Factos e o formato fixado no Capítulo 8, no entanto a sua criação seria uma actividade mais simples e de leitura e compreensão simplificada.

Bibliografia

[ABD+04] Alain Abran, Pierre Bourque, Robert Dupuis, James W. Moore, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004 version edition, 2004.

[AMC+03] Deepak Alur, Dan Malks, John Crupi, Grady Booch, and Martin Fowler. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Sun Microsystems, Inc., Mountain View, CA, USA, 2003.

[ASF12] The Apache Software Foundation ASF. Licenses. `http://www.apache.org/licenses/`, 2012.

[ASM03] Timo Asikainen, Timo Soininen, and Tomi Männistö. A koala-based approach for modelling and deploying configurable software product families. In *PFE*, pages 225–249, 2003.

[B+11] Mike Brock et al. Mvel — home. `http://mvel.codehaus.org/`, 2011. Online; Acedido em: 17-Maio-2011.

[Bal09] Michal Bali. *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing, July 2009.

[BB08] David G. Bell and Guillaume P. Brat. Automated software verification & validation: An emerging approach for ground operations. *2008 IEEE Aerospace Conference*, pages 1–8, 2008.

[BKS03] Evelyn J. Barry, Chris F. Kemerer, and Sandra A. Slaughter. On the uniformity of software evolution patterns. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 106–113, Washington, DC, USA, 2003. IEEE Computer Society.

[BPS03] André Baresel, Hartmut Pohlheim, and Sadegh Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In *GECCO'03: Proceedings of the 2003 international conference on Genetic and evolutionary computation*, pages 2428–2441, Berlin, Heidelberg, 2003. Springer-Verlag.

[Bro95] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995.

[Bro09] Paul Browne. *JBoss Drools Business Rules*. Packt Publishing, April 2009.

[CDS07] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *ISSTA '07: Proceedings of the 2007 international symposium on Software testing and analysis*, pages 129–139, New York, NY, USA, 2007. ACM.

[CDS08] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, 34:633–650, 2008.

[CG09] Andrea Calvagna and Angelo Gargantini. Ipo-s: Incremental generation of combinatorial interaction test data based on symmetries of covering arrays. *Software Testing Verification and Validation Workshop, IEEE International Conference on*, 0:10–18, 2009.

[CGT09] Andrea Calvagna, Angelo Gargantini, and Emiliano Tramontana. Building t-wise combinatorial interaction test suites by means of grid computing. In *WETICE '09: Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 213–218, Washington, DC, USA, 2009. IEEE Computer Society.

[CSU11] Computer Science Department Colorado State University. Basic vi commands. `http://www.cs.colostate.edu/helpdocs/vi.html`, 2011. Online; Acedido em: 16-Maio-2011.

[CSW10] Critical Software S.A CSW. cswow. `http://www.criticalsoftware.com/products_services/cswow/`, 2010. Online; Acedido em: 18-Junho-2010.

[dAID10a] Direcção Geral da Administração Interna DGAI. Autarquias locais - legislação e manuais. `http://www.dgai.mai.gov.pt/?area=103&mid=030&sid=032`, 2010. Online; Acedido em: 26-Dezembro-2010.

[dAID10b] Direcção Geral da Administração Interna DGAI. Autárquicas 2009. `http://www.legislativas2009.mj.pt/autarquicas2009/`, 2010. Online; Acedido em: 30-Maio-2010.

[dAID10c] Direcção Geral da Administração Interna DGAI. Europeias 2009. `http://www.legislativas2009.mj.pt/Europeias2009/`, 2010. Online; Acedido em: 30-Maio-2010.

[dAID10d] Direcção Geral da Administração Interna DGAI. Legislativas 2009. `http://www.legislativas2009.mj.pt/legislativas2009/`, 2010. Online; Acedido em: 30-Maio-2010.

[Das07] Aristides Dasso. *Verification, Validation and Testing in Software Engineering*. IGI Publishing, Hershey, PA, USA, 2007.

[DBC88] A. M. Davis, H. Bersoff, and E. R. Comer. A strategy for comparing alternative software development life cycle models. *IEEE Trans. Softw. Eng.*, 14(10):1453–1461, 1988.

[Dic10] Dictionary.com, "workaround"in collins english dictionary - complete & unabridged 10th edition. Aug 2010. Online; Acedido em: 21-Agosto-2010.

[Dic11] Dictionary.com, "proof of concept"in collins english dictionary - complete & unabridged 10th edition. `http://dictionary.reference.com/browse/proofofconcept`, Jan 2011. Online; Acedido em: 3-Janeiro-2011.

[Dij68] Edsger W. Dijkstra. The structure of the "the"-multiprogramming system. *Commun. ACM*, 11(5):341–346, 1968.

[DMM07] Mouhamed Diouf, Sofian Maabout, and Kaninda Musumbu. Merging model driven architecture and semantic web for business rules generation. In *RR'07: Proceedings of the 1st international conference on Web reasoning and rule systems*, pages 118–132, Berlin, Heidelberg, 2007. Springer-Verlag.

[dO82] A. J. Francisco de Oliveira. *Teoria de Conjuntos*. Livraria Escolar Editora, Lisboa, Portugal, 1982.

[Don08] P. Donohoe. Introduction to software product lines. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 370 –370, 8-12 2008.

[Dru02] Peter F. Drucker. *The Effective Executive Revised*. HarperBusiness, September 2002.

[Eas08] Steve Easterbrook. Lecture 15: Verification and validation. Department of Computer Science - University of Toronto, 2008.

[Ei90] Institute O. Electrical and Electronics E. (ieee). *IEEE 90: IEEE Standard Glossary of Software Engineering Terminology*. 1990.

[Fej91] Peter A. Fejer. *Mathematical Foundations of Computer Science (Graduate Texts in Mathematics) (v. 1)*. Springer, 1991.

[For79] Charles Lanny Forgy. *On the efficient implementation of production systems*. PhD thesis, Pittsburgh, PA, USA, 1979. AAI7919143.

[For82] Charles Lanny Forgy. Rete: A fast algorithm for the many pattern/-many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[Fou11] The Eclipse Foundation. Eclipse — The Eclipse Foundation open source community website. `http://www.eclipse.org/`, 2011. Online; Acedido em: 16-Maio-2011.

[Fou12] Apache Software Foundation. Apache maven project. `http://maven.apache.org/`, 2012. Online; Acedido em: 9-Maio-2012.

[Fri10] Ernest Friedman-Hill. Jess, the rule engine for the java platform oss-rules. `http://www.jessrules.com/jess/index.shtml`, 2010. Online; Acedido em: 17-Junho-2010.

[Gao10] Yu Gao. Research on the rule of evolution of software development process model. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*, pages 466 –470, april 2010.

[GHJ+93] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Taligent Inc. Design patterns: Abstraction and reuse of object-oriented design. pages 406–431. Springer-Verlag, 1993.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[Gil11] Thomer M. Gil. Vi lovers homepage. `http://thomer.com/vi/vi.html`, 2011. Online; Acedido em: 16-Maio-2011.

[GV05] Luciano Rodrigues Guimarães and Plínio Roberto Souza Vilela. Comparing software development models using cdm. In *SIGITE '05: Proceedings of the 6th conference on Information technology education*, pages 339–347, New York, NY, USA, 2005. ACM.

[Hal74] Paul R Halmos. *Naive Set Theory*, volume 68. Springer, 1974.

[HCL07] Pao-Ann Hsiung, Yean-Ru Chen, and Yen-Hung Lin. Model checking safety-critical systems using safecharts. *IEEE Trans. Comput.*, 56(5):692–705, 2007.

[HCWC09a] Xianzhi Huang, Lizhen Cui, Haiyang Wang, and Wenjing Cui. Towards a decentralized cooperative brms for service collaborative enterprise. In *CSCWD '09: Proceedings of the 2009 13th International Conference on Computer Supported Cooperative Work in Design*, pages 66–71, Washington, DC, USA, 2009. IEEE Computer Society.

[HCWC09b] Xianzhi Huang, Lizhen Cui, Haiyang Wang, and Wenjing Cui. Towards a decentralized cooperative brms for service collaborative enterprise. In *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on*, pages 66 –71, 22-24 2009.

[Ho11] Don Ho. Notepad++. `http://notepad-plus-plus.org/`, 2011. Online; Acedido em: 16-Maio-2011.

[II04] ISO and IEC. *ISO/IEC 90003:2004, Software and Systems Engineering - Guidelines for the Application of ISO9001:2000 to Computer Software*. 2004.

[JBo11] JBoss.org. Drools Guvnor — JBoss Community. `http://www.jboss.org/drools/drools-guvnor.html`, 2011. Online; Acedido em: 16-Maio-2011.

[JBo12a] JBoss.org. Drools — The Business Logic integration Platform. `http://www.jboss.org/drools`, 2012. Online; Acedido em: 30-Março-2012.

[JBo12b] JBoss.org. Hibernate tools for eclipse and ant. `http://www.hibernate.org/subprojects/tools.html`, 2012. Online; Acedido em: 9-Maio-2012.

[KMS03] Tero Kojo, Tomi Männistö, and Timo Soininen. Towards intelligent support for managing evolution of configurable software product families. In *SCM*, pages 86–101, 2003.

[KN96] R.J. Kreutzfeld and R.E. Neese. A methodology for cost-effective software fault tolerance for mission-critical systems. pages 19 –24, oct 1996.

[Kni02] J.C. Knight. Safety critical systems: challenges and directions. pages 547 – 550, 2002.

[Kru00] C.W. Krueger. Software product line reuse in practice. In *Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on*, pages 117 –118, 2000.

[LSR07] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[LTL+08] Shimin Li, Ladan Tahvildari, Weining Liu, Mike Morrissey, and Gary Cort. Coping with requirements changes in software verification and validation. In *CSMR '08: Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, pages 317–318, Washington, DC, USA, 2008. IEEE Computer Society.

[LYL09] Yuqing Lin, Huilin Ye, and Bojun Li. A new parameter for product configuration in software product lines. In *Knowledge Acquisition and Modeling, 2009. KAM '09. Second International Symposium on*, volume 2, pages 230 –233, nov. 2009.

[MD08] Tom Mens and Serge Demeyer, editors. *Software Evolution*. Springer, 2008.

[MFRP06] Nazim H. Madhavji, Juan Fernandez-Ramil, and Dewayne Perry. *Software Evolution and Feedback: Theory and Practice.* John Wiley & Sons, 2006.

[Nea10] Michael Neale. JbossRules - Jboss Comunity. `http://community.jboss.org/wiki/jbossrules`, 2010. Online; Acedido em: 17-Junho-2010.

[NK02] Pat Niemeyer and Jonathan Knudsen. *Learning Java.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[NN09] Marius Nita and David Notkin. White-box approaches for improved testing and analysis of configurable software systems. In *ICSE Companion*, pages 307–310. IEEE, 2009.

[Oli08] Paulo Jorge Machado Oliveira. *Detecção e correcção de problemas de qualidade dos dados: modelo, sintaxe e semântica.* PhD thesis, Universidade do Minho, Braga, Portugal, 2008.

[OMG12] OMG. Business Process Model and Notation. `http://www.bpmn.org`, 2012. Online; Acedido em: 30-Março-2012.

[Ora11] Oracle.com. Oracle technology network for java developers. `http://www.oracle.com/technetwork/java/index.html`, 2011. Online; Acedido em: 4-Maio-2011.

[Par76] D.L. Parnas. On the design and development of program families. *Software Engineering, IEEE Transactions on*, SE-2(1):1 – 9, march 1976.

[PS08] H. Post and C. Sinz. Configuration lifting: Verification meets software configuration. *Automated Software Engineering, International Conference on*, 0:347–350, 2008.

[PST10] Production Systems Technologies PST. OPSJ: Rules for Java. `http://www.pst.com/`, 2010. Online; Acedido em: 17-Junho-2010.

[Qu09] Xiao Qu. Configuration aware prioritization techniques in regression testing. In *ICSE Companion*, pages 375–378, 2009.

[Rak01] Steven R. Rakitin. *Software Verification and Validation for Practitioners and Managers, Second Edition.* Artech House, Inc., Norwood, MA, USA, 2001.

[Roy70] W. Royce. Managing the development of large software systems. In *Proc. IEEE Wescon*, pages 1–9, August 1970.

[RS99] P. Richardson and S. Sarkar. Adaptive scheduling: overload scheduling for mission critical systems. pages 14 –23, 1999.

[RSM+] Elnatan Reisner, Charles Song, Kin-Keung Ma, Jeffrey S. Foster, and Adam Porter. Evaluating interaction patterns in configurable software systems. Technical report, Computer Science Department, University of Maryland.

[RSMM03] Mikko Raatikainen, Timo Soininen, Tomi Männistö, and Antti Mattila. A case study of two configurable software product families. In *PFE*, pages 403–421, 2003.

[Sch07] Herbert Schildt. *Java: A Beginner's Guide, 4th Ed.* McGraw-Hill, Inc., New York, NY, USA, 2007.

[SM10] Software Engineering Institute (SEI) and Carnegie Mellon. Software product lines. `http://www.sei.cmu.edu/productlines/`, 2010.

[SS09] H. Schirmeier and O. Spinczyk. Challenges in software product line composition. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1 –7, 5-8 2009.

[SZ07] Zhang Sen and Yao Zheng. The relation of cmm and software lifecycle model. In *SNPD '07: Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 864–869, Washington, DC, USA, 2007. IEEE Computer Society.

[TR07] James Taylor and Neil Raden. *Smart Enough Systems: How to Deliver Competitive Advantage by Automating Hidden Decisions.* Prentice Hall PTR, July 2007.

[Tra99] Eushiuan Tran. Verification/Validation/Certification. Technical report, Carnegie Mellon University, 1999.

[Tra01] I. Traore. An integrated v v environment for critical systems development. pages 287 –287, 2001.

[Tur96] Russell Turpin. A progressive software development lifecycle. *Engineering of Complex Computer Systems, IEEE International Conference on*, 0:208, 1996.

[vim11] vim.org. Vim - the editor. `http://www.vim.org/`, 2011. Online; Acedido em: 16-Maio-2011.

[VP11] Visual Paradigm VP. Drawing activity diagrams in vp-uml. `http://www.visual-paradigm.com/support/documents/vpumluserguide/94/200/6713_drawingactiv.html`, 2011. Online; Acedido em: 7-Junho-2011.

[W3C] W3C. Extensible markup language (xml). `http://www.w3.org/XML/`.

[WF89] Dolores R. Wallace and Roger U. Fujii. Software verification and validation: An overview. *IEEE Software*, 6:10–17, 1989.

[Wit09] P.D. Witman. Software product lines and configurable product bases in business applications - a case from financial services. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1 –10, 5-8 2009.

[YMC01] Zan Yang, Byeong Min, and Gwan Choi. Simulation using code-perturbation: black- and white-box approach. In *Quality Electronic Design, 2001 International Symposium on*, pages 44 –49, 2001.

Apêndice A

Regras de Validação para o Caso de Estudo

Neste apêndice são apresentadas um conjunto de regras de validação (RV) originárias do caso de estudo - Plataforma Eleitoral - usado neste trabalho de mestrado. De notar que as seguintes regras de validação encontram-se escritas em linguagem natural e, dizem respeito a um conjunto de regras de configuração usadas na Plataforma Eleitoral para definir umas Eleições Autárquicas.

Antes de poderem ser validadas e utilizadas no Sistema de Validação necessitam, tal como já foi referido, de ligeiras alterações cujos detalhes podem ser encontrados nos capítulos 8 (`Desenvolvimento do Sistema de Validação`), e 9 (`Avaliação da Solução`), juntamente com alguns exemplos dessa mesma transformação.

Acto Eleitoral

- O acto eleitoral é composto por 3 eleições: AM, CM, AF;
- A eleição AF deverá ter todos os círculos eleitorais (LOCAL) definidos no nível 4 (Freguesia);
- A eleição CM deverá ter todos os círculos eleitorais (LOCAL) definidos no nível 3 (Concelho);
- A eleição AM deverá ter todos os círculos eleitorais (LOCAL) definidos no nível 3 (Concelho);
- A sequência de inserção dos votos nas diversas eleições deverá ser sequencial e pela seguinte ordem: AF $>$ AM $>$ CM.
- A inserção dos resultados do escrutínio nos Açores deverá acontecer 1H mais tarde do que no continente (devido à diferença horária de menos 1H em relação ao continente);
- Qualquer eleição configurada deverá ter pelo menos 1 círculo eleitoral criado para cálculo dos resultados.

Hierarquia de Territórios

- Só pode existir um território *ROOT*, nível 0, com região *GLOBAL*;
- Todos os territórios que não são de inserção devem ter a si associados pelo menos 1 território ("filho") no nível inferior.
- Todos os territórios devem ser alcançáveis. (Ou seja, partindo do território raiz devemos ser capazes de alcançar todos os territórios definidos).
- Nenhum território poderá ter como território no nível inferior o mesmo território associado ao nível superior (ou seja o filho de um território não pode ser igual ao seu pai).
- Um território não pode ter um território de nível inferior (ou seja um filho) que seja igual a ele próprio.
- Não podem existir territórios com nomes vazios (" ").
- Todos os territórios devem ter um número de eleitores inscritos de referência superior a zero (0);

Círculos Eleitorais

- Todos os Territórios de nível de inserção (freguesias) têm de ter 1 círculo eleitoral associado.
- Todos os Círculos Eleitorais têm de ter pelo menos 1 opção de voto.
- Todos os Territórios têm de ter pelo menos 1 opção de voto.
- O n.º de candidatos efectivos de cada partido deve ser igual ou superior ao n.º de mandatos a atribuir no Círculo Eleitoral em que concorre.
- O nome de cada candidato não pode ser vazio (" ").

Comparativos

- No contexto da comparação de resultados actuais com os anteriores, um território actual (identificado pelo *id* e *região*), caso exista na lista dos territórios anteriores, deverá ter a mesma descrição.

Afluências

- Se existirem horas de afluências definidas, então deve existir pelo menos 1 Território Tipo (Freguesia) por Concelho.
- Todos os Territórios Tipo devem ter a si associados inscritos de referência para as afluências.

Utilizadores

- Deve existir pelo menos 1 utilizador com a role "Regular" (aka Governo Civil) e com permissões de escrita associado a cada um dos territórios que são Círculos Eleitorais.
- Deve existir pelo menos 1 utilizador com a role "Manager" (aka DGAI) e com permissões de escrita com acesso total (região GLOBAL).
- As passwords de todos os utilizadores devem ter um tamanho igual ou superior a 8 caracteres.

Apêndice B

Ficheiros de Configuração - Autárquicas

Neste apêndice encontraremos todos os 14 ficheiros utilizados para configurar a Plataforma Eleitoral para umas eleições *Autárquicas*.

Os ficheiros que serão incluídos são os mesmos que foram utilizados na realização das Autárquicas em Portugal Continental e Regiões Autónomas a 11 de Outubro de 2009. Cada ficheiro estará numa secção em separado e, será acompanhado de uma breve explicação da sua estrutura e conteúdo. De notar que, no caso de ficheiros demasiado extensos, partes do ficheiro poderão ser omitidas.

B.1 configuration.xml

Neste ficheiro encontram-se definidas informações como:

- horários relativos à abertura e fecho das urnas;
- horário de disponibilização dos resultados ao público;
- dados estatísticos adicionais a serem revelados na página de resultados, como 'maiorias absolutas', 'maiorias relativas';
- níveis territoriais cujos detalhes da votação podem ser revelados na página de resultados;
- qual o algoritmo a ser utilizado para o cálculo de mandatos e resultados;
- quais os níveis territoriais sobre a qual serão recolhidos os resultados da votação e afluências;
- quais as horas definidas para início da recepção de valores de afluências e o tempo máximo para a introdução desses mesmos valores;
- configurações especificas para os relatórios a serem gerados;
- configurações de sistema específicas para o *Backoffice*;

- configurações de sistema específicas para o *Frontend*;
- informações acerca de *Grupo de Cidadãos* e o agregar dos resultados;
- data do último acto eleitoral do mesmo tipo (para efeitos de comparações de resultados);

```
<?xml version = "1.0" encoding = "UTF-8"?>
<configuration>
  <!--Data File Path -->
  <data_path>/etc/election-results-system/data/</data_path>

  <electoral_act_open_time>2009-10-11T08:00:00</electoral_act_open_time>
  <electoral_act_close_time>2009-10-11T19:00:00</electoral_act_close_time>

  <!-- Results disclosure options -->
  <results_disclosure>
    <!--
      This value shall define the hour that we start the results
      disclosure. This value his the hour relative to the server location.
      EXAMPLE: > At 20H of Portugal Continental (19h at Azores) we can
      start showing results in the web. > Azores - have just closed the
      polls > Portugal Continental & Madeira - closed 1H a go (can already
      have results)
    -->
    <hour>2009-10-11T19:00:00</hour>
    <!-- Update interval on seconds to update cache -->
    <cache_update>30</cache_update>
    <!-- results information by election -->
    <results_information>
      <election_results id='CM'>
        <presidents>true</presidents>
        <absolute_majority>true</absolute_majority>
        <relative_majority>false</relative_majority>
      </election_results>
      <election_results id='AF'>
        <presidents>true</presidents>
        <absolute_majority>false</absolute_majority>
        <relative_majority>false</relative_majority>
      </election_results>
      <election_results id='AM'>
        <presidents>false</presidents>
        <absolute_majority>false</absolute_majority>
        <relative_majority>false</relative_majority>
      </election_results>
    </results_information>
  </results_disclosure>
```

```
<!--
  Territory Levels that are allowed to be showed up in "Results
  Disclosure" - It includes: Local and Foreign report levels
-->
<report_levels is_global_report_level="true">
  <!-- (OPTIONAL) If this node is not defined, then local information will
       not be shown -->
  <local_report_levels>
    <level>1</level>
    <level>2</level>
    <level>3</level>
    <level>4</level>
  </local_report_levels>
</report_levels>

<algorithms>
  <algorithm name="Hondt">
    <!--
      Implementation class of the algorithm.
      NOTE: This class should implement interface ElectionAlgorithm
    -->
    <package>com.criticalsoftware.elections.business.hondt.HondtAlgorithm
    </package>
  </algorithm>
</algorithms>

<!-- (OPTIONAL) Default: lower local territory level -->
<local_vote_insertion_level>4</local_vote_insertion_level>
<!-- (OPTIONAL) Default: lower local territory level -->
<affluence_insertion_level>4</affluence_insertion_level>

<!--
  (OPTIONAL) If this field is not specified, affluence insertion will
  not be allowed. This configuration includes:
  insertion time: limit time in hours to insert results after defined hour.
  definitive_close_time: if true, then after all territories closed, can't
      delete values (even if its yet affluences time)
  (example: insertion_time="1.5" means 1h30min)
  hours: Affluence start date.
  NOTE: affluence only exist for "LOCAL TERRITORIES"
-->
<affluence insertion_time="00:55" definitive_close_time="true">
  <hours>
    <hour>2009-10-11T12:00:00</hour>
    <hour>2009-10-11T16:00:00</hour>
```

```
  </hours>
</affluence>

<!-- (OPTIONAL) Export current results to file -->
<export_file>
  <!-- Time interval on minutes to export file -->
  <update_interval>5</update_interval>
  <!-- Folder that will contains all exported files -->
  <all_files_folder>/var/www/html/autarquicas/dgai</all_files_folder>
  <!-- Name that will be appended to filename-->
  <filename_appender>_results</filename_appender>
  <!-- Folder that will contain last exported file -->
  <last_file_folder>/var/www/html/autarquicas/resultados</last_file_folder>
  <!-- Name of the file that will be placed on 'last_file_folder' -->
  <last_export_filename>resultados_eleicoes_{electionId}_2009
  </last_export_filename>
  <!-- classes that exports results -->
  <results_file_exporters>
    <export_election id="CM">
      <!-- Class that export current results to a file -->
      <results_file_exporter>com.criticalsoftware.elections.business.cache.
                      exporter.CityHallCSVExporter</results_file_exporter>
    </export_election>
    <export_election id="AM">
      <!-- Class that export current results to a file -->
      <results_file_exporter>com.criticalsoftware.elections.business.cache.
                  exporter.AssemblyHallCSVExporter</results_file_exporter>
    </export_election>
    <export_election id="AF">
      <!-- Class that export current results to a file -->
      <results_file_exporter>com.criticalsoftware.elections.business.cache.
                exporter.ParishAssemblyCSVExporter</results_file_exporter>
    </export_election>
  </results_file_exporters>
</export_file>

<!--
  (OPTIONAL) If this node is not defined, then reports will
             not be generated.
  NOTE: there must be at least one report
-->
<report>
  <!-- Folder that contains template and where report will be generated -->
  <folder_path>/etc/election-results-system/reports</folder_path>
  <!-- Time interval on minutes to update reports -->
  <update_interval>5</update_interval>
```

```
<!-- Report template name -->
<template_name>Mapas_controlo.xls</template_name>
<!-- Name that will be appended to report filename -->
<filename_appender>_autarquicas_2009</filename_appender>
<!--
  Name that will be appended to temporary report filename
  NOTE: THIS NAME SHOULD BE DIFFERENT FROM 'filename_appender'
-->
<tmp_filename_appender>_tmp</tmp_filename_appender>
<!-- Row number where date of report will be placed -->
<date_row>6</date_row>
<!--
  (OPTIONAL) Report with percentage of voters higher than
  'higher_percentage' or less than 'lower_percentage'
-->
<voters_subscribers>
  <!-- Sheet name-->
  <sheet_name>Percentagem_votantes</sheet_name>
  <!-- First row to write  -->
  <first_row>9</first_row>
  <higher_percentage>0.80</higher_percentage>
  <lower_percentage>0.35</lower_percentage>
</voters_subscribers>

<!--
  (OPTIONAL) Report with percentage of subscribers and reference
  subscribers difference, higher than 'higher_percentage' or less than
  'lower_percentage'
-->
<subscribers_reference_subscribers>
  <!-- Sheet name-->
  <sheet_name>Diferenca_5_por_cento</sheet_name>
  <!-- First row to write  -->
  <first_row>9</first_row>
  <higher_percentage>0.05</higher_percentage>
  <lower_percentage>-0.05</lower_percentage>
</subscribers_reference_subscribers>

<!--
  (OPTIONAL) Report with percentage from witch votes on the same option
  are considered high.
-->
<large_votes_amount>
  <!-- Sheet name-->
  <sheet_name>Freguesias_80_por_cento</sheet_name>
  <!-- First row to write  -->
```

```
    <first_row>9</first_row>
    <votes_percentage>0.80</votes_percentage>
  </large_votes_amount>

  <!-- (OPTIONAL) Closed territories report map -->
  <clearance_map>
    <!-- Sheet name-->
    <sheet_name>Mapa_apuramento_freguesias</sheet_name>
    <!-- First row to write  -->
    <first_row>9</first_row>
    <!-- Time interval on minutes for each count-->
    <time_interval>30</time_interval>
    <!-- Last count time-->
    <last_count_time>2009-10-12T00:00:00</last_count_time>
    <!-- Territory levels to be shown -->
    <levels>
      <level>2</level>
      <level>3</level>
    </levels>
  </clearance_map>

  <!-- (OPTIONAL) No voters report -->
  <no_voters>
    <!-- Sheet name-->
    <sheet_name>Freguesias_sem_votantes</sheet_name>
    <!-- First row to write  -->
    <first_row>9</first_row>
  </no_voters>

  <!-- (OPTIONAL) No subscribers report-->
  <no_subscribers>
    <!-- Sheet name-->
    <sheet_name>Freguesias_inscritos_zero</sheet_name>
    <!-- First row to write  -->
    <first_row>9</first_row>
  </no_subscribers>

  <!-- (OPTIONAL) Vote results report -->
  <vote_results>
    <!-- Sheet name-->
    <sheet_name>Data_hora_actualizacao</sheet_name>
    <!-- First row to write  -->
    <first_row>9</first_row>
  </vote_results>

  <!-- (OPTIONAL) contains tied territories -->
```

```
  <ties>
    <!-- Sheet name-->
    <sheet_name>Empates</sheet_name>
    <!-- First row to write  -->
    <first_row>9</first_row>
  </ties>

  <!-- (OPTIONAL) affluence results report -->
  <affluence_results>
    <!-- Sheet name-->
    <sheet_name>Afluencia_12</sheet_name>
    <!-- First row to write  -->
    <first_row>9</first_row>
    <hour>2009-10-11T12:00:00</hour>
    <foreign_voters>false</foreign_voters>
  </affluence_results>

  <!-- (OPTIONAL) affluence results report -->
  <affluence_results>
    <!-- Sheet name-->
    <sheet_name>Afluencia_16</sheet_name>
    <!-- First row to write  -->
    <first_row>9</first_row>
    <hour>2009-10-11T16:00:00</hour>
    <foreign_voters>false</foreign_voters>
  </affluence_results>
</report>

<!-- Backoffice configuration -->
<backoffice>
  <!-- (OPTIONAL) Backoffice title -->
  <!-- <title><title> -->
  <!-- Time to update backoffice user interface in seconds -->
  <ui_update_interval>10</ui_update_interval>
  <!-- Time to update backoffice cache in seconds -->
  <cache_update_interval>10</cache_update_interval>
</backoffice>

<!-- Frontend configuration -->
<frontend>
  <!--  frontend refresh timeout in seconds -->
  <refresh_timeout>20</refresh_timeout>
  <!-- time pattern -->
  <time_pattern>HH:mm</time_pattern>
  <!-- max search results to shown on frontend -->
  <max_search_results>80</max_search_results>
```

```
  </frontend>

  <!-- (OPTIONAL) merges options results on specified option -->
  <merge_option>
    <election id="CM">
      <init_level>2</init_level>
      <option_pattern>I|II|III|IV|V|VI|VII|VIII|IX|X|XI|XII|XIII|XIV|XV|XVI|
                          XVII|XVIII|XIX|XX|XXI|XXII|XXIII</option_pattern>
      <new_option>GRUPO CIDADÃOS</new_option>
    </election>
    <election id="AM">
      <init_level>2</init_level>
      <option_pattern>I|II|III|IV|V|VI|VII|VIII|IX|X|XI|XII|XIII|XIV|XV|XVI|
                          XVII|XVIII|XIX|XX|XXI|XXII|XXIII</option_pattern>
      <new_option>GRUPO CIDADÃOS</new_option>
    </election>
    <election id="AF">
      <init_level>3</init_level>
      <option_pattern>I|II|III|IV|V|VI|VII|VIII|IX|X|XI|XII|XIII|XIV|XV|XVI|
                          XVII|XVIII|XIX|XX|XXI|XXII|XXIII</option_pattern>
      <new_option>GRUPO CIDADÃOS</new_option>
    </election>
  </merge_option>

  <!-- Previous electoral act date -->
  <previous_electoral_act_date>2005-10-09</previous_electoral_act_date>
</configuration>
```

B.2 electoral_act.xml

Contém informações especificas acerca do acto eleitoral e das eleições que o constituem, tais como:

- nome do acto eleitoral;
- código e nome de cada uma das eleições;
- algoritmo de cálculo utilizado em cada uma das eleições;
- nível territorial a que se encontram definidos os círculos eleitorais.

```
<?xml version = "1.0" encoding = "UTF-8"?>

<!--
#####################################################
##    Configuration file for the Electoral Act     ##
#####################################################
```

```
-->

<!-- ELECTORAL ACT = Name-->
<electoral_act name="Autárquicas 2009">


<!--
====================
== Election List ==
====================
contains:
> question (optional)
> algorithm (name + path)
> local constituency (level)
> foreign constituency (level) - optional
> compensatory constituency (if exists)
> position - tell us the proper sequence that elections shall follow
in vote results insertion
-->

  <elections>
    <election id="AF" name="Assembleia de Freguesia">
      <position>1</position>
      <algorithm>Hondt</algorithm>
      <local_constituency_level>4</local_constituency_level>
    </election>
    <election id="AM" name="Assembleia Municipal">
      <position>2</position>
      <algorithm>Hondt</algorithm>
      <local_constituency_level>3</local_constituency_level>
    </election>
    <election id="CM" name="Câmara Municipal">
      <position>3</position>
      <algorithm>Hondt</algorithm>
      <local_constituency_level>3</local_constituency_level>
    </election>
  </elections>

</electoral_act>
```

B.3 territory_hierarchy.csv

Ficheiro que contém a hierarquia de territórios, nomeadamente:

- região que pode ser: GLOBAL, LOCAL ou FOREIGN;
- nível de profundidade na árvore de territórios. Sendo o nível 0 (zero) a raiz da árvore de territórios;
- descrição do nível territorial.

```
Region,Level,Description
GLOBAL,0,GLOBAL_RESULTS
LOCAL,1,País
LOCAL,2,Distrito
LOCAL,3,Municipio
LOCAL,4,Freguesia
```

B.4 territory.csv

Contém informações acerca do mapa de territórios sobre o qual ocorrem as eleições, nomeadamente:

- região que pode ser: GLOBAL, LOCAL ou FOREIGN;
- código do território;
- descrição ou nome do território;
- região do território pai agregador (e.g. no caso de ser uma freguesia, indica a região do concelho a que pertence);
- código do território pai agregador;

```
Region,Territory_ID,Description,[Parent_Region],[Parent_ID]
GLOBAL,990000,Resultados Globais,,
LOCAL,500000,Portugal,GLOBAL,990000
LOCAL,010000,Aveiro,LOCAL,500000
LOCAL,020000,Beja,LOCAL,500000
LOCAL,030000,Braga,LOCAL,500000
(...)
LOCAL,010100,Águeda,LOCAL,010000
LOCAL,010200,Albergaria-a-Velha,LOCAL,010000
LOCAL,010300,Anadia,LOCAL,010000
LOCAL,010400,Arouca,LOCAL,010000
LOCAL,010500,Aveiro,LOCAL,010000
(...)
LOCAL,030100,Amares,LOCAL,030000
LOCAL,030200,Barcelos,LOCAL,030000
LOCAL,030300,Braga,LOCAL,030000
LOCAL,030400,Cabeceiras de Basto,LOCAL,030000
LOCAL,030500,Celorico de Basto,LOCAL,030000
```

```
LOCAL,030600,Esposende,LOCAL,030000
LOCAL,030700,Fafe,LOCAL,030000
LOCAL,030800,Guimarães,LOCAL,030000
(...)
LOCAL,010101,Agadão,LOCAL,010100
LOCAL,010102,Aguada de Baixo,LOCAL,010100
LOCAL,010103,Aguada de Cima,LOCAL,010100
LOCAL,010104,Águeda,LOCAL,010100
(...)
```

B.5 affuences.csv

Contém informações acerca de quais os territórios seleccionados para fazerem parte do cálculo dos níveis de afluência às urnas para a eleição, nomeadamente:

- região que pode ser: GLOBAL, LOCAL ou FOREIGN;
- código do território;
- descrição ou nome do território;
- informação se o território recebe ou não afluências de cidadãos estrangeiros.

```
REGION,ID,Description,Foreign-Affluences
LOCAL,010101,Agadão,
LOCAL,010205,Frossos,
(...)
LOCAL,031219,Joane,
LOCAL,031304,Atiães,
LOCAL,031404,Infias,
(...)
```

B.6 reference_subscribers.csv

Contém informações relativas ao número de inscritos de cada um dos territórios do nível mais baixo (e.g. freguesias), nomeadamente:

- região que pode ser: GLOBAL, LOCAL ou FOREIGN;
- código do território;
- descrição ou nome do território;
- número de eleitores inscritos a tomar como referência.

```
RegionID,TerritoryID,Description,Subscribers
LOCAL,010101,Agadão,462
LOCAL,010102,Aguada de Baixo,1598
LOCAL,010103,Aguada de Cima,3610
LOCAL,010104,Águeda,10328
(...)
LOCAL,031219,Joane,6599
LOCAL,031220,Lagoa,775
LOCAL,031221,Landim,2706
LOCAL,031222,Lemenhe,1190
LOCAL,031223,Louro,2101
LOCAL,031224,Lousado,3325
LOCAL,031225,Mogege,1676
LOCAL,031226,Mouquim,1132
LOCAL,031227,Nine,2468
(...)
LOCAL,181001,Arca,348
LOCAL,181002,Arcozelo das Maias,1479
LOCAL,181003,Destriz,382
LOCAL,181004,Oliveira de Frades,2119
LOCAL,181005,Pinheiro,1154
LOCAL,181006,Reigoso,325
LOCAL,181007,Ribeiradio,1053
LOCAL,181008,São João da Serra,597
LOCAL,181009,São Vicente de Lafões,719
(...)
LOCAL,480201,Caveira,67
LOCAL,480202,Cedros,118
LOCAL,480203,Ponta Delgada,354
LOCAL,480204,Santa Cruz das Flores,1466
LOCAL,490101,Corvo,352
```

B.7 timezones.xml

Contém informações acerca da diferença horária (se existir) entre os territórios que *participam* na eleição relativamente ao horário do servidor da aplicação. No caso do ficheiro em anexado, entende-se que o território cuja região é LOCAL e o código é 400000 se encontra com menos 1 (uma) hora do que a indicada no servidor da aplicação.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--
================================================
== By default TERRITORIES have :    timezone=0 ==
================================================
-->
```

```
<Regions>

   <region id="LOCAL">
      <timezones>
         <timezone gmt="-1">
            <territory>400000</territory>
         </timezone>
      </timezones>
   </region>

</Regions>
```

B.8 political_parties.csv

Contém informações acerca das opções de voto (partidos) que participam no acto eleitoral, nomeadamente:

- descrição ou nome da opção de voto ou partido político;
- acrónimo da opção de voto;
- endereço da página web da opção de voto (opcional).

```
"Description","OptionName_Acronym","[URL]"
"Bloco de Esquerda","B.E.",""
"CDS - Partido Popular","CDS-PP",""
"Sangue Novo Pela Nossa Terra","CDS-PP.MPT",""
"JUNTOS POR FORNOS","CDS-PP.PPM",""
"LIFA - Lista Independente da Freguesia de Águeda","I",""
(...)
"Partido da Terra","MPT",""
"CDU - Coligação Democrática Unitária","PCP-PEV",""
"Partido Comunista dos Trabalhadores Portugueses","PCTP/MRPP",""
"Nova Democracia","PND",""
"Partido Nacional Renovador","P.N.R.",""
"Partido Social Democrata","PPD/PSD",""
(...)
```

B.9 constituencies.csv

Contém informações acerca dos círculos eleitorais definidos para as várias eleições configuradas, nomeadamente:

- código da eleição;
- região do território que pode ser: GLOBAL, LOCAL ou FOREIGN;

- código do território;
- número de mandatos a atribuir.

```
ElectionID,Region,TerritoryID,Mandates
AF,LOCAL,010101,7
AF,LOCAL,010102,9
AF,LOCAL,010201,13
AF,LOCAL,010202,9
(...)
AM,LOCAL,010100,21
AM,LOCAL,010200,21
AM,LOCAL,020100,15
CM,LOCAL,010100,7
CM,LOCAL,010200,7
CM,LOCAL,020100,5
CM,LOCAL,030100,7
(...)
```

B.10 options.csv

Contém informações acerca das opções de voto que concorrem em cada um dos círculos eleitorais definidos, ou seja, a definição dos boletins de voto para cada círculo, nomeadamente:

- código da eleição;
- região do território que pode ser: GLOBAL, LOCAL ou FOREIGN;
- código do território;
- posição da opção de voto no boletim;
- acrónimo da opção de voto ou partido.

```
"ElectionID","Region","TerritoryID","Position","OptionName"
"AM","LOCAL","010100","1","PCP-PEV"
"AM","LOCAL","010100","2","B.E."
"AM","LOCAL","010100","3","PS"
"AM","LOCAL","010100","4","CDS-PP"
"AM","LOCAL","010100","5","PPD/PSD"
"CM","LOCAL","010100","1","PCP-PEV"
"CM","LOCAL","010100","2","B.E."
"CM","LOCAL","010100","3","PS"
"CM","LOCAL","010100","4","CDS-PP"
"CM","LOCAL","010100","5","PND"
"CM","LOCAL","010100","6","PPD/PSD"
"AF","LOCAL","010101","1","PS"
```

```
"AF","LOCAL","010101","2","PPD/PSD"
(...)
"AF","LOCAL","031219","1","PCP-PEV"
"AF","LOCAL","031219","2","PS"
"AF","LOCAL","031219","3","PPD/PSD.CDS-PP"
"AF","LOCAL","031219","4","B.E."
(...)
```

B.11 candidates_list.csv

Contém informação acerca da lista de candidatos de cada opção de voto em cada círculo eleitoral, nomeadamente:

- código da eleição;
- região do território que pode ser: GLOBAL, LOCAL ou FOREIGN;
- código do território;
- acrónimo da opção de voto ou partido.
- tipo de candidato: efectivo (E) ou substituto (A);
- posição do candidato na lista;
- nome do candidato.

```
ElectionID,RegionID,TerritoryID,OptionName,CandidateType,Position,Name
"CM","LOCAL","010400","II","E","1","Vítor Fernando Gomes Brandão"
"CM","LOCAL","010500","PPD/PSD.CDS-PP","E","1","Élio Manuel Delgado da Maia"
"CM","LOCAL","010800","PPD/PSD.CDS-PP","E","1","José Eduardo Alves Valente
    de Matos"
"CM","LOCAL","011500","PPD/PSD.CDS-PP","E","1","Álvaro Manuel Reis Santos"
"CM","LOCAL","011700","IV","E","1","João Miguel Tavares de Almeida"
"CM","LOCAL","011800","X","E","1","Mário dos Santos Martins Júnior"
"CM","LOCAL","020100","PPD/PSD.CDS-PP","E","1","Carlos Miguel Maia Palma
    Agatão"
"CM","LOCAL","020300","PPD/PSD.CDS-PP.PPM","E","1","Mário Nelson da Silva
    Vaz Simões"
"CM","LOCAL","020500","VI","E","1","Dulce do Carmo Lopes Caleiro Amaral"
"CM","LOCAL","020600","PPD/PSD.CDS-PP","E","1","Pedro Daniel Ferreira de
    Castro Figueira"
"CM","LOCAL","020900","XI","E","1","José Carlos Raposo Rodrigues Celorico"
"CM","LOCAL","021100","PPD/PSD.CDS-PP","E","1","José Francisco de Sousa Prado
    Santos Silva"
"CM","LOCAL","030100","I","E","1","José Lopes Gonçalves Barbosa"
"CM","LOCAL","030300","PPD/PSD.CDS-PP.PPM","E","1","Ricardo Bruno Antunes
    Machado Rio"
```

```
"CM","LOCAL","030400","PPD/PSD.CDS-PP","E","1","Luis Miguel Jorge Gonçalves"
"CM","LOCAL","030700","PPD/PSD.CDS-PP","E","1","José Humberto Fernandes Castro"
"CM","LOCAL","030700","XVIII","E","1","Parcídio Cabral de Almeida Summavielle"
"CM","LOCAL","031100","PPD/PSD.CDS-PP","E","1","Albino José da Silva Carneiro"
"CM","LOCAL","031200","PPD/PSD.CDS-PP","E","1","Armindo Borges Alves da Costa"
"CM","LOCAL","031400","PPD/PSD.CDS-PP","E","1","Luis Miguel Soares Lopes
    Guimarães"
"CM","LOCAL","040100","PPD/PSD.CDS-PP","E","1","Arsénio da Paixão Tomé Pereira"
(...)
```

B.12 details_Manager.csv

Contém informação detalhada acerca dos utilizadores com o cargo de *Manager*, nomeadamente:

- região e código do território a que o utilizador se encontrará restricto;
- *login* do utilizador a ser criado;
- palavra-chave associada;
- tipo de acesso do utilizador: podendo ser de leitura (R) ou de leitura-escrita (RW);
- endereço IP a ser usado pelo utilizador;
- perfil do utilizador na aplicação.

```
Region,TerritoryID,LOGIN,PASS,AccessType,IPAddress,RoleProfile
GLOBAL,990000,Dgai1,yqko38r2,R,*.*.*.*,DGAI
GLOBAL,990000,Mandre,izutea3z,RW,*.*.*.*,DGAI
GLOBAL,990000,Hseixas,s3jfdymh,RW,*.*.*.*,DGAI
GLOBAL,990000,Aabreu,5tsrr4z0,RW,*.*.*.*,DGAI
GLOBAL,990000,Pvasco,gmtcvy4d,RW,*.*.*.*,DGAI
```

B.13 details_Regular.csv

Contém informação detalhada acerca dos utilizadores com o cargo de *Regular*, sendo o ficheiro semelhante em todos os aspectos ao ficheiros apresentado na secção anterior.

```
Region,TerritoryID,LOGIN,PASS,AccessType,IPAddress,RoleProfile
LOCAL,010000,D01a,fksikvxt,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01b,5rrgznhx,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01c,089m6jes,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01d,2lf62x7w,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01e,3srbbd2e,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01f,eleg8bhp,RW,*.*.*.*,Governo Civil
```

```
LOCAL,010000,D01g,p05wpro5,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01h,k3bjtkbp,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01i,uxgj0932,RW,*.*.*.*,Governo Civil
LOCAL,010000,D01j,45q0rted,RW,*.*.*.*,Governo Civil
LOCAL,020000,D02a,crx5zwhp,RW,*.*.*.*,Governo Civil
LOCAL,020000,D02b,0cyzrbh7,RW,*.*.*.*,Governo Civil
LOCAL,020000,D02c,i4alx54r,RW,*.*.*.*,Governo Civil
(...)
```

B.14 old-results-data.csv

Contém informação acerca dos resultados de eleições anteriores semelhantes para efeitos de comparações, nomeadamente:

- código da eleição com a qual são comparáveis os resultados;
- data da eleição;
- região e código do território;
- descrição ou nome do território;
- se tem ou não mandatos;
- total de mandaos a atribuir;
- número de inscritos;
- número e percentagem de votantes;
- número e percentagem de votos em branco;
- número e percentagem de votos nulos;
- para cada partido ou opção de voto:
  - número de votos e percentagem associada;
  - mandatos adquiridos;
  - número de presidentes eleitos;
  - número de maiorias absolutas e relativas;

```
Election_id,Election_date,Territory_ID,Territory_Region,Description,
    hasMandates?,MandatesToAssign,Subscribers,Voters,Voters,Blank-Votes,
    Blank-Votes,Null-Votes,Null-Votes,VotingOption-Name,VotesNumber,
    VotesNumber,Mandates,Presidents,Absolute Majorities,Relative Majorities,
    Parish Counter,County Counter,VotingOption-Name,VotesNumber,VotesNumber,
    Mandates,Presidents,Absolute Majorities,Relative Majorities,Parish
    Counter,County Counter,VotingOption-Name,VotesNumber,VotesNumber,
    Mandates,Presidents,Absolute Majorities,Relative Majorities,Parish
```

Counter,County Counter,,,,,Presidents,Absolute Majorities,Relative
Majorities,Parish Counter,County Counter,,,,,Presidents,Absolute
Majorities,Relative Majorities,Parish Counter,County Counter,,,,,
Presidents,Absolute Majorities,Relative Majorities,Parish Counter,County
Counter,,,,,Presidents,Absolute Majorities,Relative Majorities,Parish
Counter,County Counter,,,,,Presidents,Absolute Majorities,Relative
Majorities,Parish Counter,County Counter,,,,,Presidents,Absolute
Majorities,Relative Majorities,Parish Counter,County Counter,,,,,
Presidents,Absolute Majorities,Relative Majorities,Parish Counter,County
Counter,,,,,Presidents,Absolute Majorities,Relative Majorities,Parish
Counter,County Counter#,,
,,
AF,2005-10-09,990000,GLOBAL,Resultados Globais,TRUE,34616,8840223,5381759,
60.88,151529,2.82,101081,1.88,B.E.,146898,2.73,229,3,2,,448,,CDS-PP,
144328,2.68,822,65,51,,1043,,CDS-PP.PPD/PSD,1448,0.03,10,2,2,,2,,
GRUPO CIDADÃOS,245740,4.57,2200,292,260,,568,,MPT,3100,0.06,23,3,2,,7,,
P.H.,824,0.02,0,0,0,,10,,P.N.R.,38,0.00,0,0,0,,1,,PCP-PEV,644535,11.98,
2555,244,177,,2239,,PCTP/MRPP,3635,0.07,0,0,0,,28,,PND,2809,0.05,6,0,0,
,16,,PPD/PSD,1434321,26.65,12451,1723,1593,,3065,,PPD/PSD.CDS-PP,411517,
7.65,2064,219,199,,529,,PPD/PSD.CDS-PP.MPT,1756,0.03,32,3,3,,12,,
PPD/PSD.CDS-PP.PPM,84398,1.57,400,34,26,,125,,PPD/PSD.CDS-PP.PPM.MPT,
95175,1.77,216,19,7,,45,,PPD/PSD.PPM,807,0.01,6,0,0,,5,,PPM,27,0.00,0,0,
0,,1,,PS,1907721,35.45,13484,1518,1348,,3711,,PSN,72,0.00,0,0,0,,5,,,,,,
,,,,,,,,,,,,,
AF,2005-10-09,500000,LOCAL,Portugal,TRUE,34616,8840223,5381759,60.88,151529,
2.82,101081,1.88,B.E.,146898,2.73,229,3,2,,448,,CDS-PP,144328,2.68,822,
65,51,,1043,,CDS-PP.PPD/PSD,1448,0.03,10,2,2,,2,,GRUPO CIDADÃOS,245740,
4.57,2200,292,260,,568,,MPT,3100,0.06,23,3,2,,7,,P.H.,824,0.02,0,0,0,,
10,,P.N.R.,38,0.00,0,0,0,,1,,PCP-PEV,644535,11.98,2555,244,177,,2239,,
PCTP/MRPP,3635,0.07,0,0,0,,28,,PND,2809,0.05,6,0,0,,16,,PPD/PSD,1434321,
26.65,12451,1723,1593,,3065,,PPD/PSD.CDS-PP,411517,7.65,2064,219,199,,
529,,PPD/PSD.CDS-PP.MPT,1756,0.03,32,3,3,,12,,PPD/PSD.CDS-PP.PPM,84398,
1.57,400,34,26,,125,,PPD/PSD.CDS-PP.PPM.MPT,95175,1.77,216,19,7,,45,,
PPD/PSD.PPM,807,0.01,6,0,0,,5,,PPM,27,0.00,0,0,0,,1,,PS,1907721,35.45,
13484,1518,1348,,3711,,PSN,72,0.00,0,0,0,,5,,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010000,LOCAL,AVEIRO,TRUE,1928,594748,377932,63.54,11339,3.00,
6712,1.78,B.E.,3210,0.85,4,0,0,,13,,CDS-PP,25999,6.88,131,8,4,,73,,,,,,
,,,,,GRUPO CIDADÃOS,25756,6.81,168,18,17,,55,,MPT,169,0.04,0,0,0,,1,,,,
,,,,,,,P.N.R.,38,0.01,0,0,0,,1,,PCP-PEV,16537,4.38,31,0,0,,119,,,,,,,,,
,,PND,1409,0.37,3,0,0,,3,,PPD/PSD,137067,36.27,827,110,98,,167,,
PPD/PSD.CDS-PP,31641,8.37,144,17,15,,26,,,,,,,,,,,PPD/PSD.CDS-PP.PPM,
3046,0.81,29,4,4,,9,,,,,,,,,,,,,,,,,,,,,PPM,27,0.01,0,0,0,,1,,PS,114982,
30.42,584,50,45,,169,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010100,LOCAL,AGUEDA,TRUE,168,41809,25576,61.17,798,3.12,475,
1.86,B.E.,194,0.76,0,0,0,,2,,CDS-PP,1988,7.77,11,2,1,,9,,,,,,,,,,,
GRUPO CIDADÃOS,2912,11.39,23,3,3,,8,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PCP-PEV,

```
737,2.88,1,0,0,,6,,,,,,,,,,,,,,,,,,,,PPD/PSD,9633,37.66,73,10,10,,17,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PS,8839,34.56,53,4,3
    ,,16,,,,,,,,,,,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010101,LOCAL,AGADAO,TRUE,7,519,407,78.42,6,1.47,3,0.74,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,III,156,38.33,3,0,0,,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,PPD/PSD,242,59.46,4,1,1,,1,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010102,LOCAL,AGUADA DE BAIXO,TRUE,9,1491,899,60.30,89,9.90,21,
    2.34,,,,,,,,,,,,,,,,,,,,,,,,,,,,II,572,63.63,7,1,1,,1,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,PS,217,24.14,2,0,0,,1,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,
AF,2005-10-09,010104,LOCAL,AGUEDA,TRUE,13,10062,5373,53.40,184,3.42,88,1.64,
    B.E.,164,3.05,0,0,0,,1,,CDS-PP,262,4.88,0,0,0,,1,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,PCP-PEV,281,5.23,0,0,0,,1,,,,,,,,,,,,,,,,,,,,
    PPD/PSD,2104,39.16,6,0,0,,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,PS,2290,42.62,7,1,1,,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010105,LOCAL,BARRO,TRUE,9,1486,986,66.35,41,4.16,13,1.32,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,PPD/PSD,536,54.36,5,1,1,,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,PS,396,40.16,4,0,0,,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010107,LOCAL,CASTANHEIRA DO VOUGA,TRUE,7,620,414,66.77,41,
    9.90,29,7.00,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PPD/PSD,344,83.09,7,1,1,,1,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,
AF,2005-10-09,010108,LOCAL,ESPINHEL,TRUE,9,2482,1531,61.68,30,1.96,38,2.48,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,VI,699,45.66,5,1,1,,1,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PPD/PSD,361,23.58,2,0,0,,1,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PS,403,26.32,2,0,0,,1,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010109,LOCAL,FERMENTELOS,TRUE,9,2523,1720,68.17,40,2.33,40,
    2.33,,,,,,,,,,CDS-PP,258,15.00,1,0,0,,1,,,,,,,,,,,,VII,259,15.06,1,0,0,,1
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PPD/PSD,965,
    56.10,6,1,1,,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PS
    ,158,9.19,1,0,0,,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,
AF,2005-10-09,010110,LOCAL,LAMAS DO VOUGA,TRUE,7,718,472,65.74,11,2.33,3,
    0.64,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,PPD/PSD,292,61.86,5,1,1,,1,,,,,,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PS,166,35.17,2,0,0,,1,,,,,,,,,,,,,,,,,,
    ,,,,,,,,,,
(...)
```

Apêndice C

Conteúdo do ficheiro - Rules.drl

Neste apêndice podemos encontrar o conteúdo do ficheiro *Rules.drl* utilizado no contexto do Capítulo 9 para efeitos de avaliação da solução desenvolvida neste trabalho de mestrado.

```
package com.sample

import com.uminho.eleic.fact.model.Election
import com.uminho.eleic.fact.model.Territory
import com.uminho.eleic.fact.model.TerritoryRegion

/*
 * ELECTION RULES
 */
rule "constituencyLevelForAF"
    dialect "mvel"
    when
        myElection : Election( id matches "AF" ,
       localConstituencyLevel != "4" )
    then
        System.out.println("» AF constituency shall be located at level 4!")
    System.out.println("      - instead is located at: "
                           + myElection.localConstituencyLevel);
end

rule "constituencyLevelForAM"
    dialect "mvel"
    when
        myElection : Election( id matches "AM" ,
                               localConstituencyLevel != "3" )
    then
        System.out.println("» AM constituency shall be located at level 3!")
    System.out.println("      - instead is located at: "
                           + myElection.localConstituencyLevel);
```

```
end

rule "constituencyLevelForCM"
    dialect "mvel"
    when
        myElection : Election( id matches "CM" ,
                               localConstituencyLevel != "3" )
    then
        System.out.println("» CM constituency shall be located at level 3!");
System.out.println("      - instead is located at: "
                           + myElection.localConstituencyLevel);
end


/*
 * Territórios que nao sejam de nível 0, têm de ter 1 pai associado.
 */
rule "checkTerritoryParentsExistence"
    dialect "mvel"
    when
     $myTerritory : Territory()
        exists Territory( $myTerritory.parentTerritoryId == null ,
                          $myTerritory.level != "0" )
    then
        System.out.println("» There are disconnected territories: "
                           + $myTerritory.pk.region + "-"
                           + $myTerritory.pk.id);
end

/*
 * Só pode existir um território ROOT - de nível 0 e com Região GLOBAL.
 */
 rule "singleRootTerritory"
    dialect "mvel"
    when
      # $root : Territory( pk.region == TerritoryRegion.GLOBAL, level == "0")
      $list : java.util.List( ) from collect ( Territory(
              pk.region == TerritoryRegion.GLOBAL, level == "0") )
      eval ($list.size() = 1)
    then
        System.out.println("» There are: " + $list.size()
                           + " ROOT Territories:");
        for(Territory $i : $list){
            System.out.println("      - " + $i.pk.region
                               + "-" + $i.pk.id);
        }
```

```
end

/*
 * Não podem existir territórios com número de inscritos de referencia = 0.
 */
 rule "zeroReferenceSubscribers"
    dialect "mvel"
    when
        $list : java.util.List( ) from collect (
            Territory( referenceSubscribers == 0 ||
                       referenceSubscribers == null)
            )
        eval ($list.size() > 0)
    then
        System.out.println("» There are: " + $list.size()
                        + " Territories with no Reference Subscribers:");
        for(Territory $i : $list){
            System.out.println("      - " + $i.pk.region
                            + "-" + $i.pk.id);
        }
end

/*
 * Um território não pode ter um território de nível superior
 * (ou seja um pai que seja igual a si mesmos)
 */
 rule "parentTerritoryMustBeDifferentFromItself"
    dialect "mvel"
    when
        $list : java.util.List( ) from collect (
         Territory(
         parentTerritoryId != null &&
         pk.id == parentTerritoryId.pk.id ,
                        pk.region == parentTerritoryId.pk.region
         )
        )
        eval ($list.size() > 0)
    then
        System.out.println("» There are: " + $list.size()
                        + " Territories with Illegal Parent:");
        for(Territory $i : $list){
         System.out.println("      - " + $i.pk.region
                                + "-" + $i.pk.id);
        }
end
```

```
/*
 * Não podem existir territórios cujo nome seja igual a vazio ("")
 */
 rule "emptyNameTerritories"
    dialect "mvel"
    when
        $list : java.util.List( ) from collect (
            Territory( description == "")
        )
        eval ($list.size() = 0)
    then
        System.out.println("» There are: " + $list.size()
                        + " Territories with no name:");
        for(Territory $i : $list){
         System.out.println("      - "
                                + $i.pk.region + "-" + $i.pk.id);
        }
end

/*
 * As passwords de todos os utilizadores devem ter um tamanho igual ou
 * superior a 8 caracteres.
 */
 rule "passwordsSize"
    dialect "mvel"
    when
        $list : java.util.List( ) from collect ( User( pass.size() <= 8) )
        eval ($list.size() > 0)
    then
        System.out.println("» There are: " + $list.size()
                        + " users whose passwords are incorrect:");
        for(User $i : $list){
         System.out.println("      - " + $i.login);
        }

/*
 * Qualquer eleição configurada deverá ter pelo menos 1 círculo eleitoral
 * configurado.
 */
 rule "minimumElectionsConstituencies"
    dialect "mvel"
    when
$elections : java.util.List( ) from collect ( Election( ) )
        $result : java.util.List( );
forall(Election $i : $elections){
         $list : java.util.List( ) from collect (
```

```
Constituency( elections.id == $i.id )
)
if ( $list.size()>0 ){
   $result.add($i.id);
}
         eval ($list.size() > 0)
}
    then
        System.out.println("» There are: " + $result.size()
+ " elections with no constituencies.");
        for($i : $result){
         System.out.println("      - " + $i);
        }
 end
```

Apêndice D

JBoss Drools

`JBoss Rules` ou simplesmente `Drools` [JBo12a], é uma plataforma de integração de lógica de negócio desenvolvido pela `Red Hat, Inc` em conjunto com a comunidade do `JBoss` como um projecto *open source* sob a licença de software da Apache[1] [ASF12].

De um modo geral o `Drools` consiste num agregado de vários projectos numa única plataforma para gestão de **regras**, ***workflow*** e processamento de **eventos**:

- `Drools Guvnor` — como gestor de regras de negócio;
- `Drools Expert` — como motor de regras;
- `jBPM` — como gestor de processos de negócio;
- `Drools Fusion` — como processador de eventos;
- `Drools Planner` — para planeamentos automatizados;

D.1 Drools Guvnor

O `Drools Guvnor` funciona como um repositório centralizado para regras de negócio, dotado de uma aplicação web que permite efectuar toda a gestão dessas mesmas regras; permitindo a criação e manutenção de repositórios de regras.

O `Guvnor` é particularmente útil pois permite:

- controlo de acessos a todos os artefactos do repositório;
- construcção e edição das regras através de diversos editores disponíveis (gráfico, textual ou por tabelas de decisão);
- controlo de versões e das alterações produzidas ao longo do tempo;
- implementação de cenários de teste;
- mecanismo integrado para *deploy* das regras;

[1] A licença Apache é uma licença para software *open source* da autoria da *Apache Software Foundation* (`ASF`) e visa regular as contribuições individuais e colectivas com o objectivo de ajudar ao desenvolvimento de produtos de software de confiança.

D.2 Drools Expert

O `Drools Expert` é o motor responsável pelo processamento dos dados face a um determinado repositório de modo a inferir conclusões.

Este motor de *inferência* procura fazer *match* dos factos e dados existentes face a regras de produção definidas; permitindo tirar conclusões que posteriormente resultarão em acções.

As **regras de produção** referidas seguem o seguinte formato:

```
1  When
2      <conditions>
3  Then
4      <actions>
```

Ao processo de *match* referido chamamos de *pattern matching* que, no caso do `Drools Expert`, utiliza uma versão melhorada do **algoritmo de Rete** optimizada para aplicações orientadas a objectos.

De um modo geral, a vantagem de utilização de um motor de regras prende-se com o facto de permitir a separação entre os dados (nos objectos que compõem o domínio) e a lógica (presente nas regras), permitindo expressar soluções de problemas complexos de forma simples através de regras que podem ser verificadas.

D.3 jBPM

O `jBPM` é um sistema flexivel de gestão de processos de negócio, com *focus* não só nos utilizadores do negócio mas também nos programadores, que permite modelar os objectivos de negócio utilizando um fluxograma (ver exemplo na Figura D.1).

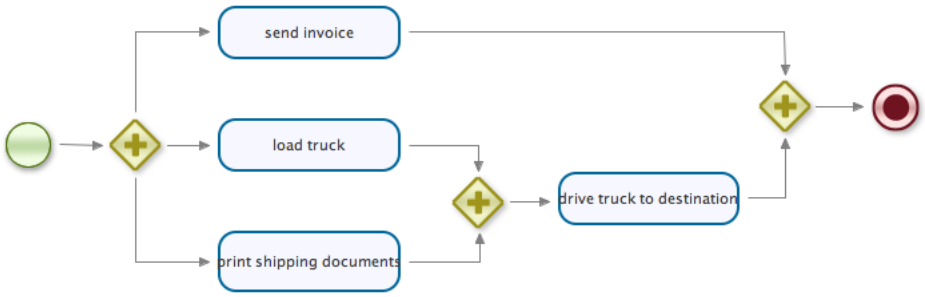

Figura D.1: Exemplo de um fluxograma.

O *core* é constituído por um motor de *workflow* escrito em `Java` que permite a definição de processos de negócio recorrendo à especificação `BPMN 2.0` [OMG12][2].

[2]BPMN — Business Process Modeling Notation.

D.4 Drools Fusion

O `Drools Fusion` é um processador de eventos complexos que, assumindo que "`evento`" é o registo de uma alteração ao estado no domínio da aplicação, permite:

1. seleccionar um conjunto de eventos;
2. detectar relações (padrões) relevantes entre eles;
3. tomar acções com base nos *padrões* detectados.

D.5 Drools Planner

O `Drools Planner` é uma *framework* para optimização de planeamentos automatizados desenvolvida em `Java`, capaz de lidar com problemas de planeamento **NP-Completos** através da utilização de algoritmos de aproximação ou heurísticos.

São exemplos os seguintes casos:

- ***Bin Packing*** — colocar cada item numa localização dentro de um contentor;
- ***Knapsack*** — dado um conjunto de items, com um peso e valor especifico, determinar o valor máximo dos objectos que se consegue colocar numa mochila;
- ***Traveling Salesman*** — problema do caixeiro viajante;
- ***Employee Shift Rostering*** — problema de escalonamento de turnos;
- ***Educational Timetabling*** — agendar aulas, cursos, exames, etc;
- ***Sport Scheduling*** — planear ligas de futebol, etc;
- ...

O algoritmo de planeamento a ser utilizado pode ser configurado de modo a ser adaptado a qualquer um dos problemas acima indicados não obstante as restrições existentes. Sendo a procura pela melhor solução feita durante uma determinada quantidade de tempo especificada.

Apêndice E

Leis de DeMorgan

As leis de DeMorgan são regras que mostram as transformações, por efeito da negação, sobre operadores lógicos (de conjunção e disjunção) e quantificadores (universais e existenciais).

Sejam p e q proposições, '$\neg$' o operador de negação, '$\vee$' o operador de disjunção e '$\wedge$' o operador de conjunção, então as seguintes propriedades descrevem as *primeiras* leis de DeMorgan [dO82, Hal74, Fej91]:

Lei 1. *a* ***negação de uma disjunção*** *de duas proposições é logicamente equivalente à conjunção de duas proposições negadas;*

$$\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$$

Lei 2. *a* ***negação de uma conjunção*** *de duas proposições é logicamente equivalente à disjunção de duas proposições negadas;*

$$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$$

As *segundas* leis de DeMorgan são importantes na negação de proposições com quantificadores: universais e existenciais.

Seja $p(x)$ uma condição em que $x \in A$ (conjunto), '$\forall$' o quantificador universal e '$\exists$' o quantificador existencial, tem-se:

Lei 3. *para todo o elemento x de A, verifica-se a condição $p(x)$; cuja negação é lógicamente equivalente à existência de pelo menos um x de A tal que a condição $p(x)$ seja falsa;*

$$\neg[\forall x \in A : p(x)] \Leftrightarrow \exists x \in A : \neg p(x)$$

Lei 4. *existe pelo menos um x de A tal que se verifica a condição $p(x)$; cuja negação é logicamente equivalente a dizer que para todo o elemento x de A a condição $p(x)$ é falsa;*

$$\neg[\exists x \in A : p(x)] \Leftrightarrow \forall x \in A : \neg p(x)$$

Pelo que se verifica que a negação de uma quantificador universal se transforma num quantificador existencial, e a negação de um quantificador existencial num quantificador universal; ambos seguidos da negação da condição.