Mestrado em Engenharia Informática
Dissertação
Relatório Final

Projeto UC-Num: desenvolvimento de uma *Data Warehouse* para a Universidade de Coimbra – Estágio A

Ana Miguel Tavares Ilharco
ilharco@student.dei.uc.pt

Orientador:
Professor Doutor Bruno Cabral (DEI)

1 de setembro de 2015

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática
Dissertação
Relatório Final

Título: Projeto UC-Num: desenvolvimento de uma *Data Warehouse* para a Universidade de Coimbra – Estágio A

Autor: Ana Miguel Tavares Ilharco
ilharco@student.dei.uc.pt

Elementos do júri:
Orientador do DEI: Professor Doutor Bruno Cabral
(bcabral@dei.uc.pt)
Júri Arguente: Professor Doutor Henrique Madeira
(henrique@dei.uc.pt)
Júri Vogal: Professor Doutor Rui Pedro Paiva
(ruipedro@dei.uc.pt)

1 de setembro de 2015

Histórico de versões

Data	Revisão	Versão
15/09/2014	Estrutura do documento.	0.1.0
22/09/2014	Pesquisa e escrita da seleção de tecnologias.	0.2.0
Até 20/01/2015	Escrita dos restantes capítulos para a primeira versão.	0.3.0
24/01/2015	Revisão por parte do professor Bruno Cabral.	0.3.1
25/01/2015	Revisão por parte do elemento de equipa Hugo Costa.	0.3.2
26/01/2015	Correções efetuadas após revisões supra citadas.	0.4.0
27/01/2015	Revisão por parte da gestora de projeto Beatriz Fragoso.	0.4.1
28/01/2015	Alterações e correções após revisão anterior.	0.5.0
30/01/2015	Entrega do relatório intermédio.	1.0.0
24/02/2015	Correções feitas após *feedback* do júri.	1.1.0
01/08/2015	Colocação do capítulo de planeamento para o início do relatório.	1.2.0
02/08/2015	Estruturação do conteúdo dos restantes capítulos. Primeiro *draft* pronto e entregar.	1.3.0
12/08/2015	Revisão por parte do professor Bruno Cabral.	1.3.1
Até 26/08/2015	Escrita dos restantes capítulos, pequenas alterações nos capítulos do relatório intermédio.	1.4.0
27/08/2015	Reescrita do capítulo de validação e testes.	1.5.0
28/08/2015	Revisão por parte da gestora de projeto Beatriz Fragoso.	1.5.1
29/08/2015	Escrita do anexo sobre análise de riscos.	1.6.0
30/08/2015	Escrita do anexo com as transformações do processo de ETL.	1.7.0.
31/08/2015	Correções e ajustes feitos após revisão anterior.	1.8.0
01/09/2015	Entrega do relatório final.	2.0.0

A versão deste documento segue o formato `X.Y.Z` (Entrega.Adição.Revisão). A entrega do documento implica o incremento de uma unidade em `X`. A adição de um capítulo, secção ou correções implica um incremento em `Y`. Uma revisão implica o incremento de `Z`. A versão atual deste documento é 2.0.0.

Agradecimentos

Gostaria de manifestar o meu agradecimento a algumas pessoas, sem as quais este meu percurso não teria sido possível ou, pelo menos, seria mais difícil.

Ao meu orientador, Professor Doutor Bruno Cabral, por me proporcionar a oportunidade de integrar um projeto inédito para a Universidade Coimbra, numa das minhas áreas prediletas.

Aos meus familiares e amigos, por todo o apoio e incentivo prestados na conquista de um dos objetivos mais importantes na minha vida.

Aos colegas da equipa UC-num, pelo seu companheirismo e interajuda durante todo o processo de estágio que fizeram de nós uma equipa coesa, motivada e apta a continuar este trabalho.

As minhas últimas palavras vão para os meus pais, a quem agradeço muito todo o carinho, apoio e motivação que me deram ao longo da vida. Obrigada ainda pelos esforços que fizeram para me proporcionar a formação e educação que sempre desejei. Sem eles não seria quem sou hoje e este trabalho não seria possível.

Ana Ilharco
Coimbra, setembro de 2015

Resumo

Com o aumento do volume de informação e dos recursos das instituições do ensino superior, a tomada de decisões de gestão tem-se revelado uma tarefa árdua e complexa. Tanto a equipa reitoral, como o conselho de gestão, conselho geral e diretores das unidades orgânicas da *Universidade de Coimbra* **(UC)** necessitam, cada vez mais, de ter ao seu alcance um conjunto de indicadores de performance (*KPIs*). A nível interno, tais indicadores são definidos no Plano Estratégico e de Ação (PEA), estabelecido até 2015, e abrangem diversas áreas, das quais se destaca a Económico-Financeira. Para além destes, existem ainda outros indicadores mais específicos dessa área que, embora não figurem no Plano Estratégico da UC, são absolutamente indispensáveis para que a equipa reitoral acompanhe a execução do modelo de distribuição orçamental aprovado em conselho de gestão. Todos estes indicadores permitem avaliar e monitorizar a instituição. O problema que se coloca é na obtenção dos seus valores: o cálculo é feito manualmente pelas equipas operacionais de cada área, tornando-se, por conseguinte, uma tarefa morosa e propensa a erros. Com vista a eliminar essas falhas, o presente estágio tem por fim desenvolver um *data mart* (subconjunto de uma *Data Warehouse* que representa uma área funcional específica) onde seja possível calcular indicadores da área Económico-Financeira, em particular os da área de Gestão Orçamental. A criação de um modelo multidimensional para o *data mart* vem permitir uma performance elevada na consulta de grandes volumes de dados, o que se torna vantajoso para, a posteriori, se realizar uma análise OLAP (*Online Analytical Processing*) sobre a informação armazenada. Essas análises são, então, disponibilizadas aos utilizadores finais sob a forma de *dashboards* interativos e intuitivos, disponíveis numa aplicação *web* já desenvolvida para o efeito, embora no âmbito de outras áreas.

Palavras-chave: *Business Intelligence*, *Data warehouse*, *ETL*, *OLAP*, *staging area*, *Dashboards*, *Data Mart*, *Indicadores de Performance (*KPI*), Análise de dados.*

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ACID	Atomicity Consistency Isolation Durability
BD	Base de Dados
BI	Business Inteligence
CDE	Community Dashboards Editor
CDF	Community Dashboards Framework
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DBMS	Database Management Systems
DCF	Divisão de Contabilidade Financeira
DOC	Divisão de Orçamento e Conta
DPGD	Divisão de Planeamento, Gestão e Desenvolvimento
DW	Data Warehouse
ETL	Extraction, Transforming and Loading
HTML	HyperText Markup Language
JS	JavaScript
KPI	Key Performance Indicators
LDAP	Lightweight Directory Access Protocol
MDX	Multidimensional Expressions
OLAP	Online Analytical Processing
PEA	Plano Estratégico e de Ação

PSE Prestação de Serviços Especializados

RDB Relational Database

RDBMS Relational Database Management Systems

SAMA Sistema de Apoio à Modernização Administrativa

SAP Systeme, Anwendungen und Produkte in der Datenverarbeitung

SAS Statistical Analysis System

SASUC Serviços de Ação Social da UC

SG Sistema de Gestão

SGBD Sistemas de Gestão de Bases de Dados

SGF Serviço de Gestão Financeira

SGSIIC Serviço de Gestão de Sistemas e Infraestruturas de Informação e Comunicação

SI Sistema de Informação

SO Sistema Operativo

SOAP Simple Object Access Protocol

SQL Standard Query Language

TIC Tecnologias de informação e Comunicação

UC Universidade de Coimbra

UECAF Unidades de Extensão Cultural e de Apoio à Formação

UO Unidade Orgânica

URL Uniform Resource Locator

XML Extensible Markup Language

Capítulo 1

Introdução

O presente documento tem como principal objetivo apresentar o trabalho realizado pela aluna de dissertação de Mestrado em Engenharia Informática, durante o ano letivo 2014/2015, no âmbito do projeto "UC-num", orientado pelo Professor Doutor Bruno Cabral.
O estágio realizou-se nas instalações dos serviços NÓNIO, atualmente alojados no Colégio de Jesus da Universidade de Coimbra.

1.1 Enquadramento

Com o aumento do volume de informação e dos recursos das instituições do ensino superior, a tomada de decisões de gestão tem-se revelado uma tarefa árdua e complexa para as mesmas. A UC não é exceção e, foi com base nessa premissa que a equipa reitoral decidiu aprovar um projeto que visa a consolidação e melhoria dos Sistema de Apoio à Modernização Administrativa, projeto *Sistema de Apoio à Modernização Administrativa* **(SAMA)** – que pretende melhorar as suas infraestruturas e serviços TIC em áreas bem definidas da universidade: infraestruturas de suporte a serviços, integração e extensão de sistemas de gestão, integração com plataformas de contratação pública, indicadores para a gestão, sistemas de pagamentos eletrónicos e monitorização da estratégia da UC e do seu desdobramento.

Tanto a equipa reitoral, como os elementos do conselho de gestão, diretores das *Unidade Orgânicas* **(UOs)** e das *Unidades de Extensão Cultural e de Apoio à Formação* **(UECAF)** e coordenadores de curso necessitam de ter ao seu

alcance um conjunto de indicadores de performance (*Key Performance Indicators* **(KPI)**) que os auxilie, a qualquer momento, na tomada de decisão. Atualmente, e apesar da forte informatização dos serviços, a recolha desses indicadores de desempenho da UC é realizada nos sistemas operacionais de suporte a cada área (gestão financeira e orçamental, gestão académica, recursos humanos, investigação científica, etc.), o que constitui um processo moroso, que obriga a uma complexa validação da integridade dos dados, acabando por se tornar um obstáculo na gestão eficaz de meios e a uma otimização da qualidade dos serviços. O desenvolvimento de um sistema de *data warehouse* para suportar a recolha, tratamento e apresentação dos dados torna-se, assim, uma necessidade premente [31].

Nesse contexto, propôs-se a criação do projeto UC-Num, que se enquadra dentro de um projeto maior, UC-cloud (que por sua vez está englobado no SAMA – ver Figura 1.1), que visa o desenvolvimento de um sistema de apoio à decisão que permita a monitorização dos indicadores de desempenho, muitos deles já definidos no *Plano Estratégico e de Ação* **(PEA)** 2011-2015 da UC.
Este projeto está atualmente dividido em oito módulos: sucesso escolar, custos

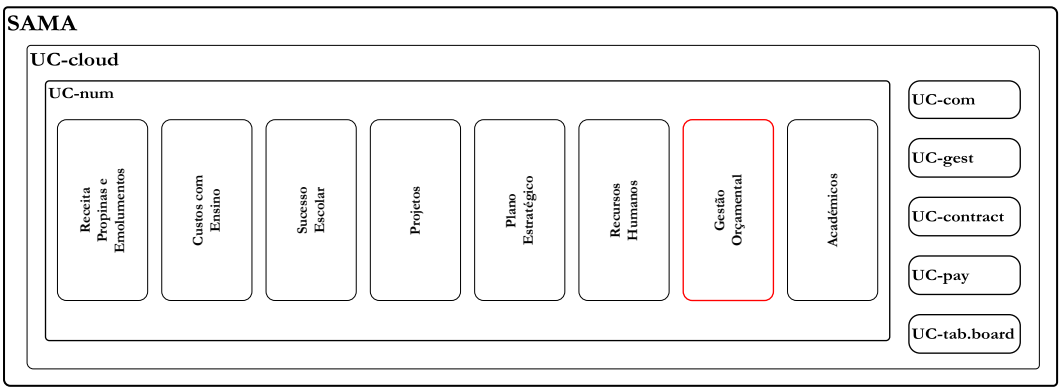


Figura 1.1: Enquadramento do projeto UC-num no projeto UC-Cloud: módulo a desenvolver destacado.

com ensino, receitas com propinas e emolumentos, projetos, plano estratégico, gestão orçamental, académicos e recursos humanos, sendo que os últimos três dizem respeito aos que estão a ser desenvolvidos pelos três elementos a realizar estágio curricular. Os restantes módulos correspondem quer a módulos desenvolvidos no ano anterior, quer à adição ou extensão de módulos pelos restantes membros desta equipa.

A presente dissertação enquadra-se no módulo de Gestão Orçamental.
A equipa UC-num é atualmente constituída por sete elementos, três dos quais se encontram a realizar estágio curricular ao abrigo deste projeto.

1.2 Contexto atual

O *Sistema de Gestão* **(SG)** da UC envolve uma série de documentos, procedimentos e vários sistemas de informação de apoio à gestão administrativa dos processos e à sua monitorização - p.ex. Lugus, NÓNIO e SAP [32]. Este último inclui um módulo, SAP-FI, que envolve dados essenciais à gestão financeira de todos os recursos da universidade e é, por isso, de extrema importância para o cálculo de indicadores dessa área, a cargo do *Serviço de Gestão Financeira* **(SGF)** da UC que inclui outras subdivisões, das quais se destaca, com particular importância para este módulo, a *Divisão de Orçamento e Conta* **(DOC)**.
Não obstante as elevadas capacidades daquele sistema, as equipas operacionais têm, todos os anos e com alguma frequência, de efetuar cálculos manuais, geralmente em ficheiros *Excel*, recolhendo os dados registados em SAP. Esta tarefa exige tempo e é propensa a erros, tornando-se cada vez mais difícil obter os indicadores de forma rápida, eficiente e fiável. Além disso, no caso da Gestão Orçamental, os relatórios produzidos variam o seu grau de detalhe consoante os *stakeholders* a quem se destinam (e.g., para alguns deles fará sentido mostrar uma visão geral da UC, para outros já pode ser necessário mostrar detalhe ao nível das unidades/serviços).
Como a questão do tempo é suprema na tomada de decisão, a UC poderá estar em desvantagem se não apresentar um sistema capaz de consolidar os dados de outros sistemas e de providenciar uma visão única de todos os processos de negócio nela modelados.

Na secção seguinte são abordados e clarificados os objetivos desta dissertação.

1.3 Objetivos e contribuições

Por forma a colmatar as falhas identificadas na secção anterior, os objetivos desta dissertação passam por:

- Desenvolver um *data mart*, subconjunto da *Data Warehouse* **(DW)** do

UC-num, relativo à área Económico-Financeira, em particular à área de Gestão Orçamental.

- Disponibilizar os indicadores de desempenho através de uma análise OLAP, responsável pela manipulação e análise de um grande volume de dados sob múltiplas perspetivas, sob a forma de gráficos interativos, integrando-os na aplicação *web*, previamente desenvolvida pela equipa anterior, que aloja o projeto UC-num[1].

Portanto, trata-se de um projeto que integra o âmbito de *Business Inteligence* **(BI)** e, como tal, torna-se indispensável identificar as questões mais pertinentes sob o ponto de vista do negócio, tais como:

- *Qual o orçamento disponível para a UC e suas unidades/serviços? Em que percentagens se distribui a nível Estrutural, Desenvolvimento e de Projetos e Atividades?*
- *Qual o nível de diversificação da estrutura de financiamento da UC?*
- *Qual o* ratio *entre despesa executada e a receita arrecadada em determinados anos/meses/semestres?*
- *Qual o grau de contribuição dos fundos públicos na receita arrecadada pela UC e pelos SASUC?*
- *Que tipo de despesa é mais significativa para a UC: Pessoal, Funcionamento ou Capital?*
- *Qual a taxa de crescimento do financiamento competitivo no primeiro semestre face ao período homólogo do ano anterior?*
- *Quais os compromissos assumidos para um determinado ano?*
- *Que Unidades Orgânicas têm maior valor de faturas a cobrar a terceiros?*
- *Qual o peso do autofinanciamento no financiamento total da UC?*

Estas são apenas algumas das muitas questões contempladas neste módulo. Para lhes dar resposta, é preciso, então, definir um conjunto de indicadores que, de forma muito sucinta, se dividem em cinco grupos distintos: orçamento, despesa, receita arrecadada, receita distribuída e PEA, e que são de extrema importância para a UC e, em alguns casos, também para os *Serviços de Ação Social da UC* **(SASUC)**.

[1] *Webpage* do UC-num disponível (acesso restrito) em: `https://dw.uc.pt/`

Mas definir indicadores não basta – é preciso também contemplar as restantes componentes que fazem parte de uma solução deste tipo. Em primeiro lugar, é preciso definir e desenvolver o processo de *Extraction, Transforming and Loading* **(ETL)**, responsável pelo carregamento dos dados dos sistemas fonte, transformação e posterior carregamento dos mesmos para um *data mart*, para o qual se desenhou um modelo de dados muito específico para *DW*: o modelo de dados multidimensional ou em estrela (para o qual se destaca o importante contributo de *Ralph Kimball* [7, 16]). De seguida, definem-se os cubos OLAP[2], por forma a facilitar e tornar mais eficaz a análise OLAP. Por fim, os resultados dessas análises são mostrados aos utilizadores finais sob a forma de gráficos disponibilizados através de uma aplicação *web* desenvolvida anteriormente.

O mercado de *Business Inteligence* tem evoluído muito nos últimos anos. Existem vários produtos no mercado, mas o mais comum é encontrar soluções distintas, consoante a área a que concernem: gestão financeira, educação, saúde, transportes, recursos humanos, etc, sendo personalizáveis à medida de cada caso. As instituições eram ou são, geralmente, aliciadas a adquirir este tipo de produtos. Contudo, dada a atual conjuntura económica, a aquisição de produtos comerciais deixou de ser viável. Consequentemente, um dos objetivos deste projeto passou também por recorrer a tecnologias e ferramentas gratuitas.

Concluindo, o projeto UC-num é um projeto inédito que visa disponibilizar aos principais *stakeholders* da UC os números de apoio à gestão e tomada de decisões nas mais diversas áreas, retirando essa carga das várias equipas e sistemas operacionais, ao oferecer uma solução de *BI*, dinâmica, intuitiva e de fácil utilização.

1.4 Estrutura do documento

No Capítulo 2 referem-se a metodologia usada na gestão e desenvolvimento do projeto e o planeamento de trabalho para ambos os semestres.

O Capítulo 3 descreve o processo de levantamento de requisitos e a especificação dos requisitos funcionais e não funcionais do sistema.

No Capítulo 4 apresenta-se a arquitetura do sistema, bem como a análise e

[2]Definição de cubo OLAP na página 28.

seleção de tecnologias a usar em cada uma das suas componentes. São também apresentados os modelos de dados da área temporária e do *data mart*.

O Capítulo 5 apresenta os detalhes mais relevantes da implementação deste módulo.

O Capítulo 6 expõe a metodologia dos testes efetuados ao módulo desenvolvido e o respetivo processo de validação.

Por fim, no Capítulo 7 encontram-se algumas conclusões sobre o trabalho desenvolvido e menciona-se o trabalho futuro.

Capítulo 2

Metodologia e Planeamento

Este capítulo apresenta a metodologia e as ferramentas de trabalho usadas durante o desenvolvimento, bem como expõe todas as atividades realizadas ao longo do ano letivo, para além de ser complementado com a análise de riscos presente no Anexo A (página 89).

2.1 Metodologia de Desenvolvimento

A Figura 2.1 representa o processo de desenvolvimento de soluções de BI adotado neste projeto, com base na metodologia proposta por *Ralph Kimball* [17] – considerado um dos precursores dos conceitos de *Data Warehousing* e *Business Intelligence*, e que aplicou, com sucesso, esta metodologia a dezenas de projetos.

Na primeira fase do estágio foi feito um planeamento do projeto, com os sete elementos da equipa e com o orientador, dando especial atenção à definição dos principais *milestones* a atingir pelos elementos que se encontravam a realizar estágio curricular.
Além do planeamento geral, esta fase envolveu as seguintes etapas: análise e especificação de requisitos (Capítulo 3), desenho da arquitetura e seleção de produtos, especificação da aplicação analítica e, por fim, modelação multidimensional.

Durante a segunda fase terminou-se o desenho da área temporária e realizou-se o desenvolvimento da mesma, desenvolveu-se a análise OLAP que cul-

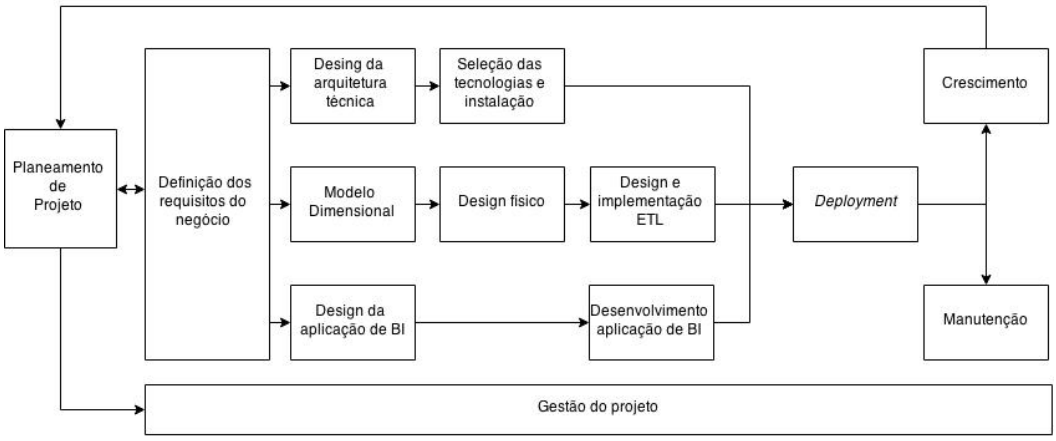


Figura 2.1: Metodologia de desenvolvimento adotada.

minou com a disponibilização dos *dashboards* apresentados na página web (`https://dw.uc.pt/`). Uma vez testados e validados, foi feito o *deployment* da aplicação junto dos utilizadores, por fases.
Atualmente o projeto encontra-se na fase de crescimento e também de manutenção.

2.2 Metodologia e ferramentas de trabalho

Na primeira reunião, antes do início oficial dos estágios curriculares, acordou-se com a periodicidade semanal das reuniões de grupo. Estas reuniões, onde se encontravam presentes os sete elementos responsáveis pelo desenvolvimento, o orientador do estágio e, por vezes, o Eng. Pedro Pinto do projeto NÓNIO (como auxiliar dos módulos académico e recursos humanos), serviram para atribuir tarefas e ir fazendo o seu acompanhamento de forma pontual. Devido ao grande número de reuniões ocorridas, sobretudo durante o primeiro semestre, tornou-se premente a partilha da calendarização entre todos os elementos do grupo, neste caso, via *Google Calendar*.

Como ajuda complementar, recorreu-se a algumas ferramentas, tais como a plataforma *redmine* [19]– uma aplicação web, gratuita, direcionada para a gestão de projetos, embora esta tenha servido apenas para gerir o projeto a nível macro durante o segundo semestre. Tal aconteceu, porque, na segunda fase, a equipa decidiu recorrer a uma ferramenta que ia mais de encontro às suas

necessidades de gestão de projeto – a ferramenta *Taiga*[1] – que permite fazer uma gestão mais micro, recorrendo à definição de *sprints* de desenvolvimento, com várias etapas, permitindo também ter um *Kanban board*, *backlogs* e identificação de *issues*, entre outras caraterísticas que o tornaram indispensável para conseguir uma gestão de projeto mais minuciosa e integrante.

Para uma eficiente gestão do código a desenvolver pelos elementos da equipa no segundo semestre e para assegurar o controlo e estruturação de versões, recorreu-se ao *GitLab* [13], uma plataforma de gestão de repositórios *Git* [28], alojada no servidor de integração e testes. Tanto as máquinas de desenvolvimento, como as máquinas de integração e testes e a de produção recorrem ao *Git*, sendo que nestas duas últimas apenas é feito o *commit* de versões estáveis do trabalho, por forma a garantir o bom funcionamento e integração de todos os módulos, sem risco de inviabilizar o que já havia sido feito.

2.3 Planeamento

O planeamento feito para o primeiro semestre envolveu, por ordem cronológica, a realização das seguintes tarefas: pesquisa e elaboração do estado da arte, pesquisa de metodologia de testes, testes sobre os módulos já desenvolvidos, definição de indicadores do módulo, criação de protótipos de *dashboards* e, finalmente, desenho dos modelos de dados do *data mart* deste módulo e o da área temporária.

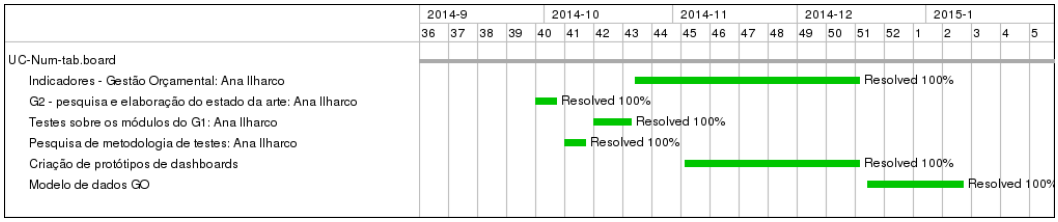


Figura 2.2: Diagrama de *Gantt* para o 1º Semestre (retirado do *redmine*).

Conforme se pode verificar pela Figura 2.2, o planeamento para o primeiro semestre foi cumprido. Note-se que a introdução da etapa *Pesquisa de Metodologia de testes* serviu não só para preencher uma lacuna temporal devido ao atraso da reunião de *kick-off*, mas também para preparar a fase final de

[1]Página oficial do *Taiga* disponível em: `https://taiga.io/`

validação e testes.

Relativamente ao desenvolvimento no segundo semestre este assentou em três pilares: implementação, testes de integração e validação. A abordagem foi cíclica, i.e., cada uma das três etapas foi realizada, por aquela ordem, para um conjunto de indicadores de um determinado âmbito e repetida para os restantes. Ou seja, existiriam tantas iterações quantos os níveis de prioridade dos indicadores – elevada, média e baixa – sendo que, no caso do presente módulo, todos os indicadores são de prioridade elevada, pelo que, efetivamente, ocorreu apenas uma iteração, dividida pelos diferentes âmbitos que o constituem.
De forma resumida, o trabalho do último semestre resumiu-se ao desenvolvimento dos requisitos propostos e respetiva validação de acordo com o planeamento macro presente na Figura 2.3.

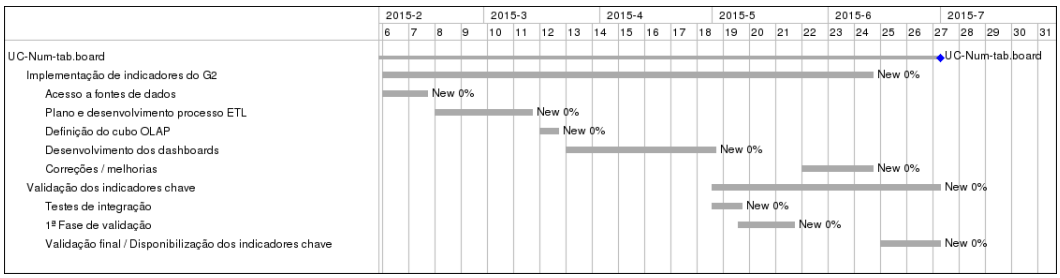


Figura 2.3: Diagrama de *Gantt* com planeamento macro para o 2º Semestre (retirado do *redmine*).

A Figura 2.4 diz respeito ao planeamento micro efetivamente levado a cabo no segundo semestre. Este planeamento teve por base o planeamento macro previamente apresentado e a sua execução passou por despender mais tempo para o primeiro âmbito (orçamento, neste caso), repetindo as mesmas tarefas para os restantes âmbitos, tarefas essas que incluíram: acesso aos dados dos sistemas fonte, processo de ETL, criação de cubos OLAP, desenvolvimento de *dashboards* do âmbito, testes de integração e funcionais e validação pelos *stakeholders*.

Em ambas as fases o planeamento sofreu alguns ajustes. No primeiro semestre, os atrasos estiveram relacionados com:

- Adiamento da reunião de *kick-off*, que atrasou o início da especificação de requisitos, dado que as equipas operacionais designadas para o efeito só foram conhecidas aquando dessa reunião;

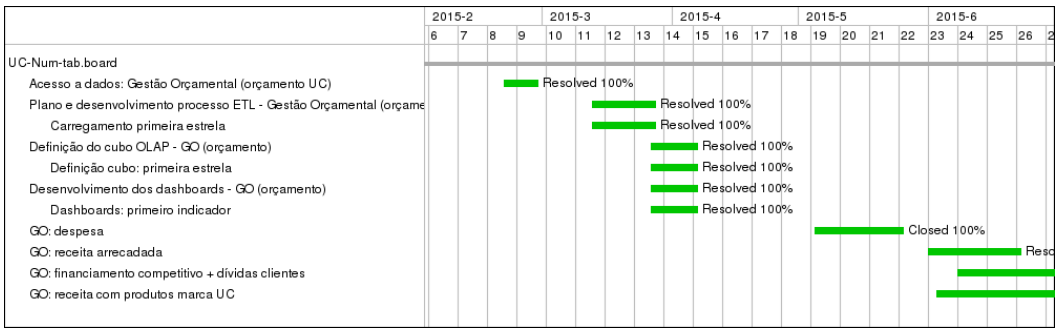


Figura 2.4: Diagrama de *Gantt* com planeamento micro para o 2º Semestre (retirado do *redmine*).

- Processo de especificação dos requisitos funcionais, com todos os elementos envolventes: foi o processo mais moroso nesta fase, dado que envolveu pessoas com *backgrounds* bastante diferentes e, também, por serem em número elevado, o que dificultou a sincronização de disponibilidades para reuniões.

No segundo semestre, os atrasos deveram-se sobretudo a dois fatores:

- Atraso na disponibilização de *webservices* (competência do elemento dos *Serviço de Gestão de Sistemas e Infraestruturas de Informação e Comunicação* **(SGSIIC)**, designado para auxiliar o presente módulo;
- Curva de aprendizagem acentuada relativamente à linguagem MDX, o que gerou atrasos no desenvolvimento dos *dashboards*;
- *Webservices* com especificação insuficiente ou ainda não fechada/validada pelo grupo operacional.

O primeiro ponto fez com que o desenvolvimento só pudesse ser iniciado em meados de março. Posteriormente, os *dashboards* do âmbito do orçamento e despesa, sofreram mais com a falta de experiência e, por esse motivo, demoraram cerca de dois meses a serem concluídos. Os restantes dois meses (junho e julho) foram dedicados aos outros quatro âmbitos.
Por fim, o último ponto, alheio à equipa UC-num, impossibilitou o desenvolvimento de apenas um âmbito (receita distribuída), razão pela qual não figura no diagrama da Figura 2.4, e de dois indicadores do equilíbrio orçamental que deveriam estar presentes no âmbito da despesa.

Todo o planeamento foi (re)definido por todos os elementos da equipa, tendo sempre em conta o *feedback* e sugestões dos colegas que desenvolveram os módulos no ano anterior. Esta abordagem cíclica mostrou-se fiável e indispensá-

vel, na medida em que permitiu ir efetuando a integração parcial dos diferentes módulos. Tal permitiu ir ultrapassando as falhas de modo incremental, em vez de condensar tudo num único momento final de integração, contribuindo assim para a qualidade da aplicação final disponibilizada.

Não obstante os atrasos apontados, todos os objetivos passíveis de serem cumpridos, foram-no, conforme comprova a Figura 2.4. De notar que a validação final (que não é da competência desta equipa) ainda não foi fechada, porque se encontra ainda a decorrer por parte das equipas operacionais de cada módulo. O desejável era que cada âmbito tivesse sido validado, à medida que iam sendo disponibilizados, mas tal não foi possível de acontecer em nenhum dos módulos.

2.4 Sumário

Escolher uma metodologia de desenvolvimento e ferramentas de trabalho adequadas são duas peças importantes na organização e estruturação do desenvolvimento de software. Nesse sentido, decidiu-se seguir a metodologia de desenvolvimento proposta por *Ralph Kimball* [17], uma vez que esta tem sido largamente aplicada em projetos desta natureza. Relativamente à metodologia de trabalho aplicada, consistiu em reuniões semanais de orientação de estágio, reuniões com o grupo operacional e *stakeholders*, sobretudo no primeiro semestre, durante a fase de especificação de requisitos – etapa mais morosa durante o primeiro período de estágio – além de reuniões semanais de equipa para fazer um ponto de situação.

O uso de ferramentas como o *Git*, *Taiga*, *redmine* e *Google Calendar*, facilitaram toda a organização e gestão do projeto, destacando-se o *redmine* para planeamento macro e o *Taiga* para um planeamento a um nível mais micro.
No segundo período de estágio, é de destacar a abordagem iterativa de desenvolvimento que permitiu que o mesmo acontecesse de forma incremental e, consequentemente, os *dashboards* fossem sendo disponibilizados à medida que ficavam concluídos.

Apesar de terem surgido alguns contratempos que obrigaram a um reajustamento do planeamento em ambos os semestres, todos os objetivos foram cumpridos.

Capítulo 3

Definição de Requisitos

Este capítulo descreve o processo de levantamento e especificação de requisitos (funcionais e não funcionais).

3.1 Levantamento de Requisitos

O levantamento de requisitos serve para entender qual o âmbito do projeto, qual será o produto final e as funcionalidades que se esperam do mesmo. É, por isso, uma das etapas mais importantes do processo que culminará com o desenvolvimento de um sistema e, como tal, não deve ser negligenciada.
É comum os utilizadores não terem uma ideia precisa do que pretendem ou não conseguirem transmitir o seu conhecimento do domínio do problema ao analista/programador. Por outro lado, este também pode falhar, não sendo capaz de descrever os requisitos do sistema de modo objetivo, sem ambiguidades, conciso e consistente com aquilo que foi proposto.

Nesse sentido, foram realizadas várias reuniões até ao início do segundo semestre com a equipa operacional principal, a constituída por elementos da DOC, e algumas com a equipa operacional da *Divisão de Planeamento, Gestão e Desenvolvimento* **(DPGD)**, sempre na presença de um elemento da equipa do SGSIIC da UC, responsável pelo desenvolvimento e disponibilização de *web-services* com os dados de SAP. Para além dessas reuniões em equipa, existiu ainda uma primeira reunião de *kick-off* do projeto UC-Num no final de outubro de 2014, conduzida pela vice-reitora Professora Doutora Margarida Mano e com a presença de vários outros *stakeholders* dos quais de destacam, com

especial interesse para o módulo de Gestão Orçamental, o chefe da DPGD, Dr. Filipe Rocha, o director do SGF, Dr. Sérgio Vicente, o chefe da DOC, Dr. Nuno Patão e o consultor externo, Dr. José Morais. Tal reunião serviu não só para divulgar os novos módulos do projeto, mas também para se dar a conhecer todos os presentes e o papel que cada um desempenharia nos diferentes módulos (quer os já desenvolvidos, quer os que surgiram nesta nova fase).

Pelo *feedback* transmitido pela anterior equipa do UC-Num, tornou-se evidente que seria vantajoso prosseguir com o desenvolvimento de protótipos rápidos nesta fase, por forma a evitar ambiguidades entre as várias partes envolvidas e diminuir o risco de ter que efetuar alterações aquando do desenvolvimento. Esta abordagem visa mostrar aos utilizadores finais as funcionalidades que se pretendem implementar e a forma como se interage com as mesmas. Assim, foram criados vários ecrãs rápidos simulando o resultado final do *site* do projeto, sendo estes alterados de acordo com as indicações da equipa operacional e/ou dos *stakeholders*.

Os protótipos rápidos (exemplo na Figura 3.1) não só foram a base de toda a validação de indicadores e a forma como os mesmos seriam contemplados na aplicação final, mas também auxiliaram na validação dos próprios modelos de dados.

3.2 Especificação de Requisitos

O modelo FURPS, inicialmente proposto por *Robert Gradu* da *Hewlett-Packard* em 1992 [14], é um modelo muito simples usado na especificação de requisitos em engenharia de software. Mais tarde, foi estendido pela *IBM Rational Software* para FURPS+, que considera que os requisitos da aplicação se dividem em duas categorias:

- funcionais (**F**URPS): (do inglês, ***F**unctional*), são requisitos que, quando satisfeitos, permitem ao utilizador obter o *output* esperado face a um determinado *input*;
- não-funcionais (F**URPS+**), cujo acrónimo, em inglês, advém das categorias ***U**sability* (usabilidade), ***R**eliability* (fiabilidade), ***P**erformance* (*idem*) e ***S**upportability* (suporte), são requisitos geralmente significantes sob o ponto de vista da arquitetura. O símbolo '+' permite identificar

categorias adicionais que representam limitações ou restrições adicionais quer ao nível da implementação, quer ao nível do *design*, da interface ou mesmo físicos/de *hardware* [9].

O primeiro passo da aplicação deste modelo é, tipicamente, o estabelecimento da prioridade de requisitos. Nesta aceção, a equipa anterior, juntamente com os *stakeholders* do projeto, acordou com os três níveis de prioridade: elevado, médio e baixo. Na equipa atual, decidiu manter-se estes níveis de prioridades cuja explicação se encontra na Tabela 3.1.

Prioridade	**Definição**
Elevada	Requisito imprescindível para o projeto. Caso não seja cumprido, comprometerá a concretização da aplicação. É, por isso, um tipo de requisitos que se desenvolve em primeiro lugar e cuja importância não pode ser menosprezada.
Média	Requisito não fundamental ao sistema mas que, quando satisfeito, valoriza a aplicação final. O seu incumprimento não compromete de modo algum as funcionalidades chave da aplicação.
Baixa	Requisito que acrescenta valor à aplicação, mas que só deve ser satisfeito caso o seu desenvolvimento seja viável sob o ponto de vista do *budget* (tempo, neste caso) do projeto. Quando não implementado, costuma servir de recomendação para versões posteriores.

Tabela 3.1: Níveis de prioridade dos requisitos.

No que diz respeito à especificação dos requisitos, existe uma convenção de nomenclatura que foi adotada, por questões de coerência de organização e facilidade de referência. Essa nomenclatura adotada é descrita na Tabela 3.2.

Código	Descrição
RF_[x+]_000	Requisito funcional, onde *x+* corresponde à categoria e 000 ao número identificador; Categorias: GE - Gerais e R - categoria "Relatórios". Exemplo: Exemplo: RF_GE_001, requisito funcional número um da categoria "Gerais".
RNF_[y+]_000	Requisito não funcional, onde *y* representa a categoria e 000 o número identificador; Categorias FURPS+: U – Usabilidade: R – fiabilidade; P – performance; S – suporte e manutenção; O – outros (segurança, restrições de implementação, design e interface, hardware, . . .).
IND_[x+]_000	Requisito funcional da categoria de indicadores, onde x+ corresponde ao módulo de desenvolvimento e 000 ao número identificador. Exemplo: IND_GO_001, indicador número um do módulo de Gestão Orçamental. Identificação dos módulos de desenvolvimento: AC - ensino; RH - pessoas; GO - gestão orçamental e financeira; PR- projetos; PES - plano estratégico; SE - sucesso escolar; PE - propinas e emolumentos; CE - despesas com ensino.

Tabela 3.2: Nomenclatura para requisitos funcionais e não funcionais.

3.2.1 Requisitos funcionais

Relativamente às funcionalidades da aplicação, distinguem-se duas categorias: gerais e indicadores. A primeira, como o próprio nome sugere, diz respeito a funcionalidades comuns a toda a aplicação (Tabela 3.3).

Código	Designação	Prioridade	Descrição sucinta
RF_GE_001	Autenticação	Elevada	A aplicação deve permitir ao utilizador a autenticação (*login*) através das credenciais utilizadas no acesso a quaisquer serviços disponibilizados à comunidade da UC (email da UC e password). A autenticação tem de ser bem sucedida, isto é, as credenciais têm de ser válidas para que seja permitida qualquer visualização de dados ao utilizador.
RF_GE_002	Fechar sessão	Elevada	O utilizador pode, após autenticação, efetuar o término da sua sessão (*logout*).
RF_GE_003	Término de sessão	Elevada	Para garantir a segurança da aplicação, um utilizador, depois de autenticado, terá associada uma sessão. Esta deve ter um *timeout* para efetuar *logout* automaticamente.

Tabela 3.3: Breve descrição dos requisitos funcionais: gerais (cont.).

Código	Designação	Prioridade	Descrição sucinta
RF_GE_004	Navegação entre módulos	Elevada	O utilizador deve conseguir aceder a todos os restantes módulos, este acesso deve ser possível a qualquer momento.
RF_GE_005	Navegação interna	Elevada	A aplicação deve permitir ao utilizador efetuar *drill down* e *roll up* nos dados que pretende visualizar e ir atualizando a barra de navegação estrutural (*breadcrumbs*) de acordo com os seguintes níveis de hierarquia: 1. Grupo UC[1]; 2. Unidades do Grupo UC (UC e SASUC[2]); 3. Unidades Orgânicas da UC; 4. Departamentos na UO; 5. Serviços do nível I do departamento; 6. Serviços do nível II do serviço do nível I[3].
RF_GE_006	Parâmetros gerais	Elevada	O utilizador deve ter disponível os diversos parâmetros (seleção de indicadores, agregadores e/ou filtros) que são permitidos modificar nos dados que este está a visualizar.
RF_GE_007	Parâmetros temporais	Elevada	Deve ser permitido ao utilizador modificar os parâmetros temporais (dimensão tempo) a aplicar nos dados que este está a visualizar.
RF_GE_008	Esconder parâmetros	Baixa	Deve ser permitido ao utilizador esconder a barra onde se encontram os parâmetros gerais e de tempo.
RF_GE_009	Secção de ajuda	Elevada	A aplicação deve disponibilizar uma secção de ajuda ao utilizador, transversal a todos os módulos, para esclarecer quaisquer dúvidas que sejam suscitadas nos utilizadores - FAQ.

Tabela 3.3: Breve descrição dos requisitos funcionais: gerais (cont.).

[1]Este nível e o seguinte foram sugeridos por alguns *stakeholders*. Não se aplica a todos os indicadores.

[2]Segundo o próprio, não será desagregado pelas suas Unidades – é considerado como um todo.

[3]A pedido do grupo operacional, o último nível não deverá ser, para já, mostrado ao utilizador final.

Código	**Designação**	**Prioridade**	**Descrição sucinta**
RF_GE_010	Informação auxiliar	Elevada	Cada vista de dados disponibilizada ao utilizador (gráfico ou tabela) deve ser acompanhada de dois mecanismos que permita consultar informação: uma referente aos dados que são apresentados, corresponde a um botão de informação (remetendo para a secção de ajuda) e outra de navegação (sugestões, erros, etc).
RF_GE_011	Visualização: gráfico ↔ tabela	Elevada	A aplicação deve permitir ao utilizador visualizar a informação apresentada num gráfico em formato de tabela e vice-versa. Os dados apresentados na tabela deverão, pelo menos, corresponder à informação que se encontra no gráfico.
RF_GE_012	Exportar informação da tabela/-gráfico	Baixa	Exportar para formato *Excel* ou *CSV* a informação presente nas tabelas de análise. Exportar gráfico como imagem.
RF_GE_013	Autorização	Elevada	O utilizador após autenticação, pode visualizar a informação afeta ao(s) módulo(s) em que está inserido. Por exemplo: se um utilizador pertencer ao grupo DW_SE deve conseguir aceder a todo o módulo de sucesso escolar.
RF_GE_014	*Zoom* de gráficos	Baixa	Efetuar zoom *in* e *out* de um dos gráficos.

Tabela 3.3: Breve descrição dos requisitos funcionais: gerais (fim).

Os requisitos da segunda categoria, indicadores, exprimem as funcionalidades específicas do módulo de Gestão Orçamental que devem estar disponíveis para os utilizadores finais, i.e., correspondem aos indicadores de desempenho esperados para o presente módulo. Na Tabela 3.4 encontra-se uma breve descrição de cada um dos requisitos definidos para o presente módulo.

Código	**Designação**	**Prioridade**	**Descrição sucinta**
IND_GO_001	Orçamento do ano	Elevada	Deve ser possível fazer *drill down* e *roll up* segundo os níveis apresentados no RF_GE_005. É possível agregar e/ou filtrar quanto ao tipo de orçamento e tipo de despesa. Nos parâmetros temporais, permitir efetuar análise apenas e só cumulativa sobre um período de anos ou meses económicos selecionados.

Tabela 3.4: Breve descrição dos requisitos funcionais: indicadores (cont.).

Código	Designação	Prioridade	Descrição sucinta
IND_GO_002	Saldo de gerência	Elevada	(Igual à do indicador IND_GO_001).
IND_GO_003	Orçamento disponível	Elevada	(Igual à do indicador IND_GO_001).
IND_GO_004	Cabimentos	Elevada	Deve ser possível fazer *drill down* e *roll up* segundo os níveis apresentados no RF_GE_005. É possível agregar e/ou filtrar quanto ao tipo de orçamento e tipo de despesa. Nos parâmetros temporais, opção de selecionar (ou não) análise cumulativa sobre um período de anos, semestres ou meses económicos selecionados.
IND_GO_005	Cabimentos face ao orçamento disponível	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_006	Compromissos	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_007	Compromissos face aos cabimentos	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_008	Despesa processada	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_009	Despesa processada face aos compromissos	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_010	Despesa executada	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_011	Despesa executada face ao orçamento disponível	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_012	Despesa executada face aos cabimentos	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_013	Compromissos por pagar	Elevada	(Igual à do indicador IND_GO_004).

Tabela 3.4: Breve descrição dos requisitos funcionais: indicadores (cont.).

Código	**Designação**	**Prioridade**	**Descrição sucinta**
IND_GO_014	Taxa de crescimento da despesa executada	Elevada	(Igual à do indicador IND_GO_004).
IND_GO_015	Receita arrecadada	Elevada	Deve ser possível fazer *drill down* e *roll up* segundo os níveis apresentados no RF_GE_005. É possível agregar quanto à origem da receita e filtrar por tipo de fundos públicos e/ou por tipo de propinas, sempre que aplicável. Nos parâmetros temporais, opção de selecionar (ou não) análise cumulativa sobre um período de anos, trimestres, semestres ou meses económicos selecionados.
IND_GO_016	Peso da receita própria no financiamento total	Elevada	Deve ser possível fazer *drill down* e *roll up* até ao nível 2. apresentado no RF_GE_005. Não tem filtros, nem agregadores. Nos parâmetros temporais, opção de selecionar (ou não) análise cumulativa sobre um período de anos, semestres ou meses económicos selecionados.
IND_GO_017	Receita distribuída	Elevada	Deve ser possível fazer *drill down* e *roll up* segundo os níveis apresentados no RF_GE_005. É possível agregar e/ou filtrar quanto ao tipo de orçamento e origem da receita. Nos parâmetros temporais, permitir efetuar análise apenas e só cumulativa sobre um período de anos, semestres ou meses económicos selecionados.
IND_GO_018	Receita distribuída face ao orçamento modelo ajustada	Elevada	(Igual à do indicador IND_GO_017).
IND_GO_020	Dívidas de clientes	Elevada	Deve ser possível fazer *drill down* e *roll up* até ao nível 3. apresentado no RF_GE_005. É possível agregar e/ou filtrar quanto ao tipo de cliente e quanto à duração do atraso no recebimento. Nos parâmetros temporais, opção de selecionar um período anual, semestral, trimestral ou mensal já cumulativo até à data de extração.

Tabela 3.4: Breve descrição dos requisitos funcionais: indicadores (cont.).

Código	Designação	Prioridade	Descrição sucinta
IND_GO_023	Taxa de crescimento da receita arrecadada	Elevada	(Igual à indicador IND_GO_015). Calculada face ao período homólogo[4] do ano anterior.
IND_GO_024	Taxa de crescimento do peso da receita própria no financiamento total	Elevada	(Igual à indicador IND_GO_016). Calculada face ao *p.h.* do ano anterior.
IND_GO_021	Equilíbrio entre a despesa executada e a receita arrecadada	Elevada	Deve ser possível fazer *drill down* e *roll up* segundo os níveis apresentados no RF_GE_005. É possível apenas agregar simultaneamente quanto à origem da receita e tipo de despesa. Nos parâmetros temporais, opção de selecionar (ou não) análise cumulativa sobre um período de anos, semestres, trimestres ou meses económicos selecionados.
IND_GO_022	Equilíbrio entre a despesa executada e a receita distribuída	Elevada	Deve ser possível fazer *drill down* e *roll up* segundo os níveis apresentados no RF_GE_005. É possível agregar e/ou filtrar apenas por tipo de orçamento. Nos parâmetros temporais, opção de selecionar (ou não) análise cumulativa sobre um período de anos, semestres ou meses económicos selecionados.
IND_GO_019	Financiamento competitivo	Elevada	Deve ser possível fazer *drill down* e *roll up* segundo os níveis apresentados no RF_GE_005. É possível desagregar ou filtrar por tipo de financiamento. Nos parâmetros temporais, e para períodos não anuais, a opção de análise cumulativa é a única disponível.
IND_GO_025	Taxa de crescimento do financiamento competitivo	Elevada	(Igual à indicador IND_GO_024). Calculada face ao *p.h.* do ano anterior.

Tabela 3.4: Breve descrição dos requisitos funcionais: indicadores (cont.).

[4] *p.h.* (abrev. de período homólogo)

Código	Designação	Prioridade	Descrição sucinta
IND_GO_026	Prazo médio de recebimentos	–	Indicador removido segundo *feedback* do chefe da DPGD .
IND_GO_027	Prazo médio de pagamentos	–	Indicador removido segundo *feedback* do chefe da DPGD .
IND_GO_033	Receita arrecadada com a venda de produtos com a marca UC	Elevada	Não é possível fazer *drill down* e *roll up*, pois este indicador só tem expressão ao nível de granularidade da UC. Não tem filtros, nem agregadores. Nos parâmetros temporais, opção de selecionar (ou não) análise cumulativa sobre um período de anos, semestres ou meses económicos selecionados.
IND_GO_034	Taxa de crescimento da receita arrecadada com a venda de produtos com a marca UC	Elevada	(Igual à indicador IND_GO_033). Calculada face ao *p.h.* do ano anterior.

Tabela 3.4: Breve descrição dos requisitos funcionais: indicadores (fim).

Em suma, o fio condutor, em termos de funcionalidades, é que cada indicador possa ter parâmetros de agregação, filtros e parâmetros temporais aplicáveis sobre os dados apresentados no ecrã (gráficos e tabelas). Sempre que possível, todos os indicadores deverão apresentar dois gráficos: um de análise de evolução temporal e outro com a vista atual (*snapshot*).

A Figura 3.1 é um exemplo de um dos vários protótipos do módulo de Gestão Orçamental. Os protótipos foram desenvolvidos de acordo com a ideia principal: contemplar um nível de interatividade com os componentes o mais fidedigno possível. Com isto, pretendeu-se diminuir a subjetividade inerente à interpretação de cada uma das partes envolvidas no projeto e, por conseguinte, facilitar a validação dos requisitos.

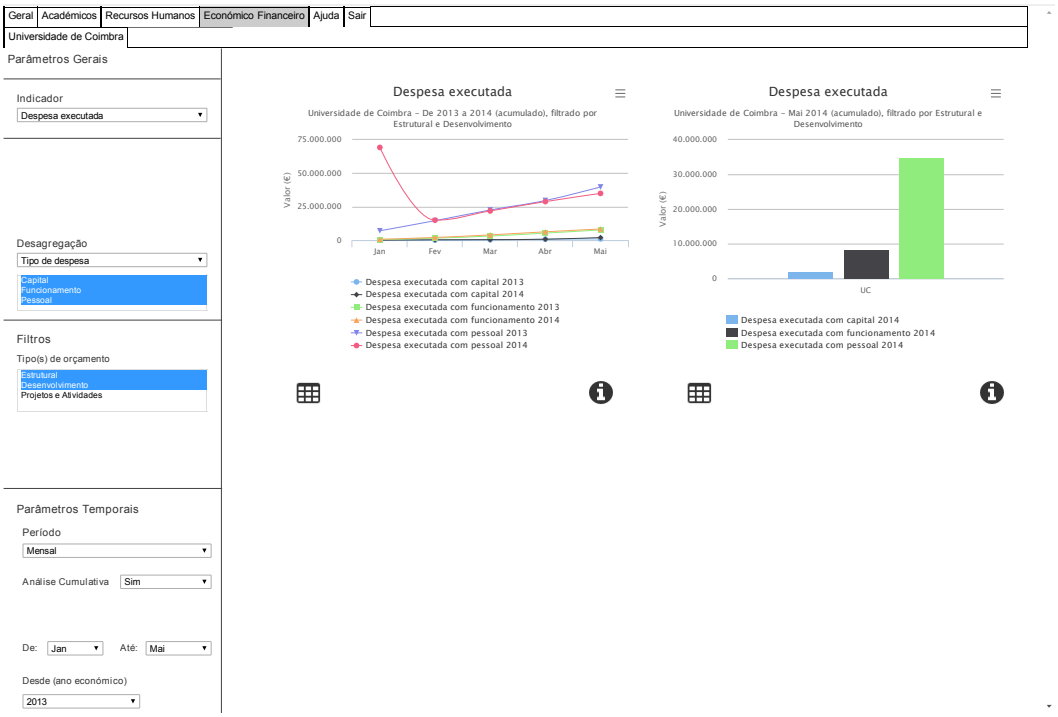


Figura 3.1: Protótipo de um *dashboard*: exemplo de um indicador com agregação e filtro simultâneos.

No Anexo digital (4) (ver página 107) encontra-se a compactação de todas as fichas de indicadores especificados e validados para o presente módulo.

3.2.2 Requisitos não funcionais

Os requisitos não funcionais estão, segundo o modelo FURPS+, categorizados de acordo com o que foi especificado anteriormente na secção 3.2, página 14. Convém relembrar que todos os módulos deste projeto integram um só sistema. Por esse motivo, os requisitos não funcionais são transversais a todos eles, tendo sido definidos conjuntamente com os responsáveis de todos os módulos constituintes, quer os três já desenvolvidos (sucesso escolar, receita de propinas e emolumentos e custo com o ensino), quer os três atuais (este módulo, o de recursos humanos e o de académicos).

Código	**Designação**	**Prioridade**	**Descrição sucinta**
RNF_S_001	Atualização de dados	Elevada	Processo ETL e atualização da DW e cubo OLAP devem ser automáticos.
RNF_S_002	Compatibilidade (*browser*	Elevada	Aplicação web deve ser compatível com os *browsers* mais modernos, a partir das versões mencionadas: Internet Explorer 9; Firefox 20, Safari 6 e Chrome 31.
RNF_S_003	Compatibilidade (SO)	Média	Como sistema operativo para os servidores deve ser suportada a distribuição de *Linux Red Hat - CentOS 6.x*.
RNF_S_004	Licenças	Elevada	A aplicação deve ser desenvolvida e disponibilizada através de software gratuito.
RNF_O_001	*Hardware*	Elevada	O software deve executar numa máquina com as seguintes características mínimas: 4Gb de RAM, 20Gb de espaço em disco e um processador dual core, não tem necessariamente de ser um ambiente de 64 bits. Estas características estão diretamente relacionadas com as mínimas exigidas pelo software que foi selecionado para desenvolvimento e disponibilização da aplicação.
RNF_O_002	Confidencialidade na comunicação	Elevada	Deve ser utilizado o protocolo HTTPS em toda a comunicação entre os utilizadores e a aplicação.
RNF_O_003	Autenticação	Elevada	Para aumentar a segurança da aplicação a autenticação e validação do acesso à aplicação deve ser efetuada com as credenciais dos utilizadores do sistema LDAP da UC.
RNF_R_001	Mecanismo de *fail over*	Elevada	Deve existir um mecanismo que garanta o funcionamento da aplicação, mesmo quando o servidor primário está indisponível.
RNF_R_002	Mecanismo de recuperação de falhas	Elevada	O sistema deve conter um mecanismo para iniciar automaticamente os serviços necessários ao seu correto funcionamento.
RNF_R_003	*Backup* dos dados	Elevada	Armazenar periodicamente backups dos dados presentes na base de dados.
RNF_P_001	Utilizadores em simultâneo	Elevada	A aplicação deve suportar pelo menos 60 utilizadores em simultâneo.

Tabela 3.5: Breve descrição dos requisitos não funcionais( ).

Código	Designação	Prioridade	Descrição sucinta
RNF_P_002	Desempenho do *fron-end*	Elevada	A medida de qualidade deve ser igual ou superior a 80 (considerando a pontuação disponibilizada pelo o *Yslow*[5].
RNF_U_001	Satisfação dos utilizadores	Elevada	A percentagem de satisfação dos utilizadores, com a usabilidade da aplicação, não deve ser inferior a 80%.
RNF_U_002	Satisfação dos utilizadores	Elevada	O tempo de carregamento das páginas *web* deve ser, no máximo, 5 segundos.

Tabela 3.5: Breve descrição dos requisitos não funcionais (fim).

3.3 Sumário

O levantamento e especificação de requisitos é, indubitavelmente, uma tarefa crucial e extremamente importante no desenvolvimento de qualquer projeto de software.
No sentido de se minimizarem os riscos inerentes a esta etapa, a técnica de prototipagem rápida revelou-se uma mais-valia junto das equipas operacionais e dos *stakeholders*, uma vez que ajudou a clarificar conceitos e funcionalidades pretendidas e a dissipar dúvidas que foram surgindo. Ao longo deste processo iterativo, foram sempre disponibilizadas as fichas de indicadores a todos os intervenientes do módulo de Gestão Orçamental (Anexo digital (4) - ver página 107).

Ainda no final do primeiro semestre, foram solicitados novos indicadores no âmbito do módulo do Plano Estratégico e de Ação [30], que iniciou especificação mais tarde, transversais a todos os módulos. A especificação dos requisitos de PEA afetos ao presente módulo foi, mais tarde, validada e os indicadores encontram-se já disponíveis para consulta.

De frisar a atualização de todos os requisitos (sobretudo os funcionais gerais e os não funcionais) num documento único, por parte da equipa UC-num e também tendo em consideração o *feedback* dado pelo júri dos elementos que se encontravam a realizar estágio curricular.

[5]Página oficial do *YSlow* disponível em: `http://yslow.org`

Não obstante a demora da validação oficial das fichas de indicadores, avançou-se para um esboço do desenho e especificação da arquitetura.

Capítulo 4

Arquitetura

Neste capítulo formaliza-se um dos pontos essenciais da Engenharia de Software: a arquitetura geral do sistema. Esta é de especial relevância, uma vez que define um conjunto de componentes necessários ao sistema, incluindo elementos de software e relações entre eles. De seguida, é feita uma análise e seleção das tecnologias a usar em cada dos componentes do sistema.
É também neste capítulo que se apresentam os modelos de dados da área temporária e do *data mart*.

4.1 Arquitetura Geral

A Figura 4.1 representa o modelo da arquitetura geral do sistema a desenvolver. Esta divide-se em três componentes principais:

1) ETL
2) Servidor de BI
3) Interface *web*

O primeiro componente, processo de ETL, é aquele que geralmente envolve maior complexidade e tempo. Em primeiro lugar, há que identificar bem os dados que vão "alimentar" a área temporária e saber de que forma se vai aceder aos mesmos. Uma vez identificadas, procede-se à extração dos dados das diversas fontes. Em seguida, esses dados vão ser sujeitos a um conjunto de transformações – etapa 2) da Figura 4.1 – que podem englobar formatação, limpeza, conversão entre diferentes tipos de dados, filtragem de valores, uniformização e cálculo de agregados necessários. Esta fase culmina com o

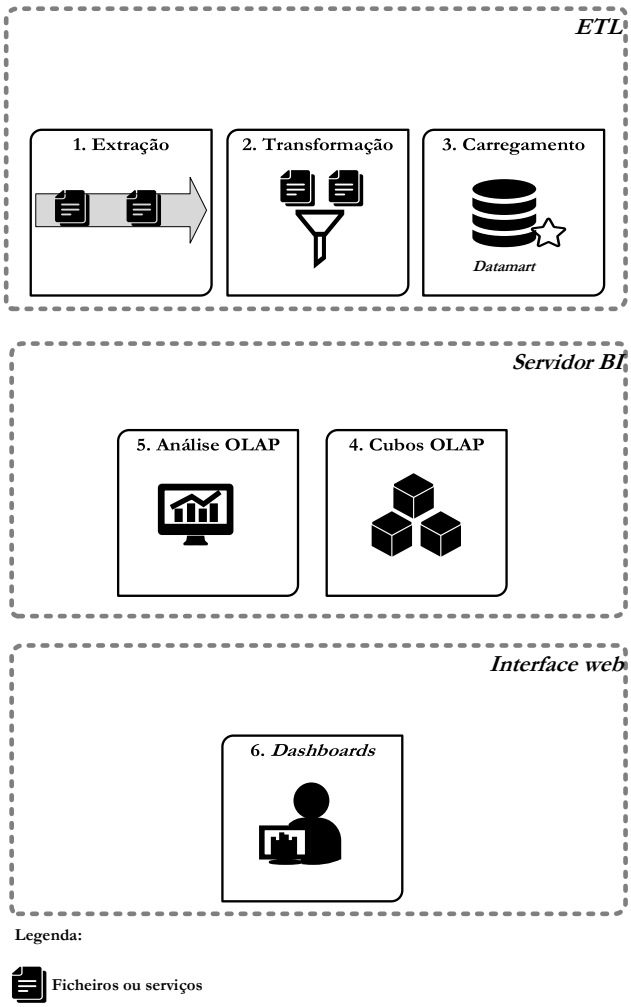


Figura 4.1: Arquitetura de alto nível.

carregamento do *data mart*[1].

Após o carregamento dos dados para o *data mart*, o servidor de BI encarrega-se de os carregar para aquilo que se define como cubo OLAP.
Um cubo OLAP é uma metáfora visual para o modelo multidimensional (ou "em estrela") da DW: cada célula corresponde a um facto (ou medida) da tabela de factos e as suas arestas representam as dimensões. No mínimo, cada

[1] *Data mart* representa uma "fatia" da Data Warehouse. Uma DW é, normalmente, constituída por vários *data marts* que representam áreas funcionais ou módulos diferentes, como é o caso.

estrela do modelo da DW será representada por um cubo. A criação de cubos facilita o acesso aos dados através dos conceitos de *drill down*, *roll up* e *slice and dice* (ver página 40), permitindo reduzir significativamente o tempo de consulta dos dados.

O último componente do sistema é o resultado do desenvolvimento de todo o *front-end* da aplicação, por parte do servidor de BI, disponibilizado aos utilizadores sob a forma uma aplicação, já criada e com alguns módulos desenvolvidos, com gráficos e tabelas interativos.

A Figura 4.2 apresenta a forma como os diferentes componentes do sistema comunicam entre si, no contexto geral, ou seja, representa o fluxo de dados. Para efetuar a extração dos dados das diferentes fontes são necessárias duas comunicações distintas: invocação de *webservices* para aceder à informação de SAP e acesso direto a ficheiros.

Numa segunda fase, o servidor de OLAP, embebido no servidor de BI, precisa de efetuar uma ligação SQL à DW, por forma a que o cubo OLAP conheça o respetivo esquema multidimensional do *data mart* presente na DW. Depois da definição e criação do(s) cubo(s) OLAP, o componente de análise OLAP efetuará consultas sobre o(s) mesmo(s), razão pela qual efetua uma comunicação direta via *Multidimensional Expressions* **(MDX)**[2].

Finalmente, para aceder à interface *web*, o utilizador terá de se autenticar com as suas credenciais da UC, autenticação essa que é solicitada pelo servidor de BI ao servidor externo de autenticação – servidor LDAP da UC.

[2]MDX: linguagem especialmente criada para executar *queries* em bases de dados multidimensionais, da mesma forma que o SQL executa *queries* sobre bases de dados relacionais.

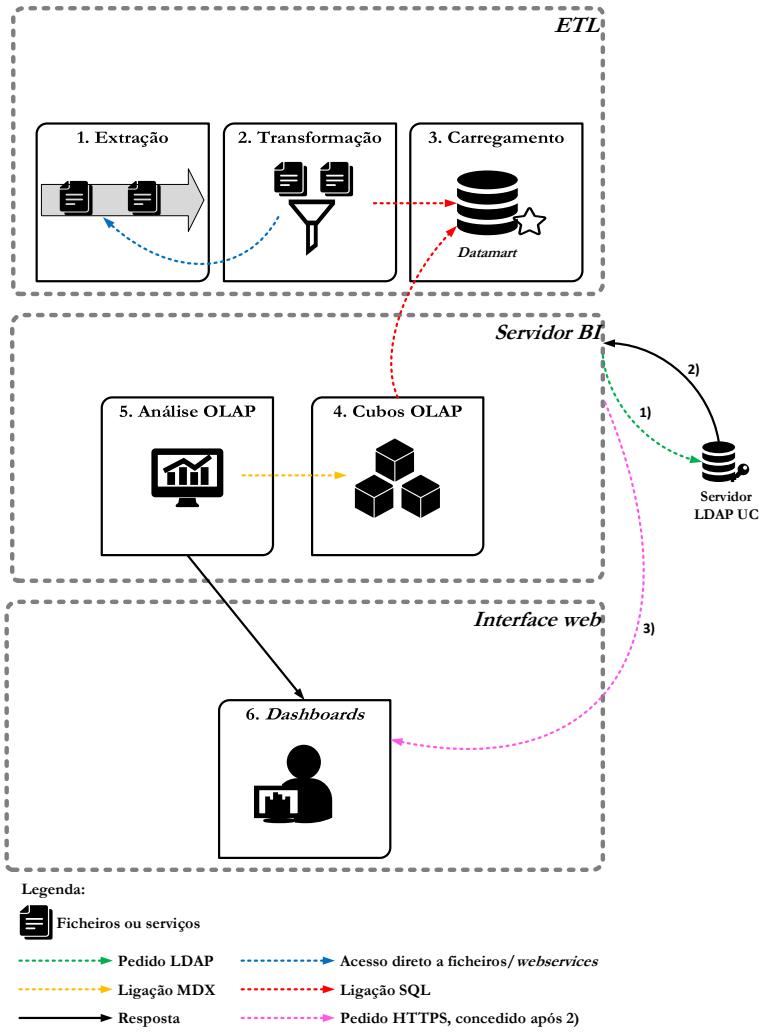


Figura 4.2: Arquitetura de alto nível – fluxo de dados.

4.2 Tecnologias

Conforme se pode observar pela Figura 4.1, existem três componentes distintas numa típica solução de BI para as quais será necessário selecionar ferramentas para o seu desenvolvimento. São elas: motores de bases de dados (quer para a área temporária, quer para a *Data Warehouse*), ferramentas para realização do processo de ETL e outra(s) para efetuar a análise OLAP e disponibilizá-la na *web*.

Esta pesquisa não pretende fazer um levantamento exaustivo do tipo de ferramentas a usar por dois motivos:

- O *budget* do projeto não prevê custos associados à aquisição de *software*, pelo que é imperativo recorrer a tecnologias gratuitas;
- As tecnologias já se encontram escolhidas, dado o desenvolvimento dos módulos anteriores deste projeto.

Não obstante as tecnologias já se encontrarem escolhidas, foi feito um levantamento e uma breve análise de algumas soluções de *Business Intelligence* mais usadas ou conhecidas no mercado, tendo em conta a primeira restrição. Essa análise encontra-se disponível no Anexo **Análise de Tecnologias** (página 93).

4.2.1 Motores de Bases de Dados

Numa solução de BI existem duas etapas onde ocorre o armazenamento de dados:

1) No processo de ETL, nomeadamente na fase de extração dos dados fonte para a área temporária, a base de dados assenta, como se verá mais à frente neste capítulo, num modelo de dados relacional;
2) No carregamento dos dados trabalhados pelo ETL para o *data mart* criado sobre uma BD cujo modelo de dados é multidimensional, que, na prática é implementado sob uma lógica relacional;

Dado que esse armazenamento é feito em bases de dados, é necessário selecionar um motor de BD.
A Figura 4.3 mostra o *ranking* de popularidade dos motores de bases de dados, calculado todos os meses pelo site *DB Engines Ranking* [29]. Dessa lista, destacam-se dois motores de BD relacionais gratuitos (posições 2 e 4):

240 systems in ranking, January 2015

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1439.16	-20.63
2.	2.	MySQL	Relational DBMS	1277.51	+8.93
3.	3.	Microsoft SQL Server	Relational DBMS	1198.61	-1.44
4.	4.	PostgreSQL	Relational DBMS	254.49	+0.48
5.	5.	MongoDB	Document store	250.90	+4.38
6.	6.	DB2	Relational DBMS	200.13	-10.12
7.	7.	Microsoft Access	Relational DBMS	139.14	-0.76
8.	↑ 9.	Cassandra	Wide column store	98.75	+4.69
9.	↓ 8.	SQLite	Relational DBMS	96.20	+1.49
10.	10.	Redis	Key-value store	94.24	+6.36

Figura 4.3: *Ranking* de popularidade dos motores de bases de dados.

- *MySQL* (Oracle) – "*The world's most popular open source database*"[2];
- *PostgreSQL* – "*The world's most advanced open source database*"[3].

No **Anexo B.1.1** (página 93) é apresentada uma análise com as principais características destes dois motores de bases de dados.

O motor *MySQL* tem um excelente desempenho de leitura e o seu *parser* de *queries* é muito eficiente para consultas simples, caraterísticas que, aliadas a uma fácil configuração e, por exemplo, a uma boa integração com *PHP*, o tornaram no motor mais popular em aplicações *web* de leitura intensiva (e.g., *Facebook*, *Twitter*, *Youtube* e *LinkedIn* são alguns dos clientes que a ele recorrem, embora não de forma exclusiva). Contudo, não suporta uma grande variedade de índices, o que limita, em certas situações, o seu desempenho em ambientes onde é necessário processar um grande volume de dados.
Por sua vez, o *PostgreSQL* tem-se vindo a desenvolver bastante nos últimos anos, sobretudo no que diz respeito ao aumento de performance (através de estruturas como índices ou vistas materializadas), integridade e fiabilidade dos dados, oferecendo suporte completo a transações *Atomicity Consistency Isolation Durability* **(ACID)**. Possui vários recursos que possibilitam uma excelente integração em aplicações de BI, tornando possível a criação de soluções gratuitas de alta performance e disponibilidade. É, por isso, ideal para *data warehouses* de tamanho pequeno a médio (0.5 a 5 TB), como é o caso do presente projeto.

Todavia, este "enraizamento" das bases de dados relacionais tem perdido força

com o surgimento e evolução daquilo que se designa por bases de dados não relacionais (***NoSQL***). Estas têm vindo a assumir um papel cada vez mais importante no "leque" de motores de bases de dados (ver motores de BD com as posições 5 e 8 na Figura 4.3), e, quando usadas de forma apropriada, podem realmente trazer benefícios ao nível do aumento de performance, escalabilidade, facilidade de replicação, baixos custos associados e elevada disponibilidade, entre outros [24]. Apesar de todo o entusiasmo que se tem gerado à volta destes motores de BD, há que ponderar a sua escolha, tendo consciência das suas reais limitações.
O facto de estarem ainda em estágios iniciais e de pré-produção, faz com que sejam alternativas pouco amadurecidas e com um suporte ao cliente normalmente prestado por *start-ups* que não têm o alcance global de grandes empresas como a *Oracle*, *Microsoft* ou *IBM* – grandes nomes associados ao SQL [5]. Um dos objetivos do *NoSQL* era ter uma administração a "custo zero", o que, na realidade, está muito aquém de acontecer: as soluções existentes atualmente ainda requerem conhecimentos especializados para instalação e um esforço considerável de manutenção.

Para além destas desvantagens, este tipo de motores apresenta ainda poucas capacidades para lidar com *queries ad-hoc* complexas – aspeto muito importante no âmbito de BI, mais concretamente, ao nível da análise OLAP. Para além disso, a maioria das ferramentas de BI existentes não disponibiliza ligação para bases de dados *NoSQL* o que, por si só, constitui um **fator eliminatório** da sua utilização em projetos.

Por este motivo, o **Anexo B.1.2** (página 95) não pretende efetuar uma análise comparativa entre motores deste tipo, mas apenas dar a conhecer as suas categorias, dando exemplos de soluções existentes para cada uma.

4.2.2 Extração, transformação e carregamento

O processo de ETL tem um papel muito importante no desenvolvimento de uma *data warehouse*, uma vez que é nele que ocorrem todos os processos necessários à preparação dos dados a carregar para a DW, de forma consistente e periódica.
As equipas de desenvolvimento têm duas opções de escolha para esse efeito: desenvolver código para criar uma ferramenta de ETL à medida e de raíz, ou escolher um software de ETL (gratuito, neste caso). Sem aprofundar muito esta questão, é fácil inferir que o uso de ferramentas já existentes no mer-

cado permite desenvolvimento simples, rápido, integração de código próprio e agendamento de tarefas, razões pelas quais se optou pela sua utilização neste projeto.
A escolha das ferramentas a utilizar deve seguir um conjunto de critérios de seleção, dos quais se destacam [21, 22]:

- Motores de BD suportados na extração e no carregamento;
- Suporte de diferentes tipos de dados e meta-dados;
- Possibilidade de carregar/exportar dados e meta-dados de/para diversas fontes de dados (diferentes motores de BD, ficheiros *Comma Separated Values* **(CSV)**, *Excel*, *Extensible Markup Language* **(XML)**, etc.);
- Linguagens de *scripting* suportadas;
- Mecanismo para agendar e executar tarefas de forma automática;
- Componentes disponíveis para dados e meta-dados (e.g., filtro, *lookup*, *join*, conversões entre formatos, cálculo de totais, cálculo de agregados, transformações definidas pelo utilizador, tratamento de registos duplicados, invocação de *webservices* (este é de especial interesse para o presente módulo, dado que é necessário invocar os *webservices* para obter os dados fonte disponibilizados pelo sistema SAP da UC, etc.);
- Processamento paralelo;
- Reutilização de código entre componentes;
- Usabilidade (também influencia a curva de aprendizagem dos elementos da equipa de desenvolvimento);
- Plataformas suportadas (e.g., Windows, Unix, Linux, OS X,...).

Tendo em conta os requisitos mencionados, o **Anexo B.2** (página 96) apresenta uma análise comparativa de duas ferramentas gratuitas e *open-source*:

- *Pentaho Data Integration* (abrev. PDI, também designado por *Kettle*);
- *JasperETL* (*Talend*).

Embora as duas ferramentas sejam muito semelhantes, o *Pentaho Data Integration* destaca-se primeiro, por já ter sido usado no desenvolvimento dos módulos do ano anterior (e com sucesso) e depois pela possibilidade de processamento paralelo e facilidade de agendamento automático de processos (*Jobs*), através da interface gráfica.

4.2.3 Análise de dados

A última, mas não menos importante, fase de um projeto de BI, é a disponibilização da(s) análise(s) feita(s) aos dados aos utilizadores, sob a forma de gráficos, tabelas, relatórios, etc. – análise OLAP. No caso do presente projeto,

esta disponibilização é feita através de uma aplicação *web*, construída anteriormente, intuitiva e fácil de usar.
Aquando da seleção destas ferramentas, devem ponderar-se alguns aspetos, tais como:

- Facilidade de utilização;
- Compatibilidade;
- Funcionalidades;
- Tecnologias suportadas;
- Geração de cubos OLAP;
- Criação de *Dashboards*;
- *Design*;
- Segurança;
- Suporte (e.g., documentação, comunidade, fóruns, ...).

Analogamente ao sucedido nas secções anteriores, foram escolhidas duas ferramentas gratuitas, cuja análise comparativa se encontra no **Anexo B.3** (página 99). São elas:

1) *Jasper Reports Server*
2) *Pentaho BI Server*

O ***Jasper Reports Server*** (versão *open-source* e gratuita da Jaspersoft), é um dos motores mais populares no que diz respeito à criação de relatórios, como o próprio nome sugere. Possibilita o desenho e desenvolvimento de relatórios OLAP, sendo capaz de incorporar dados de diversas fontes distintas. Esses relatórios podem depois ser exportados para diversos formatos, embora sejam mais direcionados para a impressão dessa informação, uma vez que são considerados *pixel-perfect*. Esta ferramenta destaca-se, negativamente, pelo facto de não suportar *dashboards* e um conjunto de operações a eles associados (e.g., *drill down* dos dados), a não ser na sua versão comercial [15].

O ***Pentaho BI Server*** (versão *community* da *Pentaho*) é uma das ferramentas gratuitas mais conhecidas e mais completas no mercado de *Business Intelligence*. Muitos outros projetos, inclusive o supra citado, limitam-se a endereçar uma ou outra função específicas (como a de criação de relatórios), mas não todo o espetro de BI. Além disso, em muitos falta a infraestrutura necessária no que diz respeito a segurança, administração, auditoria, capacidade de recuperação em caso de falhas, escalabilidade, entre outros [23, 26].
Esta ferramenta da *Pentaho* destaca-se por ter uma maior comunidade de desenvolvimento, mas sobretudo por permitir a criação de *dashboards* com ênfase na interatividade entre os componentes (gráficos e tabelas) e o utilizador final

(ver Figura B.1 do Anexo B.3, página 101).
Uma outra grande vantagem é o facto do *Pentaho BI Server* ter passado a integrar um servidor OLAP bastante conhecido em toda a comunidade — ***Mondrian*** — que permite construir e armazenar o cubo OLAP (modelo multidimensional definido para a DW). Este servidor OLAP é usado para:

- Análise interativa e de elevada performance sobre pequenos ou grandes volumes de informação;
- Exploração dimensional dos dados (e.g., analisar o valor da despesa executada, por tipo de despesa, por tipo de orçamento, num determinado período);
- Fazer o *parsing* de MDX para *Standard Query Language* **(SQL)** para devolver os resultados de *queries* dimensionais;
- Efetuar cálculos avançados utilizando as expressões de cálculo da linguagem MDX (e.g., *calculated members*. Os *calculated members* são úteis quando se pretende criar uma medida cujo valor não vem de uma coluna da tabela de factos, mas a partir de uma fórmula MDX. São definidos no esquema do *Mondrian*, como sendo parte da definição de um cubo. Um exemplo prático foi usar este conceito para calcular taxas de crescimento face ao período homólogo (abrev. ware*p.h.*).

4.3 Seleção de Tecnologias

Depois de dadas a conhecer as limitações do projeto na secção 4.2, bem como as funcionalidades e vantagens do software analisado, é possível reunir os critérios de seleção das ferramentas utilizadas para cada fim: motores de bases de dados (para a área temporária e DW), ETL e OLAP.
Uma vez que o desenvolvimento deste projeto foi iniciado na sequência de outros módulos já desenvolvidos e dado o parecer positivo da equipa anterior, consolidaram-se os critérios de seleção das tecnologias, optando por se manter as ferramentas, conforme consta na Tabela 4.1.

Embora não seja um critério de seleção, é uma mais-valia que todos os elementos da equipa tenham já tido contacto com a maioria das tecnologias aqui selecionadas. O *Community Dashboards Editor* **(CDE)** – *plugin* que integra a *framework* do CDF (ver anexo com a Figura B.1) – permite desenvolver páginas HTML com os respetivos *dashboards*, incluindo a sua personalização de estilos, através de CSS, e recurso a bibliotecas de *Javascript* para diversos fins (como é o caso da biblioteca externa *Highcharts* para criação de uma grande

variedade de gráficos em *Javascript* mais apelativos e "polidos" sob o ponto de vista estético).

Componente	Escolha	Motivos
Motor BD (área temporária + DW)	*PostgreSQL*	■ Destacam-se capacidades ao nível de índices, particionamento e vistas; ■ Processamento de *queries* em paralelo; ■ Escalabilidade; ■ Fiável e usada por clientes tão populares quanto: *Cisco*, *Skype*, *Imdb.com*, *RedHat*.
ETL	*Pentaho Data Integration*	■ Interface gráfica muito intuitiva; ■ Componentes que satisfazem as necessidades atuais relativamente a transformações, BDs suportadas, tipos de ficheiros de *input/output*; ■ Paralelização de transformações; ■ Agendamento de processos (*Jobs*).
OLAP	*Pentaho BI Server* + *Mondrian*	■ Criação e edição de *dashboards* de fácil manuseio graças ao *plugin* CDE; ■ Servidor OLAP — *Mondrian* — integra solução, permitindo desenho e construção do cubo OLAP, contribuindo para o aumento da performance de acesso aos dados. ■ Agendamento de processos (*Jobs*).

Tabela 4.1: Critérios de seleção das tecnologias.

4.4 Modelo de dados da área temporária

A área temporária, ou área de estágio, duma DW é uma área que se situa entre o sistema fonte (bases de dados, *webservices*, ficheiros,...) e o sistema alvo (tipicamente uma DW ou *data mart*) e é normalmente sobre ela que ocorrem as transformações, mas não é nela que ocorre todo o processo de extração, transformação e carregamento.

A criação do modelo de dados da área temporária justifica-se pela necessidade de agregação de dados dos diferentes sistemas fonte (como é o caso deste projeto), para que aqueles possam, posteriormente, ser limpos e transformados antes de serem carregados para o data *mart*. Este modelo, embora também seja implementado num motor de BD, pouca relação tem com o modelo de dados apresentado na Figura 4.5. Isto porque a ideia é que ele seja o mais desnormalizado possível, ou seja, que "imite" os dados fonte que carrega, armazenando-os em tabelas independentes, sempre que possível, sem chaves estrangeiras, sem referenciais de integridade, sem fazer *lookups* ou *checks* – a ideia é não sobrecarregar os sistemas fonte aquando dessa extração e, ao mesmo tempo, agilizar o processo de carregamento para a área de estágio. Só o processo de ETL deverá ter acesso aos dados presentes neste modelo, uma vez que se trata de um modelo de armazenamento temporário e para futuro processamento dos dados, não servindo, por isso, e de forma alguma, para efetuar consultas que irão alimentar os sistemas de análise.
Na Figura 4.4 é apresentado o modelo de dados de alto nível da área temporária. No Anexo C pode ser consultado este modelo em maior detalhe nas Figuras C.1, C.2 e C.3. A cor verde refere-se a entidades ligadas às unidades/depar-

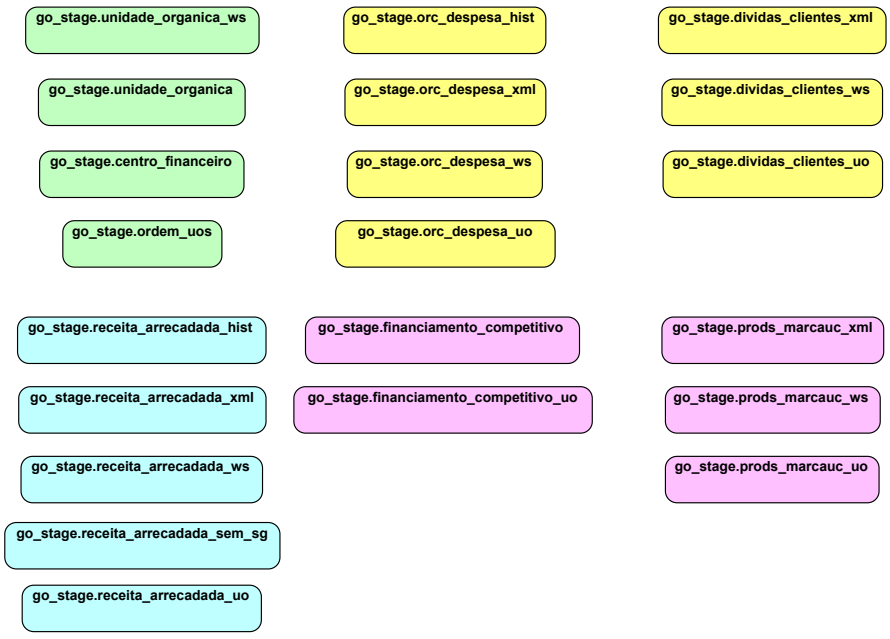


Figura 4.4: Modelo de dados de alto nível da área temporária.

tamentos que constituem a hierarquia, a cor amarelo diz respeito a entidade de âmbitos exclusivamente pertencentes à área de Gestão Orçamental, a cor

azul às que são partilhadas entre esse âmbito e o de PEA e, finalmente, a cor violeta para as entidades de indicadores exclusivos deste último.

4.5 Modelo de dados do *data mart*

A construção de um *data mart* implica, regra geral, construir dois modelos de dados: um modelo para área temporária e outro para o carregamento do próprio. O primeiro, referido na secção anterior, é um modelo inerente ao processo de ETL, enquanto que o segundo será responsável pelo armazenamento de todos os dados ao longo de vários anos. Este modelo tem que ser capaz, por um lado, de suportar uma grande quantidade de dados durante muitos anos, por outro, de devolver resultados de *queries* (quase sempre complexas e envolvendo agregações) de forma rápida – o tempo de resposta é, de facto, uma medida de eficiência de um sistema OLAP. Para ir de encontro a esses requisitos, adotou-se o modelo em estrela, proposto por *Ralph Kimball* [16]. Este modelo é composto por tabelas de factos e dimensões.

As tabelas de factos, também designadas por "estrelas", possuem factos (ou medidas), i.e., valores numéricos que podem ser aditivos ou semi-aditivos[3] e chaves estrangeiras para referenciar as dimensões. No contexto deste módulo, os factos capturados são todos aditivos. É comum que este tipo de factos represente as medidas das atividades do negócio ligadas aos seus indicadores de desempenho (KPI) e neste módulo não foi exceção: os factos constituem todas as métricas indispensáveis para o cálculo dos indicadores previamente definidos no Capítulo 3 (Tabela 3.4).
As dimensões, ao contrário das tabelas de factos, são tabelas compostas por vários atributos descritivos que servem para caraterizar os factos.
A relação entre cada tabela de factos e as dimensões é do tipo `[N]:1`, ou seja, a cada registo na tabela de factos corresponde apenas um registo da tabela de dimensões e, por sua vez, cada registo de uma dimensão pode ser referenciado por 1 ou mais registos nas tabelas de factos.
No caso deste módulo, o esquema em "estrela" do *data mart* é, na verdade, aquilo que se designa por constelação de factos – múltiplas tabelas de factos que partilham, entre si, dimensões.
Num *data mart* deve ainda ser possível realizar algumas ou todas as operações de *drill down*, *roll up*, *slice*, *dice* e *slice and dice* – conceitos que, por questões

[3]Factos semi-aditivos: quando os factos só podem ser somados em relação a algumas dimensões.

de contextualização, serão explicados na secção 4.5.
Tendo em conta todos os aspetos e conceitos supra mencionados, desenhou-se o modelo de dados para o *data mart* do módulo de Gestão Orçamental, cujo diagrama concetual se encontra na Figura 4.5.

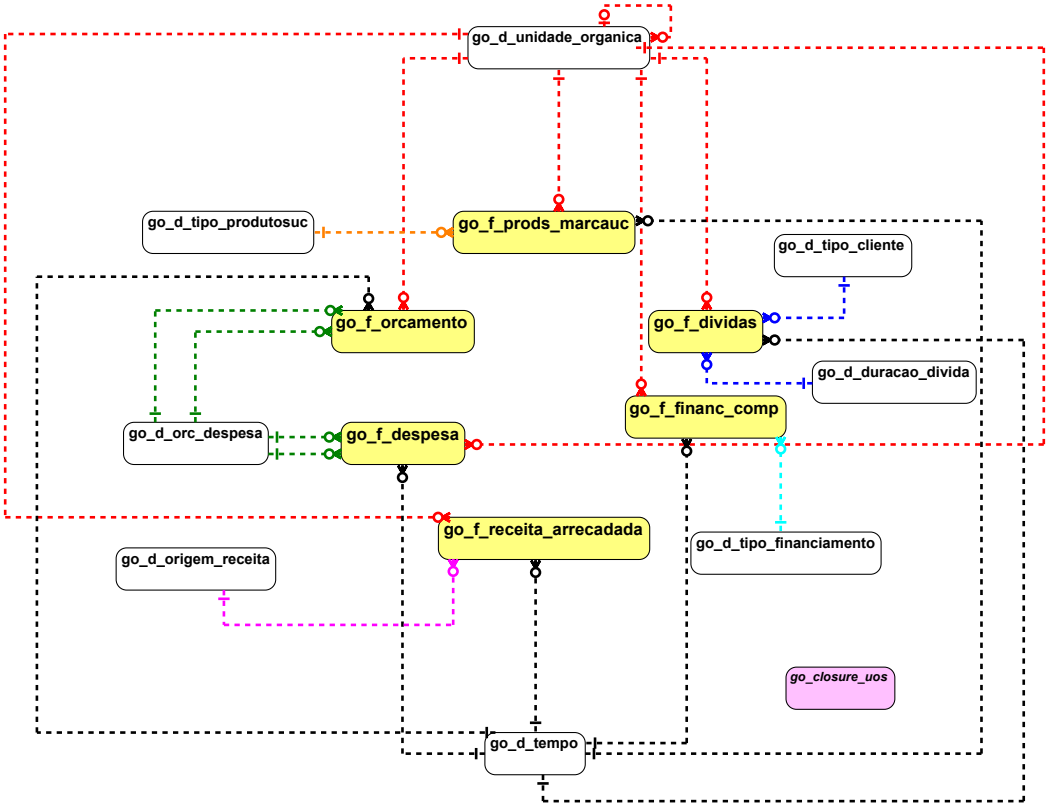


Figura 4.5: Modelo multidimensional de alto nível do *data mart*.

Na Figura C.4 (Anexo C, página 106) encontra-se este modelo em maior detalhe, sendo fácil constatar que as estrelas possuem um número reduzido de factos, `valor` e, eventualmente, `valor_acumulado`, por serem suficientes para o cálculo de cada indicador do presente módulo.

Granularidade

O conceito de granularidade é um dos mais importantes em *Data Warehousing*, independentemente da arquitetura e implementação utilizadas.
O grão é o menor nível de informação e é definido de acordo com as necessidades

do projeto. A granularidade refere-se ao nível de detalhe disponível, sendo inversamente proporcional a este — c.f. demonstrado na Figura 4.6. Ou seja:

- Baixa granularidade = maior detalhe
- Alta granularidade = menor detalhe

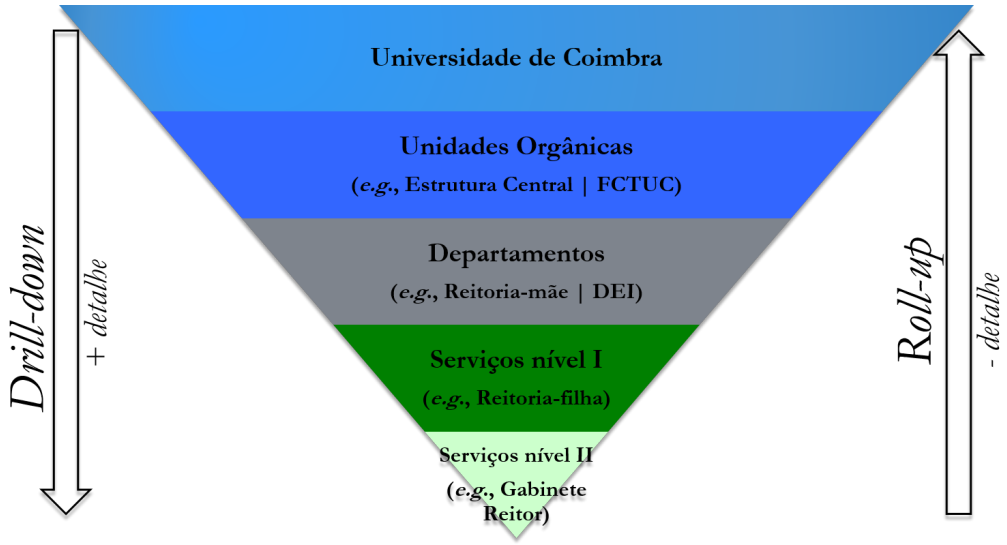


Figura 4.6: Granularidade *vs.* nível de detalhe [10].

Este conceito é também imprescindível para se entender as operações a efetuar sobre os dados. São elas:

- ***drill down*** - obter dados com maior detalhe, ou seja, partir de um nível de hierárquico superior para um nível inferior (e.g., partir da UC para a FCTUC), ou desagregar dados (equivale a adicionar uma ou mais dimensões ao cubo);
- ***roll up*** - operação inversa da anterior, obter dados subindo na granularidade de uma dimensão ou agregar dados (corresponde a remover uma ou mais dimensões do cubo);
- ***slice*** - obter dados selecionando apenas o valor de uma dimensão;
- ***dice*** - obter dados selecionando valores de uma ou mais dimensões;
- ***slice and dice*** - combinação entre as duas operações anteriores.

Ou seja, as hierarquias das dimensões estão diretamente relacionadas com a navegação pelos níveis de granularidade disponíveis (*drill down* e *roll up*), e os seus atributos com a capacidade de realizar *slice and dice* sobre os factos.

Granularidade das dimensões

As tabelas das dimensões devem conter caraterização até à granularidade mais fina possível, pelo que o seu crescimento é na horizontal – número considerável de atributos (colunas). Nas Tabela 4.2 são discriminadas todas as dimensões presentes no modelo apresentado na Figura 4.5 — dimensões cujo nome se inicia por `go_d_`.

Dimensão	**Descrição**
`go_d_duracao_divida`	Carateriza a duração do atraso no recebimento da dívidas de um cliente. Mapeia o número do intervalo inferior desse atraso na categoria que aparece nos gráficos (e.g., '0' corresponde à categoria "Até 90 dias").
`go_d_orc_despesa`	Esta dimensão, na realidade, equivale a duas: tipo de despesa e tipo de orçamento. Como estas duas dimensões eram muito semelhantes, e estavam associadas a factos com a mesma granularidade, optou por se criar uma dimensão única. Assim, os valores possíveis para tipo de despesa são: Pessoal, Funcionamento e Capital e para tipo de orçamento: Estrutural, Desenvolvimento e Projetos & Atividades.
`go_d_origem_receita`	Permite caraterizar a receita arrecadada quanto à sua origem (Orçamento de Estado, Propinas, Autofinanciamento e Outras Receitas) e ainda quanto ao tipo de propina (não conferentes de grau, Licenciatura, Mestrado, Doutoramento 3º Ciclo, entre outros), caso a origem seja "Propinas".
`go_d_tempo`	Esta é uma das dimensões mais importantes num *data mart*. Neste caso, está relacionada com a periodicidade das alterações efetuadas ao nível dos factos no sistema fonte, pelo que esta também determina a granularidade mais fina com que se consegue visualizar os dados – neste caso, será o mês económico (ou civil).
`go_d_tipo_cliente`	Permite distinguir os clientes devedores quanto à sua tipologia (e.g., Nacionais públicos, nacionais privados e Internacionais). Possui ainda outros dois atributos: a descrição do tipo de cliente segundo a fonte SAP e o nome do cliente que, embora não usados, podem vir a ser úteis em análises futuras.
`go_d_tipo_financiamento`	Permite distinguir entre financiamento competitivo, financiamento competitivo de investigação e receita arrecadada com Prestação de Serviços Especializados (PSE).

Tabela 4.2: Descrição geral das dimensões do modelo do *data mart* (cont.).

Dimensão	Descrição
go_d_tipo_produtosuc	Permite distinguir a receita arrecadada com produtos da marca UC pelo grupo de material vendido.
go_d_unidade_organica	Representa a hierarquia orgânica da UC ou do grupo UC (UC+SASUC), caso se aplique. Exemplo: Grupo UC→UC→Estrutura Central→Reitoria (mãe)→Reitoria (filha). Para representar esta hierarquia existe uma chave estrangeira para esta mesma dimensão nas respetivas tabelas de factos. Esta abordagem tem a grande vantagem de ser flexível, ou seja, consegue responder às necessidades do negócio, independentemente do grau hierárquico máximo que se estabeleceu aquando da definição de indicadores.

Tabela 4.2: Descrição geral das dimensões do modelo do *data mart*.

De uma maneira geral, constata-se que as dimensões construídas possuem um nível de detalhe muito baixo, o que não é habitual em *data warehousing*. Contudo, este é o nível possível e suficiente para responder a todas as questões de negócio, dado que em SAP já existia muita informação construída dessa maneira, por forma a ir respondendo às necessidades das equipas operacionais. Ao nível da receita, foi feito um esforço maior com a equipa operacional para que se conseguisse estabelecer uma origem de receita com maior detalhe, pelo menos para a receita arrecadada.

Granularidade das tabelas de factos

O primeiro e mais importante passo de *design* de um modelo de dados para uma *data warehouse* ou *data mart* é definir a granularidade das suas tabelas de factos. A granularidade de uma tabela de factos é a definição daquilo que representa um único registo da tabela de factos sob o ponto de vista do negócio.

É comum pensar-se que tal definição consiste em listar todas as chaves estrangeiras das dimensões como constituindo a chave primária da tabela de factos (i.e., que todas as chaves estrangeiras concatenadas dariam origem a uma chave primária composta (*composite key*). Esta é, aliás, uma das fábulas comuns em *Data Warehousing* e desmitificadas por *Ralph Kimball* [16]. Portanto, sob uma perspetiva lógica da modelação, as tabelas de factos não necessitam de ter chave primária. Esta é geralmente, e foi também no caso deste projeto, a

abordagem seguida e que funciona (por experiência própria).

Nas tabelas de factos sucede o oposto à das dimensões: para além das métricas, cada registo deve conter chaves das diversas dimensões segundo a sua granularidade, portanto, cresce na vertical – elevado número de registos (linhas).
A Tabela 4.3 apresenta a granularidade mínima de cada uma dessas tabelas de factos, com o intuito de compreender melhor o modelo de dados desenvolvido.

Tabela de factos	**Granularidade mínima**
`go_f_despesa`	Um registo {cabimentos & compromissos & despesa executada & despesa processada }, por tipo de orçamento, por tipo de despesa, por mês, com valor não acumulado.
`go_f_dividas`	Um registo, por tipo de cliente, por duração do atraso no recebimento, por mês, com valor acumulado à data de extração.
`go_f_financ_competitivo`	Um registo, por tipo de financiamento, por mês, com valores acumulado e não acumulado.
`go_f_orcamento`	Um registo {orçamento do ano & saldo de gerência & orçamento disponível}, por tipo de orçamento, por tipo de despesa, por mês, com valor não acumulado.
`go_f_prods_marcauc`	Um registo, por grupo de material, por mês, com valores acumulado e não acumulado.
`go_f_receita_arrecadada`	Um registo, por origem de receita, por mês, com valor não acumulado.

Tabela 4.3: Granularidade mínima das tabelas de factos.

4.6 Sumário

Neste capítulo foi definida a arquitetura geral do sistema, descrevendo também o fluxo de dados entre as suas componentes.
Foram ainda analisadas algumas tecnologias gratuitas passíveis de serem utilizadas em cada um dos componentes da arquitetura, embora a seleção das mesmas já tivesse sido efetuada pela equipa do ano anterior.

Apresentaram-se ainda os modelos de dados desenvolvidos quer para a área temporária onde se realizará grande parte do processo de ETL, quer para o *data mart*. Este último modelo engloba seis "estrelas" e as dimensões que caraterizam cada uma delas, para além de duas outras importantes dimensões,

partilhadas entre aquelas: a da hierarquia (diz respeito às unidades orgânicas do Grupo UC) e a do tempo.

Todos os modelos apresentados foram alvo de validação por parte da equipa, embora tenham sido alterados de acordo com as ligeiras necessidades que foram surgindo durante o desenvolvimento.

Capítulo 5

Implementação

Neste capítulo apresentam-se os detalhes mais importantes da fase de implementação, nomeadamente no que concerne ao processo de ETL, à criação e funcionamento dos cubos OLAP e desenvolvimento de *dashboards*. No final, é dado a conhecer o resultado da interface produzida.

5.1 Plano do processo de extração, transformação e carregamento

O processo de ETL, já referido no capítulo anterior, geralmente consome um mínimo de 70% do esforço dedicado a um projeto de *Data Warehousing* e é, quando mal desenhado, a principal causa do insucesso de projetos nesta área [7]. Por conseguinte, prestou-se especial atenção a esta fase, por forma a facilitar a resolução de problemas a jusante do desenvolvimento.

Este processo é constituído por três fases, sendo a primeira – **extração** – responsável pela recolha de dados dos sistemas fonte para o modelo da área temporária (consultar modelo nas Figuras do Anexo C), enquanto que a segunda – **transformação** – diz respeito à limpeza e formatação dos dados (remoção de duplicados, tratamento de valores nulos, eliminação de carateres, cálculo de valores, criação e modificação de campos, relacionamento de atributos, entre outros) para que os mesmos fiquem normalizados e aptos a popular o modelo multidimensional definido no capítulo anterior.
É nesta segunda fase que se apresentam mais dificuldades. É praticamente impossível que os dados provenientes dos sistemas fonte não contenham erros,

inconsistências, problemas de formatação e até incoerências. Por isso, ter um conhecimento aprofundado sobre as mesmas e dedicar algum tempo a conhecer os dados é uma mais-valia na altura de definir as transformações. Claro que, tratando-se de um processo automático, é difícil prever o que irá ser recolhido, por isso deve contemplar-se o maior número de cenários passível de ser tratado, de maneira a assegurar que nenhuns dados inconsistentes ou inválidos sejam guardados no *data mart* e, posteriormente, disponibilizados aos utilizadores da aplicação final.

A última fase do ETL consiste no **carregamento** dos dados para o *data mart* (cujo modelo se encontra na Figura C.4, Anexo C), sendo normalmente carregadas as dimensões em primeiro lugar, seguidas das tabelas de factos.

Nas secções seguintes o processo de ETL é abordado de forma mais pormenorizada, apresentando algumas das transformações mais significativas no desenvolvimento do mesmo.

5.1.1 Fluxo geral do processo de ETL

A criação de um *job* geral que efetue chamadas a outros *jobs*, nomeadamente o de carregamento da área temporária e o(s) de carregamento das dimensões e tabelas de factos do *data mart*, além de uma boa prática e recomendável, é condição para o cumprimento do requisito não funcional *RNF_S_001* (página 24). Em ambiente de produção, são depois executados um ou mais *jobs* gerais

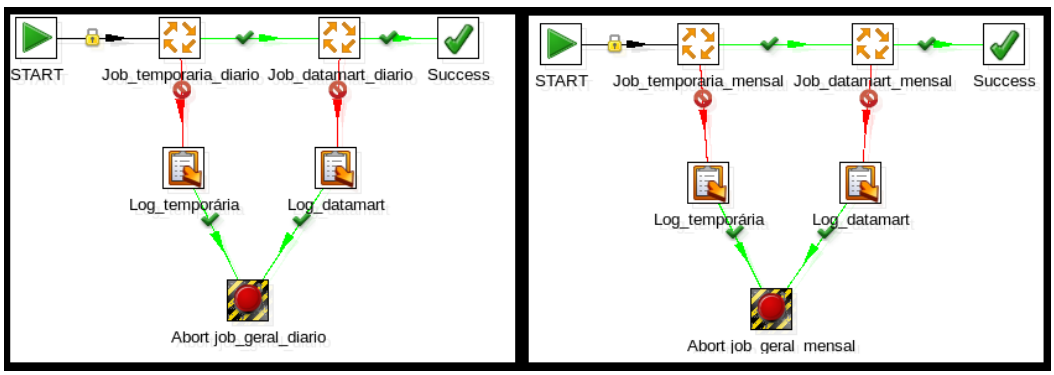


Figura 5.1: Processos ETL gerais: *jobs* diário e mensal (go_job_geral_diario.kjb e go_job_geral_mensal).

por cada módulo, consoante as diferentes periodicidades de recolha dos dados

(e.g., diária, semanal, mensal, ...).
A periodicidade de recolha de dados (via *webservices*) foi definida aquando da especificação de requisitos funcionais, nomeadamente nas fichas de cada indicador, por forma a responder convenientemente às questões de negócio. Por conseguinte, no presente módulo, houve necessidade de separar um *job* geral para carregamento diário e outro para âmbitos de carregamento mensal. Cada um dos *jobs* (***root jobs***) presentes na Figura 5.1 deve ser executado quer para o primeiro carregamento, quer para os carregamentos subsequentes.
Na definição de cada um destes *jobs* encontram-se chamadas a outros *jobs* que, por sua vez, são constituídos por várias transformações específicas de cada fase nele representada.

Por forma a minimizar o custo e impacto de alterações futuras, o processo foi faseado e organizado de forma semelhante entre âmbitos, não só no sentido de facilitar a sua compreensão e desenvolvimentos posteriores, mas também no de agilizar a criação e homogeneização da sintaxe dos *logs* e correção de erros.

5.1.2 Carregamento da área temporária

Conforme referido na secção anterior, o processo de recolha e carregamento dos dados dos sistemas fonte para a área temporária constitui a primeira fase do processo. Esta área não é mais do que um conjunto de tabelas avulsas (conforme se pode verificar no modelo presente nas Figuras C.1-C.3 no Anexo C) numa base de dados, sem quaisquer restrições de integridade, chaves primárias ou estrangeiras. Foi implementada deste modo com o intuito de agilizar o desenvolvimento do modelo de dados e de tornar mais simples e rápidos os processos de extração e carregamento dos dados.
Nas Figuras 5.2 e 5.3 encontram-se os *jobs* gerais para os dois tipos de periodicidade de recolha de dados. A recolha diária engloba apenas o carregamento da hierarquia das unidades orgânicas e o carregamento dos dados dos âmbitos de orçamento e despesa. Os restantes âmbitos são carregados mensalmente.

Independentemente da periodicidade de recolha, é em cada *job* constituinte dos *jobs* das Figuras 5.2 e 5.3 que são chamadas as transformações necessárias para carregar e trabalhar cada âmbito. De uma forma genérica e resumida, pode dizer-se que essas transformações envolveram os seguintes passos:

1. Carregamento de dados históricos[1] (caso existam), sejam eles a partir

[1]Para carregar dados históricos até à data atual, é passado o valor '0' como primeiro

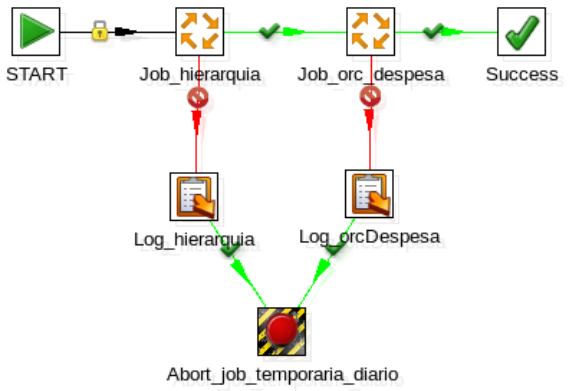


Figura 5.2: *Job* geral da área temporária para âmbito com carregamento diário (go_job_temporaria_diario.kjb)

daqueles ficheiros e/ou invocação de *webservices* (com dados do sistema SAP da UC) ou acessos a outras Bases de Dados externas (como será o caso dos SAS). Este carregamento é feito para uma tabela intermédia (que se designará por tabela ***a***), sempre que a granularidade e/ou desagregação dos dados não seja a mesma dos dados de carregamentos subsequentes (geralmente provenientes de *webservices*), caso contrário será feito de imediato o *join* com a tabela das unidades orgânicas, resultando logo na tabela final, tabela ***c***;

2. Carregamento de dados não históricos[2]. O carregamento subsequente é feito, *ipsis verbis*, para uma tabela exclusiva para o efeito (que será designada por tabela ***b***);

3. Tratamento, filtros, limpezas e cálculos das tabelas ***a*** e ***b*** e posterior junção de cada uma destas com a tabela da hierarquia, resultando na tabela ***c***. A ideia é que esta tabela final sirva de *input* com tudo o que é necessário às transformações do carregamento do *data mart*.

Nesta secção não se encontram detalhadas todas as transformações pertencentes aos *jobs* das Figuras 5.2 e 5.3, mas será feita uma descrição exemplificativa de algumas mais significativas ou complexas.

A Figura 5.4 diz respeito ao *job* geral da hierarquia, com todas as transforma-

argumento no *root job*.

[2]O carregamento de dados subsequentes é feito passando o 1º argumento com o valor '1'. Se o segundo argumento do *root job* for especificado, então carrega só esse ano, caso contrário carrega o ano atual (por defeito).

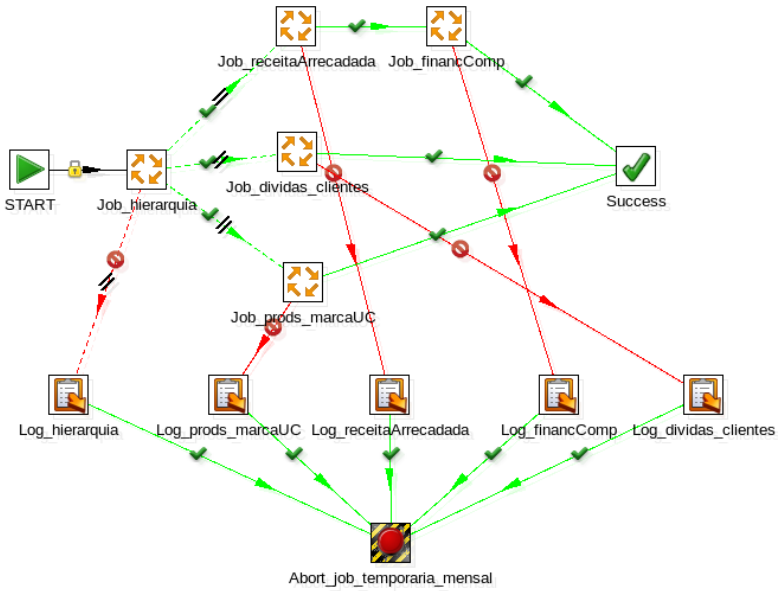


Figura 5.3: *Job* geral da área temporária para âmbitos com carregamento mensal (go_job_temporaria_mensal.kjb).

ções necessárias para construir a tabela das unidades orgânicas, quer a partir de ficheiros, quer a partir da invocação do *webservice*.

Na Figura 5.5 encontram-se dois fluxos distintos: o primeiro (ramo superior para argumento com valor '0'), vai buscar a variável `ano_inicial` definida pela transformação anterior (***input_from_file***) da Figura 5.4 e o ano atual; o segundo ramo define o ano a carregar como sendo o ano atual (defeito) ou o ano passado como segundo argumento do *root job*.
Os dois ramos prosseguem, embora nunca simultaneamente, para a mesma etapa, (*select_from_anoInit_to_anoAtual*) que vai, no fundo, funcionar como um ciclo para gerar o intervalo de anos para os quais vai ser feito o pedido *Simple Object Access Protocol* **(SOAP)** para o *webservice* (*ws*), já que este necessita de que lhe seja passado um ano como parâmetro. Este *webservice* em particular devolve 3 listas: a primeira com a árvore, ordenada, das unidades e serviços da UC, a segunda com os intervalos de centros financeiros de SAP a que corresponde cada folha da árvore anterior e a última lista que contém apenas uma mensagem com o *status* do *ws*. Se esse *status* indicar a presença de registos, é feito o *parsing* da resposta XML para extrair a informação das duas primeiras listas e colocá-las nas tabelas respetivas, pela sequência original (importante, para, mais tarde se atualizarem os pais das unidades orgânicas na tabela final, *unidade_organica*, e para se fazer a correspondência entre uma

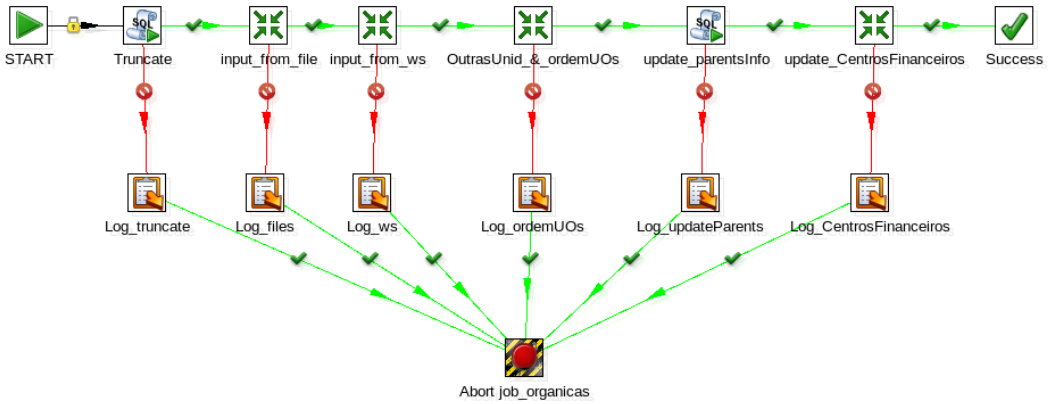


Figura 5.4: *Job* geral da área temporária para a hierarquia das unidades orgânicas (job_hieraquia.kjb).

unidade orgânica e o seu intervalo de centros financeiros (CFs)). De notar que o fluxo de dados é acompanhado de um controlo de erros para *log*. Outro dos processos que se destaca por ser diferente da maioria é o *job* da área temporária para o financiamento competitivo (Figura 5.6). Ao contrário do que acontece, por exemplo, com o âmbito do orçamento ou despesa, este âmbito não precisa de invocar um *webservice* nas chamadas subsequentes, pois é obtido a partir do da receita arrecadada (através da tabela *receita_arrecadada_ws*). Além disso, os seus valores não vêm pré-calculados da origem: é preciso filtrá-los para obter cada tipo de financiamento (único filtro/agregação que possui). Para o efeito, existem três transformações responsáveis por construir cada um dos três tipos de financiamento. Todas elas executam *queries* extensas que mapeiam as regras definidas pelo grupo operacional na ficha do indicador deste âmbito e seguem uma ordem, dado que os três tipos não são mutuamente exclusivos, como é o caso do financiamento competitivo e do competitivo de investigação, razão pela qual estas duas transformações têm de seguir a ordem indicada na Figura 5.6.

Tendo este âmbito sido especificado como sendo só para mostrar os valores acumulados, decidiu-se calcular os mesmos na área temporária, até porque o financiamento competitivo só possui um filtro (tipo de financiamento). O cálculo dos valores mensais acumulados foi feito através de uma *window function* do *PostgreSQL*[3]. Contudo, esta função só devolve resultados corretos se todos

[3]Definição e utilização de *window functions* em Postgres disponível em: `http://www.postgresql.org/docs/9.3/static/tutorial-window.html`

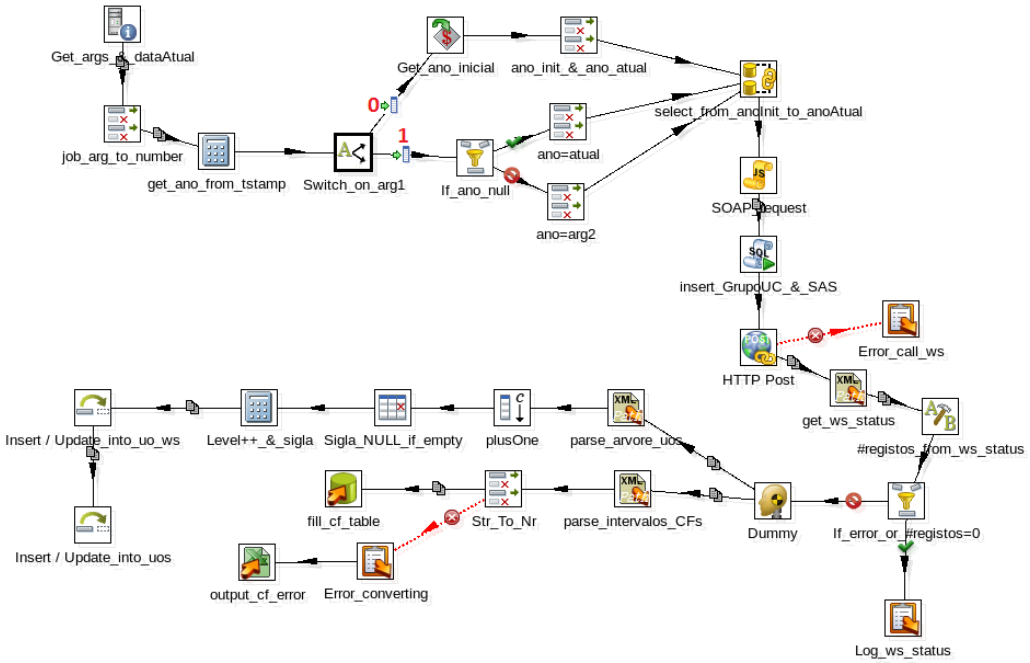


Figura 5.5: Transformação para carregamento da hierarquia das unidades orgânicas da UC a partir da invocação de um *webservice* (go_organicas_ws.ktr).

os meses existentes tivessem valores (ainda que fossem zero) para cada tipo de financiamento. Como tal, foi necessário criar uma transformação que tratasse desse pormenor. Assim, a transformação ***go_financComp_preencheMeses*** (presente na Figura 5.7) para cada ano de chamada do *webservice* (`ano_ws`) cria uma sequência com os 12 meses desse ano se o ano em que o *job* for executado (`ano_atual`) for superior àquele; se o ano de chamada do *ws* for o atual, então devolverá todos os meses até ao anterior inclusive. Esse número de meses é depois multiplicado por cada um dos tipos de financiamento existentes e por cada unidade orgânica, resultando no número de registos a inserir à granularidade mínima com valor mensal zero na tabela final deste âmbito (*financiamento_competitivo_uo*). Desta forma, a transformação ***competitivo_uo*** já poderá atualizar os valores mensais de cada registo e calcular corretamente os valores mensais acumulados à granularidade mínima pretendida.

Em suma, e conforme comprovado pelas Figuras 5.4 e 5.6, o carregamento da área temporária requer um elevado número de etapas, sobretudo devido à existência de duas fontes de dados distintas, cuja informação é agrupada no modelo de dados correspondente.

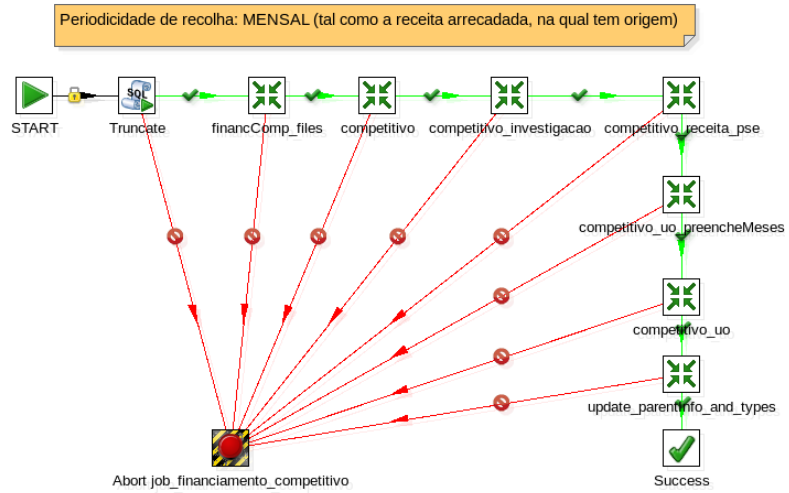


Figura 5.6: *Job* da área temporária para o âmbito do financiamento competitivo (job_financComp.kjb).

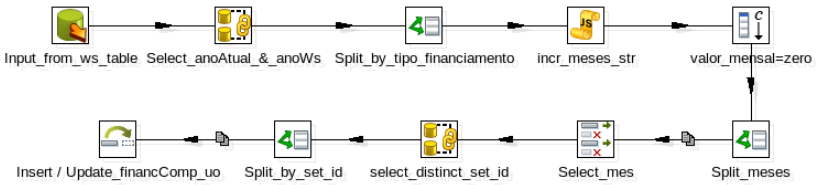


Figura 5.7: Transformação para auxílio do cálculo de valores por mês, tipo de financiamento e unidade orgânica para o financiamento competitivo (go_financComp_preencheMeses.ktr).

5.1.3 Carregamento do *data mart*

O carregamento de um *data mart* pressupõe carregar primeiro as dimensões e depois os factos, pela simples razão de preservar a integridade, já que as tabelas de factos possuem chaves estrangeiras para as dimensões que as caraterizam. Portanto, a prática mais comum é a de desenvolver um processo que execute todo o carregamento da área temporária, seguido do carregamento das dimensões e finalmente dos factos.

No presente módulo, essa lógica foi ajustada, uma vez mais, à periodicidade de recolha dos dados. Ou seja, o número de processos de carregamento do *data*

mart coincide com os de carregamento da área temporária, i.e., serão tantos quanto as diferentes periodicidades de recolha (imagine-se, por exemplo, um *job* de recolha diária, outro mensal no dia 15, outro mensal a executar no último dia do mês, ...), o que significa que haverá um *job* geral por cada periodicidade diferente(*cf.* já referido na secção 5.1.1).
Cada processo geral é agendado posterior e independentemente, razão pela qual convém garantir que o seu conteúdo executa na sequência correta. Deste modo, especificaram-se dois *jobs* para carregamento do *data mart*: um diário e outro mensal, para os âmbitos que já foram referidos nas secções anteriores.
Nas Figuras 5.8 e 5.9 encontram-se os dois *jobs* para carregamento do *data*

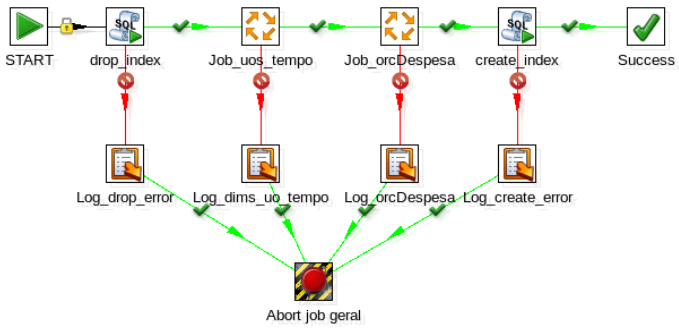


Figura 5.8: *Job* do *data mart* para o âmbito de carregamento diário. (job_datamart_diario.kjb).

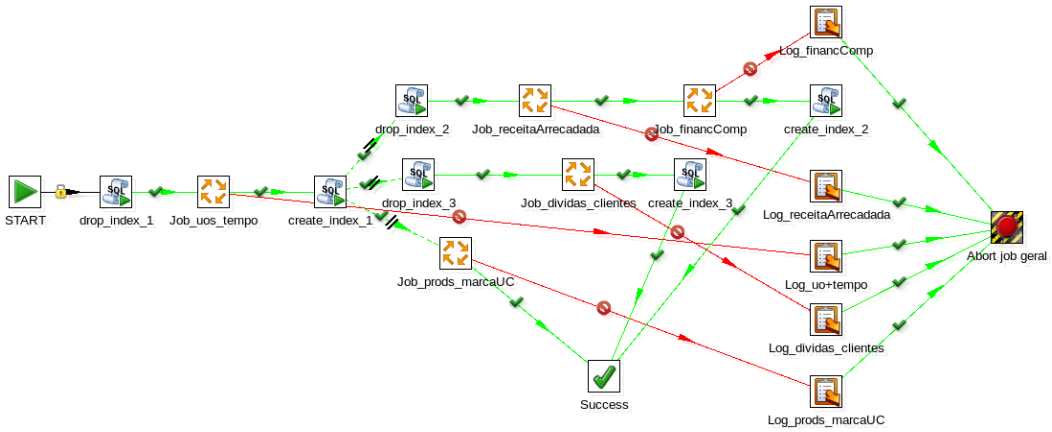


Figura 5.9: *Job* do *data mart* para os âmbitos de carregamento mensal. (job_datamart_mensal.kjb).

mart.

Nas secções seguintes são descritos alguns processos mais exemplificativos quer de carregamento de dimensões, quer de factos.

Carregamento de dimensões

O carregamento das dimensões corresponde à primeira etapa do carregamento de um âmbito no *data mart*.
O modelo multidimensional (ver Figura C.4, página 106) contém oito dimensões, mas apenas duas, *go_d_unidade_orgânica* e *go_d_tempo*, são partilhadas entre todas as tabelas de factos. Assim, o que é feito em cada *job* periódico é carregar primeiro essas duas dimensões (conforme se pode verificar nas Figuras 5.8 e 5.9), seguidas das dimensões específicas de cada âmbito.
A Figura 5.10 ilustra o processo de carregamento dos tipos de orçamento e

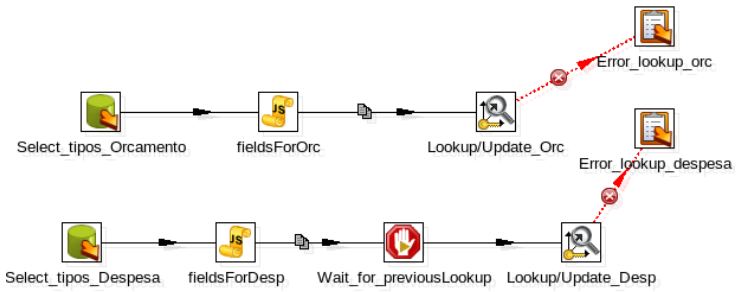


Figura 5.10: Carregamento da dimensão do tipo de orçamento e despesa (go_d_orcDespesa.ktr).

despesa que fazem parte da mesma dimensão (*go_d_orcDespesa*), em vez de existirem duas para o efeito. O processo é muito simples: selecionam-se os diferentes tipos de orçamento/despesa, atribui-se um tipo numérico ('0' ou '1', respetivamente) e inserem-se ou atualizam-se os registos na dimensão.

Já para a dimensão origem de receita, os dados a inserir não têm correspondência direta com aqueles que se encontram nas tabelas da área temporária. Como tal, foi necessário criar um ficheiro contendo os campo *origem de receita* e *tipo de propina*. Quando a origem de receita é relativa a propinas, o tipo de propina não é nulo e os dois campos são inseridos/atualizados na dimensão, caso contrário é inserida a origem de receita com o tipo de propina igual a 'ND' em vez de nulo (tomou-se esta opção apenas no sentido de facilitar as *queries* feitas em MDX, na fase de desenvolvimento dos *dashboards*).

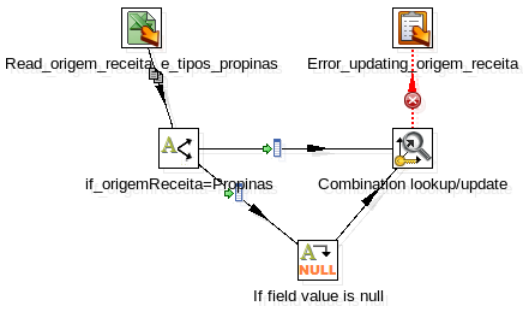


Figura 5.11: Carregamento da dimensão da origem de receita (go_d_origemReceita.ktr).

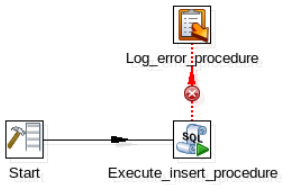


Figura 5.12: Carregamento da dimensão das unidades orgânicas (go_d_organicas.ktr).

Outro exemplo, diferente dos anteriores, mas muito importante é o do carregamento da unidade orgânica (Figura 5.12). Este processo é o único que precisou de recorrer a um procedimento SQL, pelo facto de as transformações do PDI não suportarem ciclos nas *streams*. Tal procedimento extrai primeiro todos os registos da tabela da área temporária com a informação sobre unidades orgânicas (sigla, descrição, nível do filho e do pai), depois para cada registo procura o identificador do pai e o identificador do filho com esse pai. Se esse filho não existir na dimensão, então insere e atualiza a ordem do mesmo (caso se trate duma unidade orgânica do nível das faculdades da UC, nível para o qual o grupo operacional especificou uma ordem de apresentação obrigatória).

Carregamento dos factos

Conforme referido anteriormente, o carregamento das tabelas de factos é a última fase do processo de carregamento (*loading*). Tal como o carregamento das dimensões, esta fase do processo não se revelou muito complexa, dado que a ideia de desenvolvimento foi a de aplicar o maior esforço e complexidade na área temporária.

Para o carregamento das tabelas de factos é geralmente feita uma seleção dos registos da(s) tabela(s) da área temporária que contêm toda a informação necessária para caraterizar cada facto. Em seguida, é necessário obter todos os identificadores únicos das respetivas dimensões (chaves estrangeiras). Uma vez encontrados, o registo é inserido com essas mesmas chaves e o(s) valor(es) do(s) facto(s). A Figura 5.13 é exemplificativa do tipo de processo mais comum no carregamento das tabelas de factos.
A transformação para o carregamento das tabelas de factos do orçamento e da

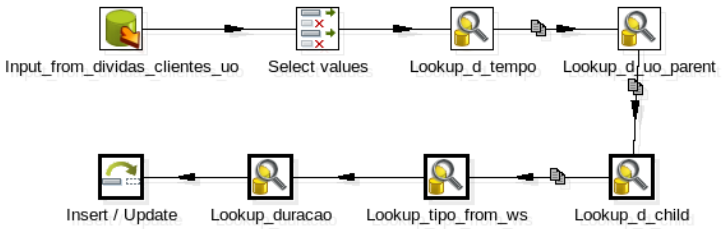


Figura 5.13: Carregamento da tabela de factos de dívidas de clientes (go_f_dividas_clientes.ktr).

despesa – presente na Figura 5.14 – começa por obter a soma mensal dos registos da tabela da área temporária por cada combinação de tipo de orçamento com tipo de despesa e por cada unidade orgânica (que não seja pai de outra) e, para cada registo pesquisa, pela ordem que se segue, os identificadores dos registos nas dimensões:

1. Tempo (*tempo_id*);
2. Tipo de orçamento (*tipo_orc_id*);
3. Tipo de despesa (*tipo_despesa_id*);
4. Unidade orgânica (*sup_id* e *uo_id*).

Após a obtenção dos identificadores, o registo do valor mensal é inserido ou atualizado na tabela de factos, consoante haja alterações nos dados a carregar (e, geralmente haverá, dado que alguns *webservices* irão devolver registos de meses passados que podem sofrer alterações já depois desse mês ter terminado).
A única *nuance* neste carregamento prende-se com a existência das duas chaves estrangeiras para a mesma dimensão – *go_d_orcDespesa* – uma para caraterizar o tipo de orçamento e outra para o tipo de despesa. Como cada registo

pode ter esses campos com valores nulos (devido ao que consta nos dados históricos deste âmbito, e não propriamente pelo conteúdo que tem origem no *webservice*), foi necessário desdobrar a procura dessas duas chaves nessa dimensão – uma para procurar o campo quando este não é nulo, outro para o caso contrário – devido ao facto de no passo de *Lookup* não ser permitido pesquisar campos por valores e valores não nulos simultaneamente.

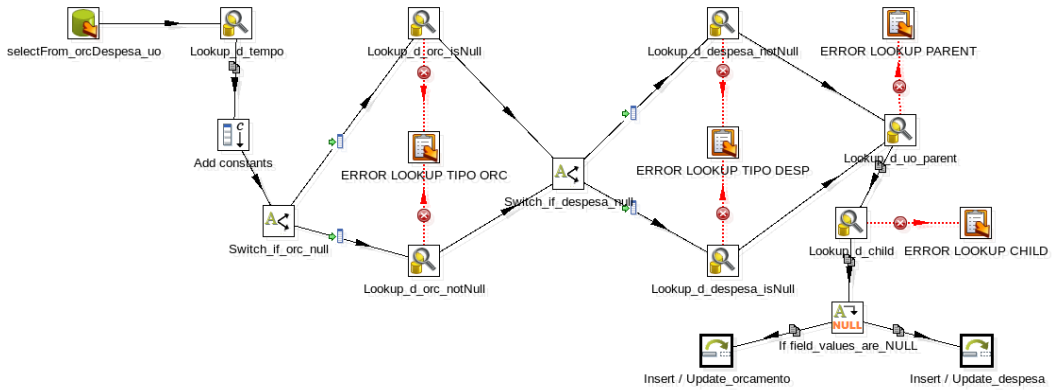


Figura 5.14: Carregamento da tabela de factos do orçamento e despesa (go_f_orcDespesa.ktr).

Criação de índices

Embora a performance não tenha sido um problema neste módulo, considera-se boa prática a criação de índices, pelo menos, para as colunas das tabelas de facto que contêm as chaves estrangeiras para as dimensões que as descrevem. De notar que o *drop dos índices* é feito antes do carregamento das dimensões e das tabelas de factos, sendo a criação dos mesmos só feita no final do processo. Para além daqueles, foram também criados alguns índices nas colunas das dimensões que aparecem mais frequentemente nas cláusulas *WHERE*. Para o efeito, foram executadas várias *queries* através da navegação dos *dashboards*, sendo possível consultar, nos *logs* do servidor, as pesquisas efetuadas efetivamente em MDX e também a conversão que o próprio faz de MDX para SQL internamente. Através do *log* de SQL foi então possível averiguar as *queries* que estavam a ser executadas, escolher os índices de acordo com os campos que eram usados e testá-los. Esse teste foi possível graças à funcionalidade

"*Explain plan*" da ferramenta *PGAdmin*[4], que demonstra o plano de execução de uma pesquisa executada pelo motor *Postgres*, incluindo os índices usados durante essa consulta. Só desta forma foi possível manter ou descartar os índices criados.

Com surpresa, os índices criados para a dimensão do tempo, nunca chegaram a ser usados pelo motor de base de dados. As pesquisas que envolvem esta dimensão executam sempre pesquisas com o mês/trimestre/semestre seguido do ano civil na cláusula *WHERE*. Por conseguinte, foram criados índices compostos (*e.g.*, (mês,ano)), depois de se ter chegado à conclusão que os simples não eram utilizados. Contudo, estes últimos também não foram a escolha do *Postgres*, motivo pelo qual foram descartados, para já.

5.2 Cubos OLAP

Conforme referido no capítulo da Arquitetura, o recurso a cubos OLAP para análise de dados de uma *data warehouse* ou de um *data mart* representa uma vantagem, sobretudo devido à sua performance neste tipo de análise, mas também pelo suporte que presta, a operações de *drill down/roll up*, *slice* e *dice* que embora possíveis de efetuar em bases de dados relacionais, seriam menos responsivas sem este tipo de estrutura.

O *Mondrian* é um servidor de OLAP que se encontra embutido no *Pentaho BI Server* e que permite efetuar pesquisas multidimensionais a grandes quantidades de dados do negócio, utilizando uma linguagem de *script* – MDX (já referida na página 28). Para o fazer, necessita de conhecer o(s) cubo(s) definidos para realizar as análises pretendidas. Cada cubo tem de ser definido num esquema em ficheiro XML, podendo a sua construção ser auxiliada por uma aplicação gráfica mais intuitiva e desenvolvida especificamente para o efeito, como é o caso do *Schema Workbench*[5] também da *Pentaho*.

O esquema baseia-se no modelo multidimensional do *data mart* e requer que se definam primeiro as dimensões e posteriormente os cubos (associados aos

[4] *PGAdmin*: ferramenta gráfica para administrar e desenvolver *PostgreSQL*. Mais informação disponível em: `http://www.pgadmin.org/`

[5] Documentação do *Mondrian Schema Workbench* disponível em: `http://mondrian.pentaho.com/documentation/workbench.php`

factos).
Para adicionar uma dimensão é preciso atribuir-lhe um nome e a tabela correspondente no modelo em estrela. Cada dimensão pode conter várias hierarquias – conjunto de membros organizados numa estrutura conveniente para análise – que por sua vez são constituídas por um ou mais níveis, que não são mais do que uma coleção de membros que têm a mesma distância a partir da raíz da hierarquia e que correspondem ao mapeamento de colunas. Na Figura 5.15

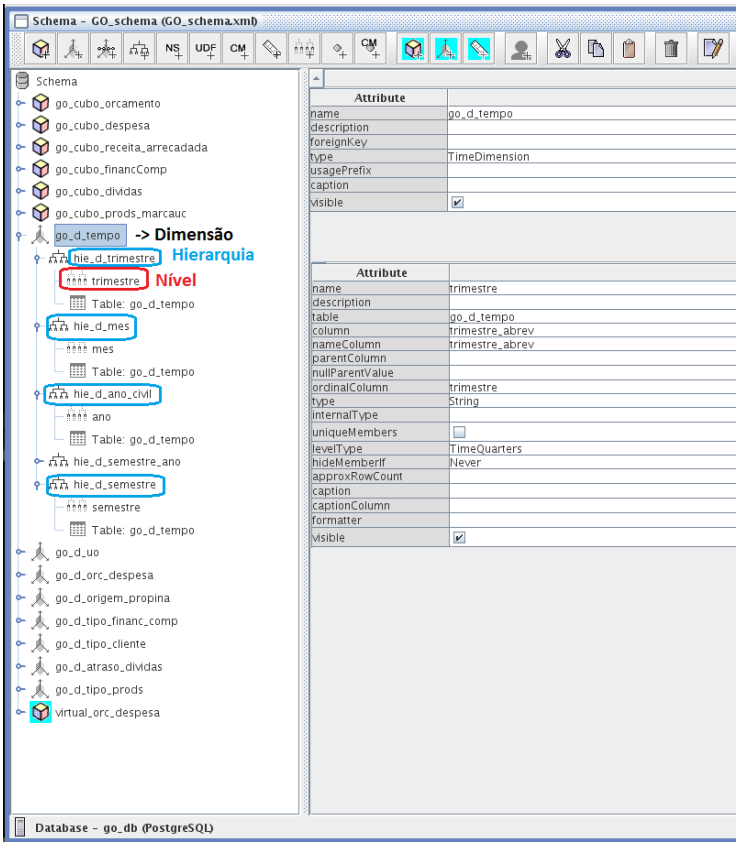


Figura 5.15: Exemplo de especificação de uma dimensão – dimensão tempo – detalhe de especificação do nível trimestre.

encontra-se a dimensão tempo com as diferentes hierarquias definidas. No caso do presente módulo, foi preciso especificar uma hierarquia única para o ano civil para depois cruzá-la com outras hierarquias também com um único nível (mês, trimestre ou semestre) da mesma dimensão , ao invés de especificar uma hierarquia para cada um daqueles períodos, juntamente com o nível do ano civil. Estas diferentes abordagens, embora possam parecer semelhantes e inócuas têm, na verdade, resultados práticos muito diferentes. Essa necessidade

surgiu, devido à diferença que existe ao nível dos *dashboards* deste módulo: o grupo operacional sugeriu e solicitou, aquando do desenvolvimento dos protótipos, que cada ano civil correspondesse a uma série no gráfico de evolução dos *dashboards* e o período figurasse no eixo dos *xx*, em vez vez de aparecer a designação contínua [mês/trimestre/semestre]/[ano] nesse mesmo eixo.

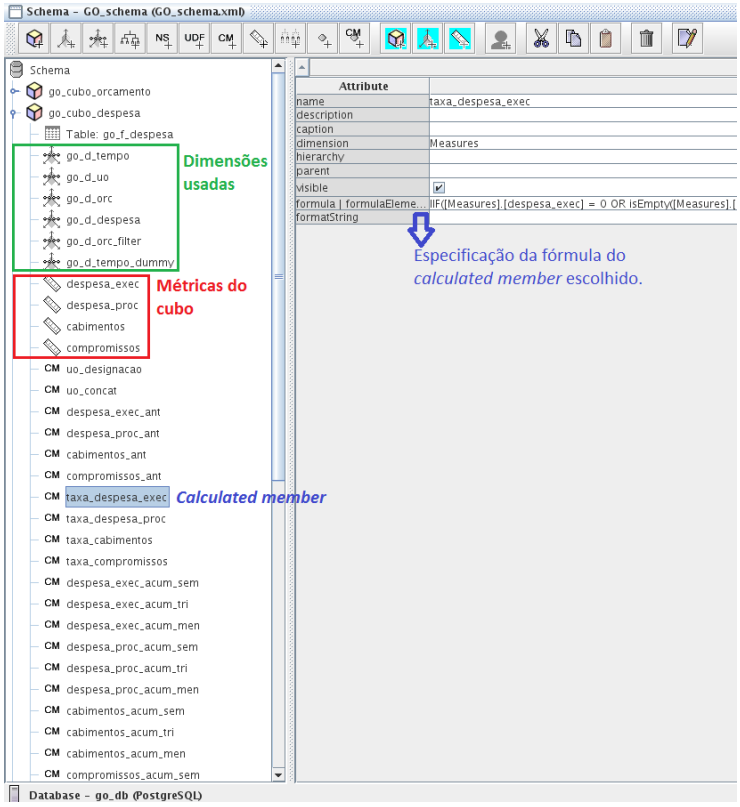


Figura 5.16: Exemplo de especificação de um cubo – cubo despesa – detalhe de especificação de um *calculated member*.

Para especificar um cubo procede-se de forma semelhante ao método usado nas dimensões: atribuir um nome e mapear a tabela correspondente do modelo. Em seguida definem-se as medidas ou factos que constam na tabela e a forma como estas podem ser agregadas – através de soma, média, contagem, contagem de distintos, mínimo ou máximo – podendo ainda ser especificadas novas métricas – que se designam por *calculated members* – a partir de outras já existentes , como foi o caso do cálculo da taxa de crescimento do período homólogo conforme se pode observar na Figura 5.16.

O *Schema Workbench* permite ainda definir cubos virtuais – estruturas que permitem cruzar métricas de duas tabelas de factos diferentes, mas com a mesma granularidade (*drill across*). Para este módulo, foi necessário criar um cubo virtual, uma vez que existem indicadores que envolveram cruzar métricas (*measures*) das tabelas de factos orçamento e despesa.
A definição de um cubo virtual é semelhante à de um cubo normal: bastou selecionar as dimensões pelas quais se iria fazer *slice* e selecionar os factos pelos quais se pretende fazer *drill across*, embora, neste caso em particular, fosse necessário especificar o cubo do qual cada um provinha. Posteriormente, criaram-se *calculated members* para criar os novos factos do cubo virtual, factos esses que envolvem divisões entre factos das duas tabelas supra citadas.

De referir ainda que esta etapa foi, de todas as etapas do desenvolvimento, a menos morosa e complexa, mas absolutamente necessária para passar à fase seguinte.

5.3 *Dashboards*

Nesta secção são abordados os aspetos mais relevantes do desenvolvimento dos *dashboards*. Desta implementação resulta o único componente do projeto em que há interação com o utilizador, sendo o local onde todos os indicadores desenvolvidos em cada módulo estão disponíveis para análise. Como tal, é a componente que requer maior atenção ao detalhe. Para o efeito, recorreu-se ao *plugin* do servidor de BI da *Pentaho* – o *Community Dashboards Editor* (CDE) – incluído na consola do *Pentaho* (Figura 5.17), ambiente de desenvolvimento do servidor de BI, que se encontra divido em três secções: ***layout***, **componentes** e ***datasources***.
O ***layout***, ao qual corresponde o ícone selecionado no destaque da Figura 5.17, diz respeito à camada de apresentação de cada *dashboard*, i.e., é aqui que é definido o código HTML, os *scripts* em *JavaScript* **(JS)** e as classes CSS que definem os estilos da página. Todos os estilos e ficheiros JS importados e utilizados pelo projeto no ano anterior foram mantidos, pelo que foi apenas necessário fazer alguns ajustes, consoante cada módulo e âmbito, no código HTML.

A segunda secção, a mais importante, cujo ícone se encontra selecionada na Figura 5.17, refere-se aos **componentes** dos *dashboards*, dos quais se destacam:

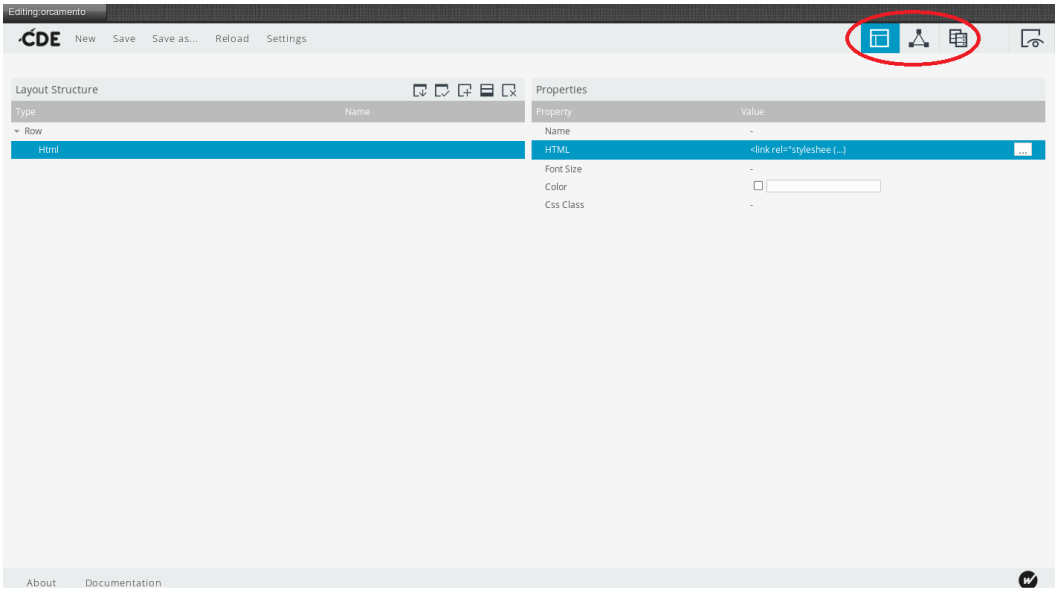


Figura 5.17: *Pentaho Console*: especificação do *layout* do *dashboard*.

- Caixas de seleção simples/múltipla;
- Gráficos;
- Tabelas;
- Parâmetros;
- Consultas afetas a uma componente (*Custom parameter*);
- Consultas independentes e com prioridade parametrizável (*Query component*).

Os primeiros três componentes listados são aqueles com os quais o utilizador interage e que têm de estar afetos, individualmente, a um objeto HTML, através da propriedade *HTMLObject*.
Os parâmetros têm especial importância porque é graças a eles que é possível criar toda a interatividade e sincronismo entre os diversos componentes. Literalmente todas as alterações que ocorrem no menu lateral, em qualquer caixa, afetam a informação que é apresentada nos gráficos. Essas alterações são guardadas num ou mais parâmetros (que podem ser *strings*, *arrays*, números, ...) associados a cada componente, sendo logo detetadas por todos os outros componentes que tiverem definido *listeners* para esses parâmetros (ver Figura 5.18).
Foi também nesta secção que se realizou a tarefa que constituiu o maior desafio a nível de desenvolvimento de *dashboards*: especificar as *queries* necessárias para apresentar as análises de *slice* and *dice* nos gráficos e/ou tabelas, bem

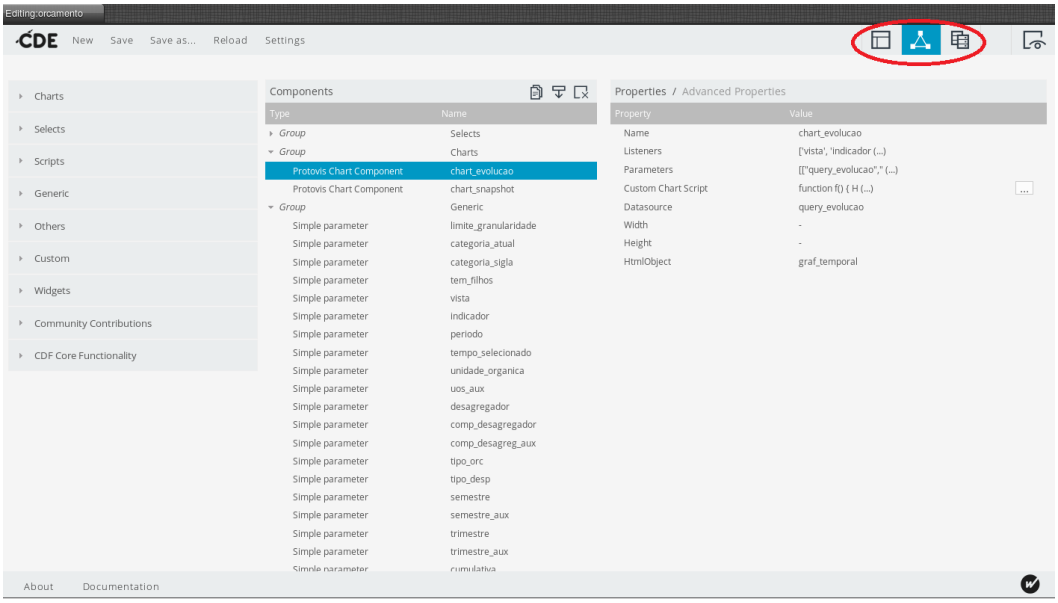


Figura 5.18: *Pentaho Console*: especificação de componentes do *dashboard*.

como as *queries* responsáveis por popular cada um dos componentes relativos a filtros/agregações e períodos temporais. Visto que as *queries* precisam de ser computadas de maneira diferente cada vez que o utilizador seleciona um parâmetro para fazer *slice*, depressa se percebeu a necessidade de tentar generalizar ao máximo funções JS que construíssem as *queries* MDX em tempo real dentro de um componente que, posteriormente, alimenta a fonte de dados adiante citada.

Também os componentes dos gráficos necessitam de dados para mostrar. Tal como já foi referido, esses dados podem ter origem numa fonte de dados com uma *query* que lhe está associada. Nestes componentes, como é utilizada a biblioteca externa do *highcharts*, o código *Javascript* para tratamento e construção do gráfico é feito na propriedade *Custom Chart Script* (Figura 5.18). Este código também foi generalizado no sentido de tentar responder às necessidades dos diferentes *dashboards*/âmbitos e, assim, diminuir o custo (tempo) de alterações futuras.
Finalmente, a terceira secção, ***datasources***, como o próprio nome indica, é onde se definem os acessos às bases de dados e esquema de cubos OLAP, através de consultas MDX ou SQL (daí que na propriedade *Query* deva figurar a consulta a efetuar).
No presente desenvolvimento apenas houve necessidade de recorrer a consultas

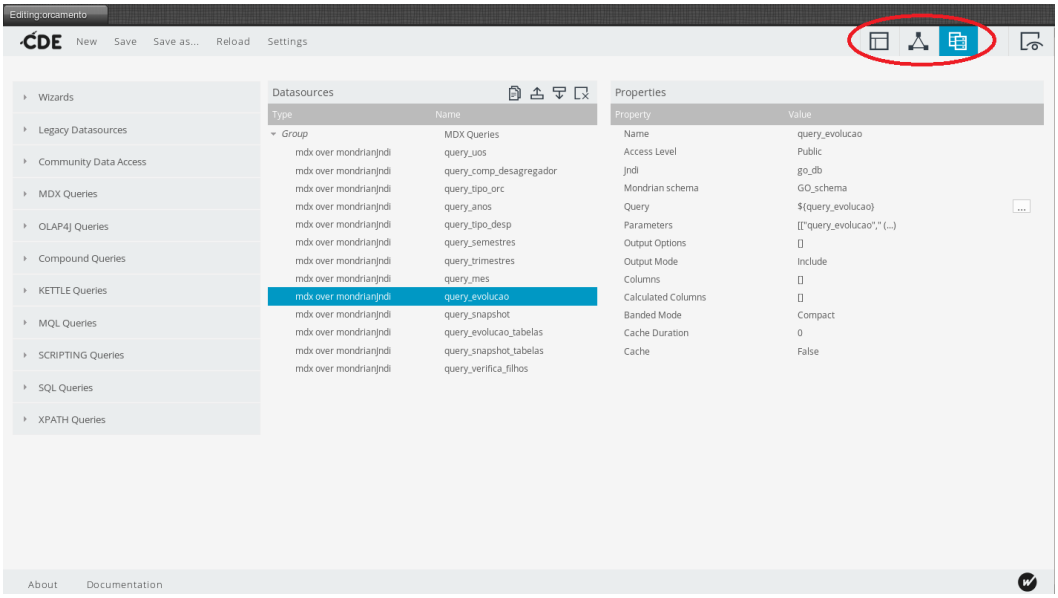


Figura 5.19: *Pentaho Console*: especificação de fontes de dados (*datasources*).

MDX, como se comprova pela Figura 5.19.

Em suma, as três secções aqui apresentadas estão interligadas e permitem modular e organizar a criação de um *dashboard* em três fases distintas. No final, a *framework CDF* (da qual fazer parte o *plugin CDE*) encarrega-se de compilar a informação presente em todos e criar o ficheiro com a extensão *.xcdf*, contendo a definição de todo o *dashboard* – conforme se pode verificar pela Figura B.1, na página 101.

5.4 Resultados

Nesta secção são apresentados alguns ecrãs, resultado do produto final ao qual o utilizador tem acesso atualmente. No total foram desenvolvidos seis *dashboards* correspondentes, também, a seis âmbitos – orçamento, despesa, receita arrecadada, dívidas de clientes, financiamento competitivo e receita da venda produtos com a marca UC – cada um deles contendo um ou até mais de uma dezena de indicadores.

O *dashboard* que mais se destaca é o da despesa, não só pela diversidade e

número de indicadores que possui, mas também pelo peso que teve no desenvolvimento em termos de esforço. A grande maioria dos indicadores da despesa figuram nos relatórios periódicos de Gestão Orçamental, razão pela qual é tomado como exemplo. O ecrã da Figura 5.20 mostra o indicador dos cabimentos para a Universidade de Coimbra.
Ao clicar na barra do gráfico da direita, é possível efetuar *drill down* e, assim,

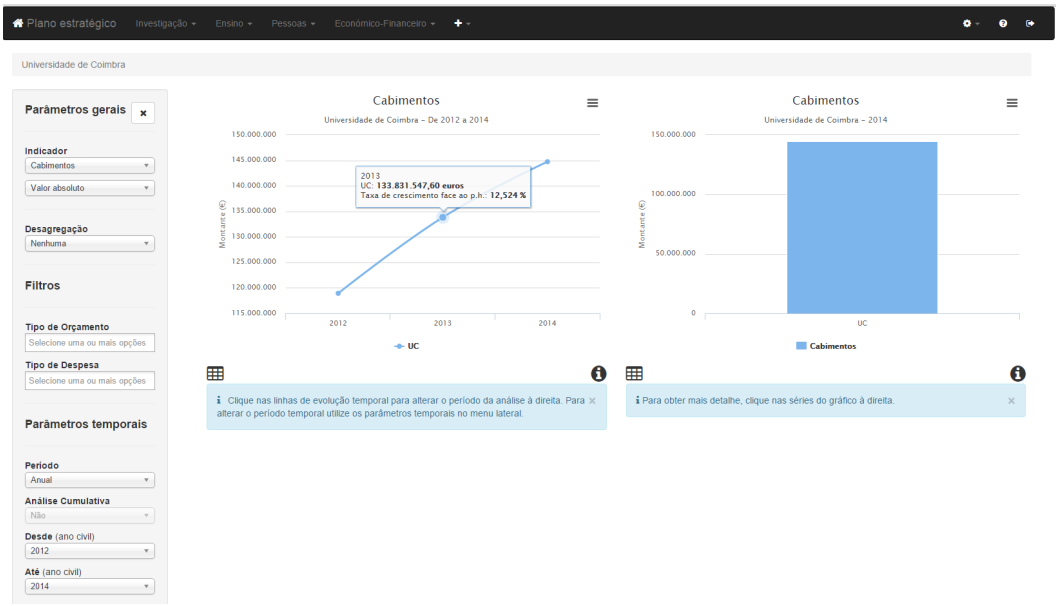


Figura 5.20: Ecrã do âmbito despesa, indicador cabimentos para a UC.

passar à análise do nível seguinte: o das unidades orgânicas da UC. Note-se ainda a seleção da agregação *Tipo de Orçamento*, que não tendo valores selecionados na caixa infra, corresponde à agregação dos dados por todos os valores existentes para aquela agregação, no período temporal selecionado, para o tipo de análise (cumulativa ou não) selecionada – Figura 5.21.

Um exemplo de uma granularidade mínima possível é o referido indicador com valor acumulado, para o mês de dezembro, para as unidades orgânicas da UC, por tipo de Orçamento Estrutural, por tipo de Despesa Funcionamento – Figura 5.22. Conforme se constata nesta figura, não existem valores da taxa de crescimento do período homólogo (*p.h.*), pois não existem valores do indicador para esse período, com aqueles filtros selecionados, nos anos anteriores, o que está correto.

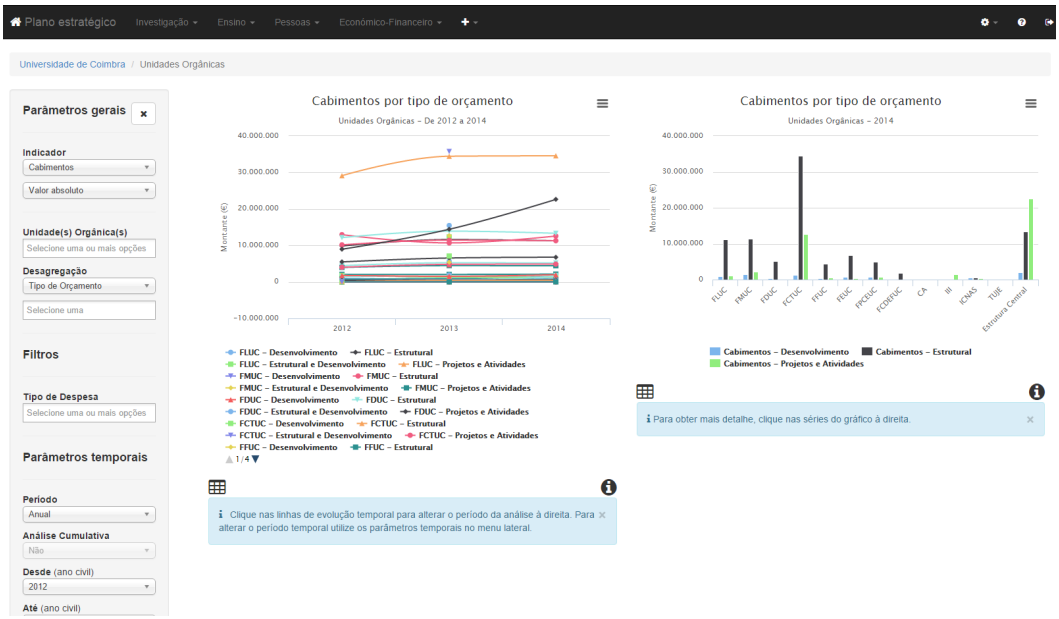


Figura 5.21: Ecrã do âmbito despesa, indicador cabimentos para as unidades orgânicas da UC, agregadas por tipo de orçamento.

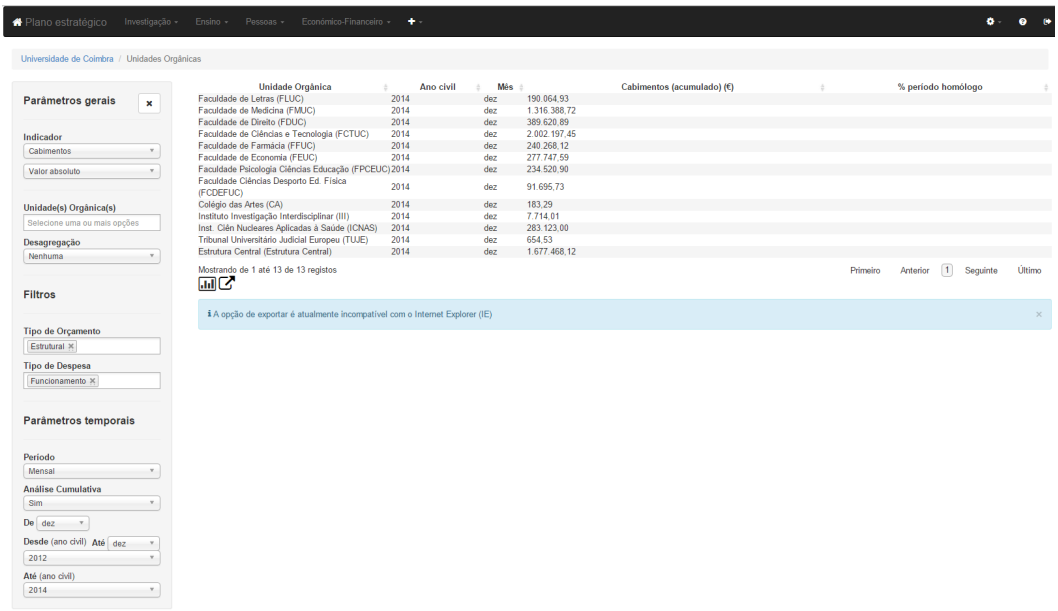


Figura 5.22: Ecrã do âmbito despesa, indicador cabimentos para as unidades orgânicas da UC, agregadas por tipo de orçamento e por tipo de despesa, para o mês de dezembro – tabela de evolução temporal.

5.5 Sumário

Neste capítulo foram destacados os detalhes mais relevantes de cada etapa da implementação.
Na primeira fase, o processo de ETL envolveu duas etapas distintas: carregamento da área temporária e carregamento do *data mart*. A maior parte do esforço e complexidade foi aplicada na primeira parte, por forma a facilitar a última. Foram criados dois processos (*jobs*) gerais: um diário e outro mensal, para âmbitos diferentes, englobando cada um deles em primeiro lugar o processo de carregamento da área temporária, seguido do *job* para carregamento do *data mart*, este carregando primeiro as dimensões e por fim as tabelas de facto do respetivo âmbito.

Posteriormente, foram definidos os cubos OLAP no *Mondrian* para o modelo multidimensional especificado.A grande mais valia na utilização de cubos reside na gestão de memória *cache* que estes efetuam, armazenando dados de pré-agregados e resultados de consultas anteriores, permitindo um aumento da velocidade e capacidade de resposta do modelo de dados.

A última e mais exigente fase do desenvolvimento diz respeito à criação dos *dashboards*, que passou sobretudo pela construção de *queries* MDX – linguagem cuja curva de aprendizagem foi acentuada – genéricas para popular dinamicamente alguns componentes de cada *dashboard*. Para além disso, foi uma fase que exigiu especial atenção ao detalhe, quer na apresentação e tratamento de casos específicos (como por exemplo, mostrar valores para um determinado filtro em função daquilo que esteja selecionada noutro filtro ou agregação, período temporal, tipo de análise cumulativa, etc.), quer no cumprimento da homogeneização de mensagens de erro/informação entre módulos (e.g., alertas indicando que não existem dados).

Convém aqui destacar a importância de seguir dois dos guias presentes nos anexos digitais (ver página 107): um contendo as guias para cada etapa do processo (Anexo (1)) e outro relativo à especificação de *design* (Anexo (2)). Estes documentos foram, naturalmente, sendo atualizados pela equipa durante o desenvolvimento do projeto.

Capítulo 6

Validação e testes

Neste capítulo é detalhado o processo de validação do módulo de gestão orçamental, descrevendo as várias fases, as partes envolvidas, os testes para validação das funcionalidades da aplicação final, bem como toda a documentação produzida para o efeito.

6.1 Validação do processo de ETL

A primeira verificação ocorreu ao nível dos processos de ETL para cada âmbito, tendo sido verificada a correspondência entre o número de registos obtidos após carregamento para a área temporária com o número de registos que se encontrava nos sistemas fonte. Tal foi possível quer para os dados históricos, quer para os dados obtidos a partir da invocação de *webservices*, uma vez que pelos primeiros é trivial saber pelo número de linhas e/ou colunas quantos registos esperar, e no segundo caso porque os *webservices* foram especificados por forma a devolver uma lista final com um item contendo uma mensagem de informação com o número de registos devolvidos.

Posteriormente, para assegurar que as bases de todo o trabalho se encontravam sólidas, no final do processo ETL (*job*) de cada âmbito eram feitas as somas dos valores em bruto nas tabelas da área temporária (tipicamente a do histórico e outra proveniente do *webservice*) e também na tabela de factos correspondente no *data mart*. Em alguns casos, a soma total estava correta, apesar de alguns subtotais que a constituíam apresentarem valores errados ou trocados – daí que as consultas de verificação tenham, por vezes, sido mais complexas do que

se previa.

No final, eram sempre executados os processos de ETL em separado, várias vezes, para garantir que não eram introduzidos duplicados nas tabelas de factos, para os mesmos dados. Caso esta etapa fosse bem sucedida em cada âmbito, era realizado o mesmo com os *jobs* gerais de ETL, por forma a certificar-se que a sequência e/ou o paralelismo dos processos/transformações constituintes não introduzia erros na computação dos dados finais. Em caso negativo, repetiam-se todos os passos até este ponto, até que os resultados dos dois modelos estivessem em concordância, e, consequentemente, prontos a ser utilizados pelos *dashboards*.

De relembrar que, neste processo, foram ainda definidos processos gerais (*jobs*) para atualização automática do *data mart*, sendo estes agendados de acordo com a periodicidade especificada, validando-se assim o requisito não funcional *RNF_S_001* (Tabela 3.5, página 25).
A criação de *logs*, com sintaxe transversal a todos os módulos, em cada processo ou transformação para que pudessem mais tarde ser tratados pelo sistema de gestão de *logs*, permitiu ainda validar o requisito não funcional – *RNF_S_005* (Tabela 3.5, página 25).

6.2 Validação do esquema dos cubos OLAP

A validação do esquema onde foram definidos os cubos OLAP foi feita de duas formas muito simples: aquando da definição dos mesmos, através de mensagens de erro/aviso do próprio *Schema Workbench* e após *deploy* para o *Mondrian* (servidor OLAP), com o auxílio da ferramenta *Saiku*[1]. Esta ferramenta permite listar todos os cubos e dimensões especificados no esquema anteriormente publicado e, por inspeção visual, é fácil concluir se existe algum erro no esquema, erro esse que pode ser investigado nos *logs* do próprio servidor. O *Saiku* permite ainda executar *queries* MDX nos cubos, quer através de um editor, quer através de *drag and drop*, o que facilitou a verificação de algumas *queries* construídas nos *dashboards*, sempre que se suspeitava que o erro não teria origem nas mesmas, mas sim no tratamento por parte do JavaScript.

[1] A ferramenta *Saiku* encontra-se, atualmente, embebida no servidor de BI, nomeadamente no ambiente de desenvolvimento (*Pentaho User console*).

6.3 Validação dos *dashboards*

A validação dos resultados dos *dashboards* foi, numa primeira fase, sendo feita à medida que se realizavam testes funcionais exaustivos (que figuram na secção seguinte). Foi também durante essa fase que, inevitavelmente, se realizaram os testes de compatibilidade da aplicação com os *browsers* definidos no requisito não funcional *RNF_S_002* – descrito na Tabela 3.5, página 25.
Durante a validação dos *dashboards*, foi também sendo verificado se o carregamento das páginas não era inferior a 5 segundos. Como o resultado foi positivo, o requisito não funcional *RNF_U_002* foi também cumprido.

Posteriormente, e no fim de todos os *bugs* corrigidos e dos testes terminados, os *dashboards* foram sendo colocados em produção à medida que ficavam prontos, ficando assim disponíveis para ser consultados e validados na plataforma.
A equipa responsável pela validação do presente módulo é constituída por elementos de divisões diferentes, dado terem sido desenvolvidos indicadores tanto de Gestão Orçamental, como do Plano Estratégico da UC. São eles:

- Chefe da *Divisão de Planeamento, Gestão e Desenvolvimento* (DPGD), Dr. Filipe Rocha;
- Chefe da *Divisão de Orçamento e Conta* (DOC), Dr. Nuno Patão;
- Equipa operacional da DOC: Cristina Reis e Paula Ferreira;
- Equipa operacional da DPGD: Ana Quental e Dora Lontro.

Para auxiliar e documentar este processo de validação "oficial" junto dos referidos *stakeholders*, foram elaborados os documentos que podem ser consultados nos Anexos digitais (4) e (5) (ver página 107), sendo que o primeiro documento contém todas as fichas de indicadores já previamente validadas pelos elementos supra citados.

As reuniões realizadas com as duas equipas operacionais foram possibilitando a recolha de *feedback* e sugestões também ao nível da usabilidade dos *dashboards*, pelo que, sempre que possível, essas foram tidas em conta, por forma a garantir a satisfação dos utilizadores-alvo. Por outro lado, também as fichas de validação enviadas permitem averiguar, de uma forma geral, a satisfação dos utilizadores relativamente a este ponto. Estes aspetos contribuíram, assim, para a validação do requisito não funcional *RNF_U_001*.

6.4 Testes funcionais

Deparou-se com a impossibilidade de, neste projeto, criar um processo de testes automatizado, devido às diferenças entre *dashboards* e indicadores de cada módulo.
Assim, a ideia base foi seguir uma abordagem *Black Box Testing*, ou seja, o utilizador testa o software como uma caixa preta: apenas as entradas, saídas e especificação são visíveis, e a funcionalidade é determinada observando as saídas para as entradas correspondentes.
Ao realizar os testes, utilizam-se vários *inputs* e os resultados são comparados com a especificação dos requisitos funcionais para validar a correção. Todos os casos de teste são derivados a partir da especificação.
De salientar que não são considerados os detalhes sobre a implementação do código, i.e., aquele que executa os testes não necessita de ter esse conhecimento.

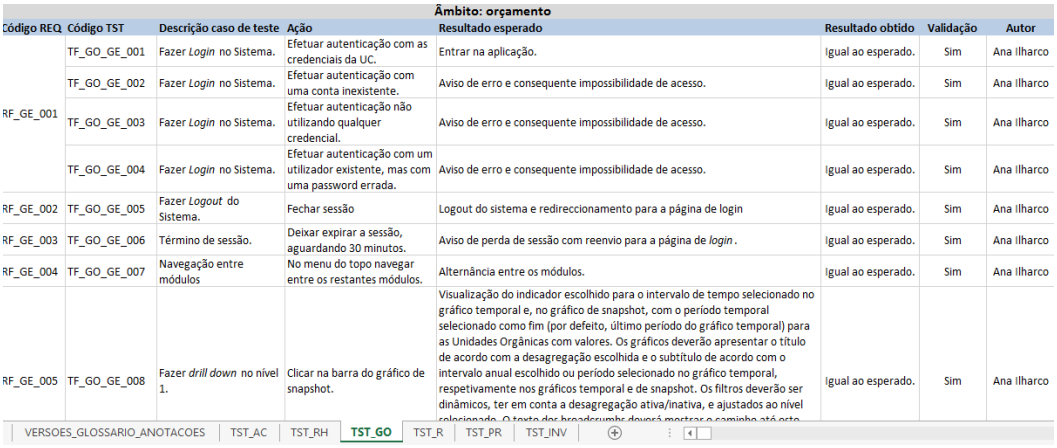
A metodologia dos testes funcionais foi escolhida, após pesquisa bibliográfica e reunião de equipa, no primeiro semestre, tendo sido aplicada, com sucesso, ao módulo de Propinas e Emolumentos (desenvolvido por um elemento do ano anterior). Por esse motivo, foi também aplicada na validação dos requisitos funcionais de cada novo módulo e engloba as seguintes etapas:

1) Criação de casos de teste;
2) Identificação dos testes;
3) Execução dos testes.

O ponto 1) implica reunir as fichas de indicadores de cada módulo e verificar se os requisitos funcionais já especificados são suficientes ou se têm a granularidade adequada (i.e., um requisito funcional pode ser demasiado genérico e não ser possível mapeá-lo num único caso de teste, podendo ter de ser subdividido em pequenos casos de teste) para realizar os testes.

No segundo ponto, a ideia base é derivar os casos de testes a partir dos requisitos funcionais especificados, salientando todas as operações de *drill down*, *roll up* e *slice and dice*.

Por último, o ponto 3) especifica que os casos de testes sejam registados de acordo com o exemplo do *template* que se encontra na Figura 6.1, anotando os resultados obtidos em cada linha, indicando, para cada uma, quem o executou e se é válido.



Âmbito: orçamento							
Código REQ	Código TST	Descrição caso de teste	Ação	Resultado esperado	Resultado obtido	Validação	Autor
RF_GE_001	TF_GO_GE_001	Fazer *Login* no Sistema.	Efetuar autenticação com as credenciais da UC.	Entrar na aplicação.	Igual ao esperado.	Sim	Ana Ilharco
	TF_GO_GE_002	Fazer *Login* no Sistema.	Efetuar autenticação com uma conta inexistente.	Aviso de erro e consequente impossibilidade de acesso.	Igual ao esperado.	Sim	Ana Ilharco
	TF_GO_GE_003	Fazer *Login* no Sistema.	Efetuar autenticação não utilizando qualquer credencial.	Aviso de erro e consequente impossibilidade de acesso.	Igual ao esperado.	Sim	Ana Ilharco
	TF_GO_GE_004	Fazer *Login* no Sistema.	Efetuar autenticação com um utilizador existente, mas com uma password errada.	Aviso de erro e consequente impossibilidade de acesso.	Igual ao esperado.	Sim	Ana Ilharco
RF_GE_002	TF_GO_GE_005	Fazer *Logout* do Sistema.	Fechar sessão	Logout do sistema e redireccionamento para a página de login	Igual ao esperado.	Sim	Ana Ilharco
RF_GE_003	TF_GO_GE_006	Término de sessão.	Deixar expirar a sessão, aguardando 30 minutos.	Aviso de perda de sessão com reenvio para a página de *login*.	Igual ao esperado.	Sim	Ana Ilharco
RF_GE_004	TF_GO_GE_007	Navegação entre módulos	No menu do topo navegar entre os restantes módulos.	Alternância entre os módulos.	Igual ao esperado.	Sim	Ana Ilharco
RF_GE_005	TF_GO_GE_008	Fazer *drill down* no nível 1.	Clicar na barra do gráfico de snapshot.	Visualização do indicador escolhido para o intervalo de tempo selecionado no gráfico temporal e, no gráfico de snapshot, com o período temporal selecionado como fim (por defeito, último período do gráfico temporal) para as Unidades Orgânicas com valores. Os gráficos deverão apresentar o título de acordo com a desagregação escolhida e o subtítulo de acordo com o intervalo anual escolhido ou período selecionado no gráfico temporal, respetivamente nos gráficos temporal e de snapshot. Os filtros deverão ser dinâmicos, ter em conta a desagregação ativa/inativa, e ajustados ao nível	Igual ao esperado.	Sim	Ana Ilharco

Figura 6.1: Exemplo de testes para validação de requisitos funcionais gerais no âmbito do orçamento do módulo de Gestão Orçamental.

Código	Designação	Prioridade	Validação
IND_GO_001	Orçamento do ano	Elevada	✓
IND_GO_002	Saldo de gerência	Elevada	✓
IND_GO_003	Orçamento disponível	Elevada	✓
IND_GO_004	Cabimentos	Elevada	✓
IND_GO_005	Cabimentos face ao orçamento disponível	Elevada	✓
IND_GO_006	Compromissos	Elevada	✓
IND_GO_007	Compromissos face aos cabimentos	Elevada	✓
IND_GO_008	Despesa processada	Elevada	✓
IND_GO_009	Despesa processada face aos compromissos	Elevada	✓
IND_GO_010	Despesa executada	Elevada	✓
IND_GO_011	Despesa executada face ao orçamento disponível	Elevada	✓
IND_GO_012	Despesa executada face aos cabimentos	Elevada	✓
IND_GO_013	Compromissos por pagar	Elevada	✓
IND_GO_014	Taxa de crescimento da despesa executada	Elevada	✓
IND_GO_016	Peso do autofinanciamento no financiamento total	Elevada	✓
IND_GO_017	Receita distribuída	Elevada	✗
IND_GO_018	Receita distribuída face ao orçamento modelo ajustado	Elevada	✗

Tabela 6.1: Validação de requisitos funcionais: indicadores (cont.).

Código	Designação	Prioridade	Validação
IND_GO_023	Taxa de crescimento da receita arrecadada	Elevada	✗
IND_GO_024	Taxa de crescimento do peso do autofinanciamento no financiamento total	Elevada	✓
IND_GO_021	Equilíbrio entre a despesa executada e a receita arrecadada	Elevada	✗
IND_GO_022	Equilíbrio entre a despesa executada e a receita distribuída	Elevada	✗
IND_GO_019	Financiamento competitivo	Elevada	✓
IND_GO_025	Taxa de crescimento do financiamento competitivo	Elevada	✓
IND_GO_033	Receita da venda de produtos com a marca UC	Elevada	✓
IND_GO_034	Taxa de crescimento da receita da venda de produtos com a marca UC	Elevada	✓

Tabela 6.1: Validação dos requisitos funcionais: indicadores.

A Tabela 6.1 apresenta a validação dos requisitos funcionais (todos eles de prioridade elevada) que dizem respeito aos indicadores especificados. A implementação destes indicadores foi indispensável para posteriormente se validarem os requisitos gerais, por âmbito, anteriormente apresentados.

No total de 25 indicadores, apenas não foram possíveis de implementar os 4 marcados como não validados na tabela anterior, por motivos que são alheios ao desenvolvimento.

6.5 Sumário

O presente capítulo descreve os cuidados tidos nas diferentes etapas de desenvolvimento por forma a validar a aplicação produzida – desde a validação do processo ETL, passando pela validação dos *dashboards*, dando especial ênfase à validação de requisitos funcionais, com testes realizados no final do desenvolvimento de cada âmbito. Cada âmbito deste módulo foi testado em ambiente de testes e, só depois de validado, é que foi colocado em ambiente de produção

para que ficasse disponível aos utilizadores finais e, assim, poder ser oficialmente validado pela(s) equipa(s) operacional(ais).

Até à data, ainda não foram devolvidas quaisquer fichas de validação preenchidas. Contudo, graças às várias reuniões com estes grupos operacionais, foi sendo possível reunir algum *feedback* e sugestões sobre os *dashboards*, bem como detetar erros nos dados fonte (*webservices*), cuja geração e pré-validação são da sua responsabilidade.

Apesar dos requisitos não funcionais serem da responsabilidade da equipa como um todo, houve alguns que tinham de ser cumpridos individualmente por cada módulo, durante o desenvolvimento. Tais requisitos foram também referidos neste capítulo, nomeadamente os que dizem respeito ao suporte e usabilidade.

Capítulo 7

Conclusões

Terminado o período de estágio há que fazer um balanço geral e crítico ao trabalho realizado, as contribuições do mesmo e de que forma o presente estágio contribuiu para a experiência pessoal da estagiária.
No final é ainda referido um possível crescimento da solução desenvolvida.

7.1 Balanço do estágio

Em primeiro lugar, destaca-se o cumprimento de todas as tarefas planeadas para ambas as fases deste trabalho. Consequentemente, os objetivos macro foram igualmente cumpridos, nomeadamente a criação de um *data mart* para a área de Económico-Financeira/Gestão Orçamental e a disponibilização de análises OLAP sob a forma de *dashboards* na plataforma, já existente, do projeto UC-num.

A área Económico-Financeira exige um grande esforço por parte das equipas operacionais (geralmente 4 a 6 meses) para recolher e tratar os dados manualmente, por forma a obter e construir os indicadores de performance (*KPIs*) que figuram não só em relatórios de Gestão Orçamental, como no Plano Estratégico e de Ação da UC e, não menos importante, em apresentações ao Conselho Geral. Perante tantas solicitações, tornou-se evidente a necessidade de encontrar uma solução mais prática e eficiente. Nesse sentido, foram realizadas várias reuniões com equipas operacionais diferentes, um elemento do SGSIIC e alguns *stakeholders*.

O facto de se tratar de uma área muito específica e de difícil assimilação por parte de desconhecidos na matéria (como é o caso da estagiária), poderá ter estado na origem da inexistência de alternativas mais viáveis até à data.
Por isso, o grande desafio consistiu em fazer com que esses elementos se abstraíssem daquilo que é a realidade operacional e entendessem o que era possível fazer com os dados ou seja, no fundo, entenderem qual o propósito da construção deste *data mart*.
Com efeito, e com a persistência e esforço aplicados durante o levantamento e especificação de requisitos, a estagiária ajudou a solucionar problemas, como, por exemplo, especificar indicadores que se julgavam impossíveis de obter ou que, sem o devido auxílio, possivelmente não existiriam hoje, ou pelo menos num futuro muito próximo, na plataforma.
Para além disso, contribui ativamente na equipa UC-num, tanto em questões de *design*, como em questões relacionadas com o desenvolvimento e validação, passando pela colaboração na elaboração de documentos do projeto, fatores esses que ajudaram na concretização do projeto, como um todo, para além de permitirem um bom funcionamento em equipa.

Os *dashboards* foram desenvolvidos para os diferentes âmbitos deste módulo e encontram-se disponíveis no *site* do projeto para um grupo restrito de utilizadores da UC.

A nível pessoal, o desenvolvimento do projeto possibilitou o apuramento das "habilidades sociais" (do inglês, *soft skills*), a evolução na capacidade de trabalho e gestão de tempo, graças a uma boa gestão de projeto e recurso a ferramentas adequadas, para além de uma motivação acrescida pelo facto de se tratar não de uma prova de conceito para a UC, mas de um produto inédito que pretende efetivamente ser usado por esta instituição.

A nível académico, técnico e profissional destacam-se a integração numa equipa de desenvolvimento, jovem, dinâmica e com grande vontade de aprender, a integração em equipas multidisciplinares – que se aproximou muito das típicas reuniões com clientes, em ambiente empresarial – o recurso a ferramentas de gestão e métodos de desenvolvimento que também se assemelham aos de um projeto de software na vida real e, finalmente, mas não menos importante a aquisição de competências no âmbito da engenharia de software, nomeadamente na área de *Business Intelligence*, com especial ênfase para o conhecimento adquirido com cubos e análises OLAP e aprendizagem de uma nova linguagem, MDX.

Em suma, considera-se que o presente estágio correspondeu às expectativas, para além de possibilitar que a estagiária aplicasse e apurasse, na prática, os conhecimentos teóricos adquiridos ao longo do seu percurso académico e se preparasse para o mundo empresarial, afigurando-se como uma peça chave na formação de uma futura Engenheira Informática.

7.2 Trabalho futuro

Criadas as bases para o módulo Económico-financeiro/Gestão Orçamental e dada a permanência da estagiária no projeto, é previsível que este módulo venha a incluir novos indicadores e incorpore, finalmente, dados do SAS.

Existe também a possibilidade de se consolidar o processo de ETL, aplicando abordagens seguidas na fase final do desenvolvimento (e que só foram possíveis graças à experiência contínua), nomeadamente a de pré-calcular valores acumulados em vez de permitir que o servidor OLAP (*Mondrian*) o faça.

Para além disso, e dependendo do *feedback* de usabilidade, não será de descartar a possibilidade de melhorar alguns aspetos a nível de *design*, como por exemplo, melhoria dos gráficos e alteração do seu tipo de uma forma mais dinâmica.

Bibliografia

[1] *NoSQL*. http://nosql-database.org/, acedido em 2015-01-05.

[2] *MySQL*, 2014. http://www.mysql.com/, acedido em 2014-10-01.

[3] *PostgreSQL*, 2014. http://www.postgresql.org/, acedido em 2014-10-01.

[4] *The four categories of NoSQL databases*, 2014. http://rebelic.nl/2011/05/28/the-four-categories-of-nosql-databases/, acedido em 2015-01-05.

[5] Bhattacharjee, Arpita: *NoSQL vs SQL – Which is a Better Option?*, 2014. https://www.udemy.com/blog/nosql-vs-sql-2/, acedido em 2015-01-05.

[6] Bouman, R e J van Dongen: *Pentaho Solutions: Business Intelligence and Data Warehousing with Pentaho and MySQL*. Wiley, 2011, ISBN 9781118080474. https://books.google.pt/books?id=5nQv58N4wzQC.

[7] Caserta, Joe e Ralph Kimball: *The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data*. Wiley Publishing, Inc., 2004, ISBN 8126505540. http://books.google.com.br/books/about/THE_DATA_WAREHOUSE_ETL_TOOLKIT.html?id=KW9KmUBe2kQC&pgis=1.

[8] Casters, Matt: *Pentaho Data Integration vs Talend (part 1)*. http://www.ibridge.be/?p=150, acedido em 2015-01-08.

[9] Eeles, Peter: *Capturing Architectural Requirements*, 2005. http://www.ibm.com/developerworks/rational/library/4706.html, acedido em 12/01/2015.

[10] Elias, Diego: *A granularidade de dados no Data Warehouse*, 2014. `http://corporate.canaltech.com.br/materia/business-intelligence/A-granularidade-de-dados-no-Data-Warehouse/`, acedido em 2015-01-15.

[11] Espinosa, Roberto: *ETL´s: Talend Open Studio vs Pentaho Data Integration (Kettle). Comparative.*, 2010. `https://churriwifi.wordpress.com/2010/06/01/comparing-talend-open-studio-and-pentaho-data-integration-kettle/`, acedido em 2015-01-08.

[12] Fragoso, Beatriz Paiva: *Projeto DW-UC : Desenvolvimento de uma data warehouse para a Universidade de Coimbra – Área A*. Tese de Mestrado, Universidade de Coimbra, Coimbra, julho 2014.

[13] Gitlab B.V.: *GitLab*. `https://about.gitlab.com/`, acedido em 2015-01-20.

[14] Grady, Robert B: *Practical Software Metrics for Project Management and Process Improvement*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992, ISBN 0-13-720384-5.

[15] Jaspersoft Community: *JasperReports Server - Features*, 2015. `https://community.jaspersoft.com/wiki/jasperreports-server-features`, acedido em 2015-01-10.

[16] Kimball, Ralph e Margy Ross: *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley, 2ª edição, 2002, ISBN 0471200247. `http://books.google.pt/books?id=2OCbq8Azdm8C`.

[17] Kimball, Ralph *et al.*: *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*. Wiley, 2ª edição, 2011, ISBN 9781118079560. `https://books.google.pt/books?id=ONQio9do$\delimiter"026E30F$_70C`.

[18] Kumar, Girish: *Exploring the different types of NoSQL databases part II*, 2014. `http://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases`, acedido em 2015-01-05.

[19] Lang, Jean Philippe: *Redmine*. `http://www.redmine.org/`, acedido em 2015-01-20.

[20] Marchetto, Roberto: *Talend Open Studio vs Pentaho Kettle, a comparison*, 2011. `http://www.robertomarchetto.com/talend_studio_vs_kettle_pentao_pdi_comparison`, acedido em 2015-01-08.

[21] Mohanty, Asis: *ETL tool evaluation criteria*, 2012. `http://pt.slideshare.net/asismohanty/etl-tool-evaluation-criteria`, acedido em 2015-01-07.

[22] Passionned Group: *Selection criteria for ETL tools*, 2015. `http://www.etltool.com/categories-and-criteria-examined/`, acedido em 2015-01-07.

[23] Pentaho Community: *General FAQ*. `http://community.pentaho.com/faq/general.php`, acedido em 2015-01-09.

[24] Planet Cassandra: *NoSQL Databases Defined and Explained*, 2014. `http://planetcassandra.org/what-is-nosql/`, acedido em 2015-01-05.

[25] Postgresql: *PostgreSQL 9.3.5 Documentation*, 2014. `http://www.postgresql.org/docs/9.3/static/index.html`, acedido em 2014-10-03.

[26] Predictive Analytics: *25 Open Source and Free Business Intelligence Solutions*, 2015. `http://www.predictiveanalyticstoday.com/open-source-free-business-intelligence-solutions/`, acedido em 2015-01-09.

[27] Quora: *Which OpenSource ETL tool is easier to use & more agile? Pentaho Kettle/Jitterbit/Talend/Clover/Jasper/Rhino?* `http://goo.gl/x6X6BV`, acedido em 2015-01-08.

[28] Software Freedom Conservancy: *Git*. `http://git-scm.com/`, acedido em 2015-01-20.

[29] Solid IT: *DB-Engines Ranking of Relational DBMS*, 2015. `http://db-engines.com/en/ranking/relational+dbms`, acedido em 2015-01-24.

[30] Universidade de Coimbra: *Plano estratégico 2011-2015 da Universidade de Coimbra*, 2011. `http://www.uc.pt/planeamento/PEA_2011_2015_out2012.pdf`, acedido em 2014-11-15.

[31] Universidade de Coimbra: *UC-cloud - Infraestruturas e sistemas TIC para a cloud*. Relatório Técnico, Universidade de Coimbra, Coimbra, 2012.

[32] Universidade de Coimbra - DAMC: *Sistema de gestão da UC - Enquadramento*, 2014.

Anexos

Anexo A

Análise de riscos

A análise de riscos constitui uma série de passos que permite à equipa de desenvolvimento perceber, gerir e lidar com a incerteza.
Um risco é um problema potencial: tanto *pode* acontecer, como pode *não* acontecer. Independentemente do resultado, torna-se importante pensar à partida quais são os riscos que poderão vir a ocorrer durante o desenvolvimento, identificá-los, estimar a sua probabilidade de ocorrência, o seu impacto e pensar em planos de contingência por forma a poder mitigá-los e, assim, não colocar em causa o projeto.

Nesse sentido, a equipa UC-num elaborou, em conjunto, um documento com a análise de riscos (conteúdo infra citado), que contempla não só a ocorrência de um determinado risco no projeto em geral, mas também no módulo de **GO** (**G**estão **O**rçamental) em particular.

Código `RISCO_001`	**Tipo** `Indireto`	**Categoria** `Recurso`
Probabilidade `Alta`	**Impacto** `Médio`	**Data de identificação** `24/02/2015`
Designação	Atraso na especificação dos indicadores.	
Descrição	Atraso na conclusão de toda a especificação de indicadores.	
Plano de contingência	Iniciar desenvolvimento incremental, de acordo com os indicadores especificados até à data. Monitorização atenta e constante dos trabalhos desenvolvidos pelas equipas operacionais.	
Estado geral	Ocorreu e foi aplicado plano de contingência.	
Estado módulo GO	Ocorreu e foi aplicado plano de contingência.	

Código `RISCO_002`	**Tipo** `Indireto`	**Categoria** `Recurso`
Probabilidade `Alta`	**Impacto** `Alto`	**Data de identificação** `25/02/2015`
Designação	Disponibilização dos dados fonte.	
Descrição	Disponibilização não atempada dos dados fonte.	
Plano de contingência	Análise completa de todas as necessidades dos sistemas fonte (exemplo: documentação com todos os atributos necessários). Iniciar desenvolvimento à medida que vão sendo disponibilizados dados (independentemente da prioridade do requisito). Solicitar dados fictícios enquanto os reais não estão disponíveis.	
Estado geral	Ocorreu e foram aplicados o primeiro e segundo pontos do plano de contingência.	
Estado módulo GO	Ocorreu e foram aplicados o primeiro e segundo pontos do plano de contingência.	

Código `RISCO_003`	**Tipo** `Direto`	**Categoria** `Recurso`
Probabilidade `Média`	**Impacto** `Alto`	**Data de identificação** `25/02/2015`
Designação	Conhecimento das tecnologias utilizadas.	
Descrição	Falta de conhecimento dos elementos da equipa nas ferramentas utilizadas para o desenvolvimento.	
Plano de contingência	Auxílio constante dos elementos da equipa de desenvolvimento; Realização de workshops; Disponibilização e consulta de material bibliográfico.	
Estado geral	Ocorreu e foi aplicado plano de contingência.	
Estado módulo GO	Ocorreu e foi aplicado plano de contingência.	

Código `RISCO_004`	**Tipo** `Indireto`	**Categoria** `Técnico`
Probabilidade `Baixa`	**Impacto** `Alto`	**Data de identificação** `25/02/2015`
Designação	Limitações das tecnologias.	
Descrição	Problemas de implementação das tecnologias utilizadas.	

Plano de contingência	Atualizar ou reverter a versão das tecnologias. Corrigir os problemas no código fonte da tecnologia, só se aplica a tecnologias *open source*.
Estado geral	Ainda não ocorreu.
Estado módulo GO	Ainda não ocorreu.

Código `RISCO_005`	**Tipo** `Indireto`	**Categoria** `Recurso`
Probabilidade `Alta`	**Impacto** `Alto`	**Data de identificação** `25/02/2015`
Designação	Validação da aplicação.	
Descrição	Atrasos na validação da aplicação, pela equipa destacada para tal.	
Plano de contingência	Disponibilização atempada da aplicação. Acompanhar o estado do processo de validação juntamente com a equipa destacada.	
Estado geral	Ocorreu, sendo aplicado o segundo método de mitigação.	
Estado módulo GO	Ocorreu, sendo aplicado o segundo método de mitigação.	

Código `RISCO_006`	**Tipo** `Indireto`	**Categoria** `Recurso`
Probabilidade `Média`	**Impacto** `Alto`	**Data de identificação** `25/02/2015`
Designação	Especificação de indicadores inválida.	
Descrição	Erros na especificação dos indicadores (fórmulas de cálculo, agregações, filtros, parâmetros temporais) cometidos pela equipa de trabalho.	
Plano de contingência	Múltiplas validações da especificação de indicadores, efetuadas por diferentes stakeholders.	
Estado geral	Ocorreu e foi aplicado plano de contingência.	
Estado módulo GO	Ainda não ocorreu.	

Código `RISCO_007`	**Tipo** `Direto`	**Categoria** `Recurso`
Probabilidade `Baixa`	**Impacto** `Alto`	**Data de identificação** `25/02/2015`

Designação	Saída de elementos da equipa de desenvolvimento.
Descrição	Saída de elementos do projeto, causando instabilidade, atrasos.
Plano de contingência	Reorganizar o planeamento das tarefas e afetação dos responsáveis. Renegociação dos prazos com o cliente.
Estado geral	Ainda não ocorreu.
Estado módulo GO	Ainda não ocorreu.

Anexo B

Análise de Tecnologias

B.1 Motores de Bases de Dados

Esta secção apresenta informação mais detalhada ao nível das bases de dados relacionais selecionadas para análise e ainda informação complementar para caraterizar os motores de BD *NoSQL*. Pelos motivos apontados na secção 4.2.1 (página 33), é apenas feita uma distinção entre as categorias deste tipo de modelos na secção B.1.2.

B.1.1 Motores de Bases de Dados relacionais

Nesta secção é feita uma análise comparativa entre as diferentes características do *PostgreSQL* e *MySQL* (mais concretamente, *MySQL Community Edition*[1]), por forma a inferir aspetos positivos e negativos de ambos os motores de BD.

[1]Também existem versões comerciais, mais completas.

		MySQL	**PostgreSQL**
Plataformas suportadas	*Windows*	✓	✓
	OS X	✓	✓
	Linux	✓	✓
	BSD	✓	✓
	UNIX	✓	✓
	AmigaOS	✓	✗
	Symbian	✓	✗
	z/OS	✓	✗[a]
	iOS	?	✗
	Android	✓	✓
	openVMS	✗	✗
Licença		Open Source	Open Source
Índices secundários		✓	✓
Triggers		✓	✓
Integridade referencial		✓[b]	✓
ACID		✓[b]	✓
Escalabilidade		✗	✓
Alta disponibilidade		✗	✓
Backups		✗	✓
Mecanismo de *Lock*		MVCC[b]	MVCC
Suporte formato XML		✗	✓
Suporte para Java		✓	✓

[a] Possível compilando o *PostgresSQL* para Linux num Sistema Z (ver `http://www-03.ibm.com/systems/z/os/linux/about.html`).
[b] Com motor *InnoDB*.

Tabela B.1: *MySQL* vs. *PostgreSQL* – Caraterísticas gerais [29].

		MySQL	PostgreSQL
Tabelas temporárias		✓	✓
Vistas materializadas		✗	✓
Paralelização de *queries*		✗	✓
Índices	*R-Tree*	✓[a]	✓
	Hash	✓[b]	✓
	Expression	✗	✓
	Partial	✗	✓
	Reverse	✗	✓
	Bitmap	✗	✓
	GiST	✗	✓
	GIN	✗	✓
	Full-text	✓	✓
	Spatial	✓[a]	✓[c]
Particionamento Simples	*Hash*	✓	✗
	List	✓	✓[d]
	Range	✓	✓[d]
	Outros	✓[e]	✗
Particionamento Composto	*{Range + Hash}*	✓	✗
	Outros	✗	✗

[a] Apenas com o motor *MyISAM*.
[b] MEMORY, Cluster(NDB), apenas com o motor *InnoDB*.
[c] Implementado através de POSTGIS.
[d] Atualmente só é possivel através da herança de tabelas [25].
[e] *Columns* (*Range*|*List COLUMNS*); *Key*

Tabela B.2: *MySQL* vs. *PostgreSQL* – Propriedades relevantes para DW.

B.1.2 Motores de Bases de Dados *NoSQL*

As bases de dados *NoSQL* foram desenvolvidas para contornar algumas limitações das relacionais, nomeadamente no que concerne a escalabilidade, replicação de dados e performance ao lidar com enormes quantidades de dados (Bigdata) [5]. Atualmente, as bases de dados *NoSQL* dividem-se em quatro categorias [1, 4, 18]:

- Armazenamento chave-valor: a ideia é usar uma grande *hashtable*, onde a

uma chave única corresponde um valor. É o modelo mais simples e fácil de implementar, mas também o mais ineficiente quando interessa, por exemplo, apenas consultar ou atualizar parte de um valor – e.g., *Redis, MemcacheDB, Amazon DynamoDB, Voldemort.*

- Armazenamento por colunas: os dados são agrupados em células por colunas de dados, em vez de ser por linhas. Todas as células correspondentes a uma coluna são guardadas contiguamente no disco, tornando, assim, as pesquisas muito mais rápidas. Para além da elevada performance, este modelo também é altamente escalável, tendo sido criado para armazenar e processar grandes quantidades de dados distribuídos por várias máquinas – e.g., *Cassandra, HBase, Hypertable.*
- Armazenamento baseado em documentos: expande o modelo chave-valor, na medida em que cada chave única faz corresponder um "documento" (que consiste em dados semi-estruturados), permitindo, assim, valores imbricados associados a cada chave. Esses dados semi-estruturados são normalmente armazenados em formatos como XML, JSON, BSON e YAML – e.g., *MongoDB, CouchDB, RavenDB.*
- Baseadas em grafos: como o próprio nome sugere, baseia-se na teoria de grafos, em que as relações entre os dados são mais naturais, por oposição às bases de dados em que as relações fazem parte do esquema fixo. Este modelo assente em três componentes básicos: nós (também chamados de vértices, que correspondem às entidades), relações entre nós (ou arestas) e as propriedades (atributos) dos nós e das relações. A vantagem de recorrer a este tipo de modelo torna-se evidente quando se compara uma consulta que envolve várias junções de tabelas num motor de BD relacional por à oposição à simplicidade de se procurar essas relações em grafos - e.g., *Neo4j, Titan, InfiniteGraph.*

B.2 Ferramentas de ETL

Nesta secção é feita uma análise comparativa entre duas ferramentas de ETL, gratuitas e *open-source* [11, 20]:

1. *Pentaho Data Integration* (abrev. PDI);
2. *JasperETL* (*Talend*).

Analisando a Tabela B.3, constata-se que ambas as ferramentas têm um grande potencial, satisfazendo os requisitos necessários a um processo de ETL (c.f. de-

finidos na secção 4.2.2). Ambas são *user-friendly* [20], bem documentadas e com um forte suporte comunitário [27]. Ao nível das componentes, as duas apresentam uma grande variedade de escolha, sendo que ações/transformações para *input/output* são dependentes de cada motor de BD (e.g., *Oracle*, *MySQL*, *MongoDB*,...). Destacam-se, nos dois, a possibilidade de invocar *web-services* (que é uma imposição do módulo de Gestão Orçamental) e o suporte a vários formatos de ficheiros tipicamente usados nestes casos (e.g., CSV, XML, JSON,...). O *PDI*, vai mais além ao disponibilizar uma componente, *Job*, que não é mais do que um conjunto de transformações, sobre o qual assenta todo o projeto (é comum criar sequências de *Jobs* e executar esse *workflow* de modo automático, com a vantagem de ser tudo em modo gráfico).

A única diferença é que o *PDI* é um interpretador de procedimentos ETL armazenados em formato XML, enquanto que o *JasperETL* é uma ferramenta que gera código Java ou Perl. Por isso, o *JasperETL* será mais orientado a programadores, com um maior nível de perícia, por oposição à simplicidade do *Pentaho Data Integration*. Em contrapartida, a flexibilidade que o *JasperETL* oferece é absoluta. Por sua vez, o *PDI* é mais intuitivo e tem uma menor curva de aprendizagem, o que pode ser determinante na hora de escolher, sobretudo se existirem constrangimentos temporais ao nível do projeto.

A nível de desempenho, a escolha do *JasperETL* pode ficar comprometida dado o processamento paralelo ser limitado na versão gratuita e, talvez por isso, apresentar melhores resultados que o PDI apenas em ambientes *single threaded*. Assim, o PDI destaca-se ao nível da **performance**, por permitir muito bons resultados com processamento paralelo [8] – o que é uma mais valia quando se tratam grandes volumes de dados na área temporária.

	Pentaho Data Integration	**JasperETL**
Multiplataforma	✓	✓
Usabilidade	Curva de aprendizagem pouco acentuada; Interface gráfica muito intuitiva e simples, embora não unificada entre componentes.	Curva de aprendizagem mais acentuada; Interface gráfica pouco intuitiva, mas unificada para todos os componentes (baseada no Eclipse).

Tabela B.3: *PDI* e *JasperETL*: comparação (cont.).

	PDI	JasperETL
Scripting	■ Javascript (para cálculos e fórmulas); ■ Java; ■ SQL; ■ Shell; ■ Fórmulas do OpenOffice	■ Java; ■ Groovy; ■ SQL; ■ Shell.
Processamento paralelo	Aplica-se com grande facilidade, escolhendo a opção de "Distribuição de dados"entre os componentes.	Limitado (embora avançado nas versões pagas).
Metadados	Estão dependentes das ligações às BDs/fontes de dados, cujos metadados podem ser partilhados entre transformações e jobs; Armazenados por etapa (e não centralmente), não podendo ser reutilizáveis por/a partir de outros componentes;	Armazenados centralmente; Podem ser partilhados e reutilizados por outros componentes;
Repositórios	*Jobs* e Transformações são guardados em ficheiros XML, permitindo que os mesmos sejam armazenados ou no sistema de ficheiros local, ou numa BD para serem mais facilmente partilhados.	Todos os componentes de um projeto (Jobs, definições de metadados, código personalizado e contextos) são armazenados no sistema de ficheiros local. O repositório é atualizado com as dependências dos objetos alterados [2].
Reutilização de código	Não é possível reutilizar código entre componentes, pois este está definido para cada um deles de maneira individual.	É possível, porque podem incluir-se bibliotecas próprias, visíveis a todos os *Jobs* de um projeto, permitindo, assim a projeção e reutilização de componentes definidos pelo utilizador.

Tabela B.3: *PDI* e *JasperETL*: comparação (cont.).

[2] e.g., se se alterar a definição duma tabela no repositório, essa atualização vai ocorrer em todos os *Jobs* onde ela é usada.

	PDI	JasperETL
Log* e *debug	■ Diferentes níveis de *log* (do mais simples ao mais avançado); ■ Possibilidade de guardar os logs de registos na base de dados, mas muito limitado; ■ Uma única ferramenta de *debug*, muito simples e gráfica.	■ Sistema de *logs* muito avançado (distinção de vários tipos: de processamento ou os que fornecem estatísticas e métricas); ■ *Logs* podem ser enviados para BDs, ficheiros ou para a consola; ■ Para *debug* é possível recorrer à correspondente ferramenta gráfica do *Eclipse*.

Tabela B.3: *PDI* e *JasperETL*: comparação.

B.3 Ferramentas para análise de dados (OLAP)

Atualmente no mercado de BI existem diversas ferramentas direcionadas à análise OLAP e à disponibilização de relatórios e *dashboards* aos utilizadores. Nesta secção, serão, pelos mesmos motivos das anteriores, apresentadas a análise comparativa e informação complementar à secção 4.2.3 para as duas ferramentas gratuitas selecionadas – *JasperReports Server* e *Pentaho BI Server* (abrev. PDI).

	JasperReports Server	Pentaho BI Server
Iniciação		
Facilidade de Instalação	Razoável	Razoável
Relatórios-exemplo disponíveis	✓	✓
Requer conhecimento SQL?	✓	✓
Caraterísticas		
Input via: JDBC|XMLA|Flat file	✓|✓|✓	✓|✓|✓

Tabela B.4: *JasperReports Server* e *Pentaho BI Server*: quadro comparativo (cont.).

	Pentaho Data Integration	**JasperETL**
Múltiplas fontes de dados num relatório	✓	✓
Formatos de output: PDF|HTML| CSV	✓|✓|✓	✓|✓|✓
Templates de relatórios	✓	✓
Nº de gráficos disponíveis	20	14
Adição de imagens	✓	✓
Parâmetros de relatórios	✓	✓
Agendamento de relatórios	✓	✓
Envio de relatórios por email	✓	✓
Relatórios OLAP	✓	✓
Dashboards	✗	✓
Dashboards permitem fazer drill-down	–	✓
Dashboards criados pelo utilizador	–	✓
Segurança	✓	✓
Linguagem para consultas	MDX	MDX
Caching	✓, acesso muito rápido a dados.	✓, acesso a dados é mais lento que no PDI.
Ferramenta *front-end* para visualizar o output dos cubos OLAP	■ Pviot4J Analytics ■ Saiku Analytic's ■ JPivot View ■ BTable ou ■ Ivy Schema Editor	JPivot View
Suporte		
Posts frequentes no fórum	✓	✓
Documentação	✓, com muitos exemplos online.	✓, com muitos exemplos online.
Suporte comunitário	Muito ativo – respostas rápidas.	Não tão ativo, uma vez que não é um produto proprietário da Jasper.

Tabela B.4: *JasperReports Server* e *Pentaho BI Server*: quadro comparativo.

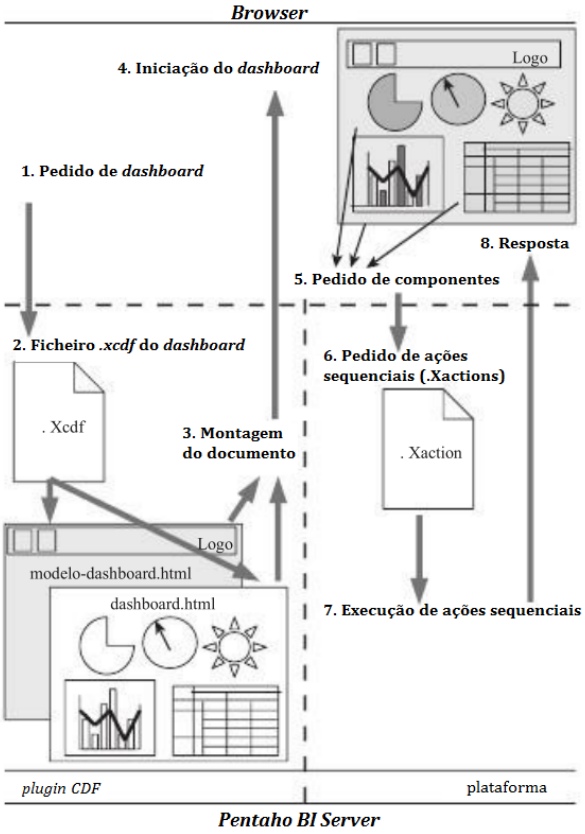


Figura B.1: Arquitetura Geral do CDF do *Pentaho BI Server*.

Os passos que melhor descrevem a forma como o *plugin Community Dashboards Framework* **(CDF)** se processa encontram-se na Figura B.1, resumidamente [6, 12]:

1) Utilizador faz o pedido HTTP para o *dashboard*;
2) Através do nome e caminho o servidor percebe que é um pedido para *dashboard* (ficheiro *.xcdf* – contém instruções HTML e *JavaScript* associadas a cada componente do *dashboard*);
3) A página *web* é renderizada e apresentada ao utilizador, iniciando assim o *dashboard*. Cada componente é posteriormente criado no *JavaScript* e fica associado ao objeto *Dashboards*, a partir do qual é possível realizar comandos como o *update*, que não é mais do que iniciar/atualizar determinado(s) componente(s);
4) O servidor recebe os pedidos anteriormente referidos e executa-os – são o que se denominam de *action sequences* ou Xactions – definidas em ficheiros XML que contêm uma sequência de instruções a ser realizada

sobre o(s) componente(s) a ser apresentados.

Anexo C

Modelos de dados

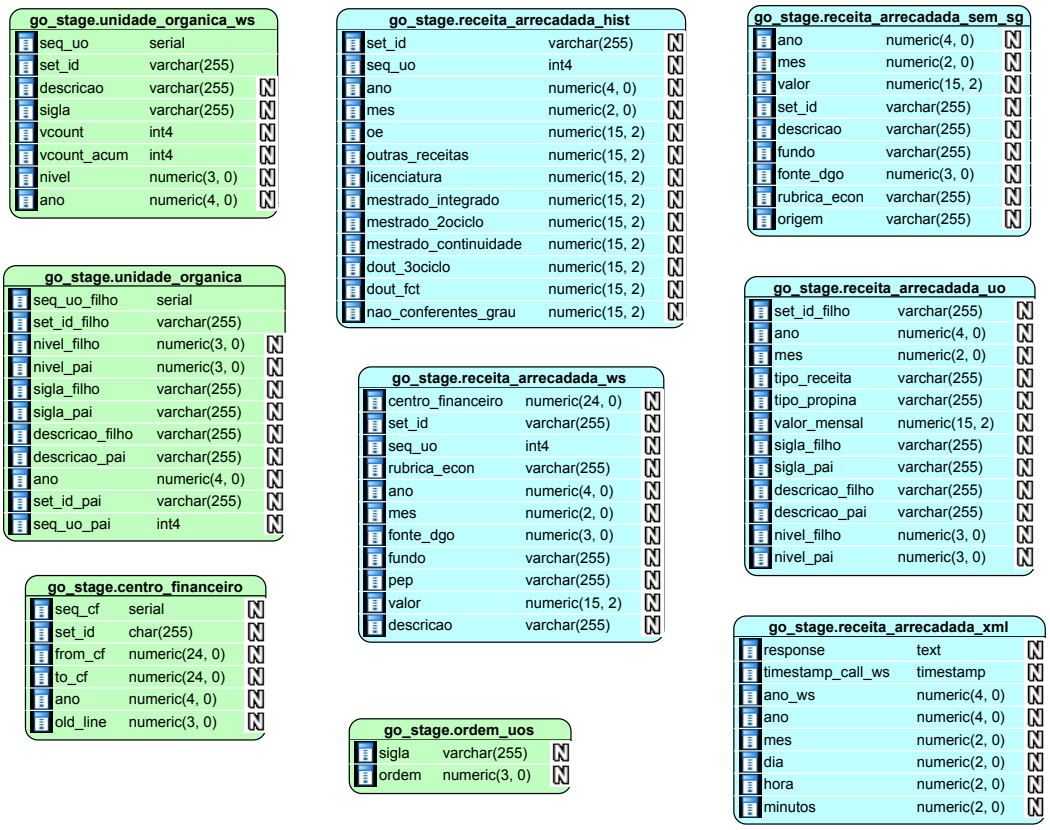


Figura C.1: Modelo concetual da área temporária (detalhado) – parte 1/3.

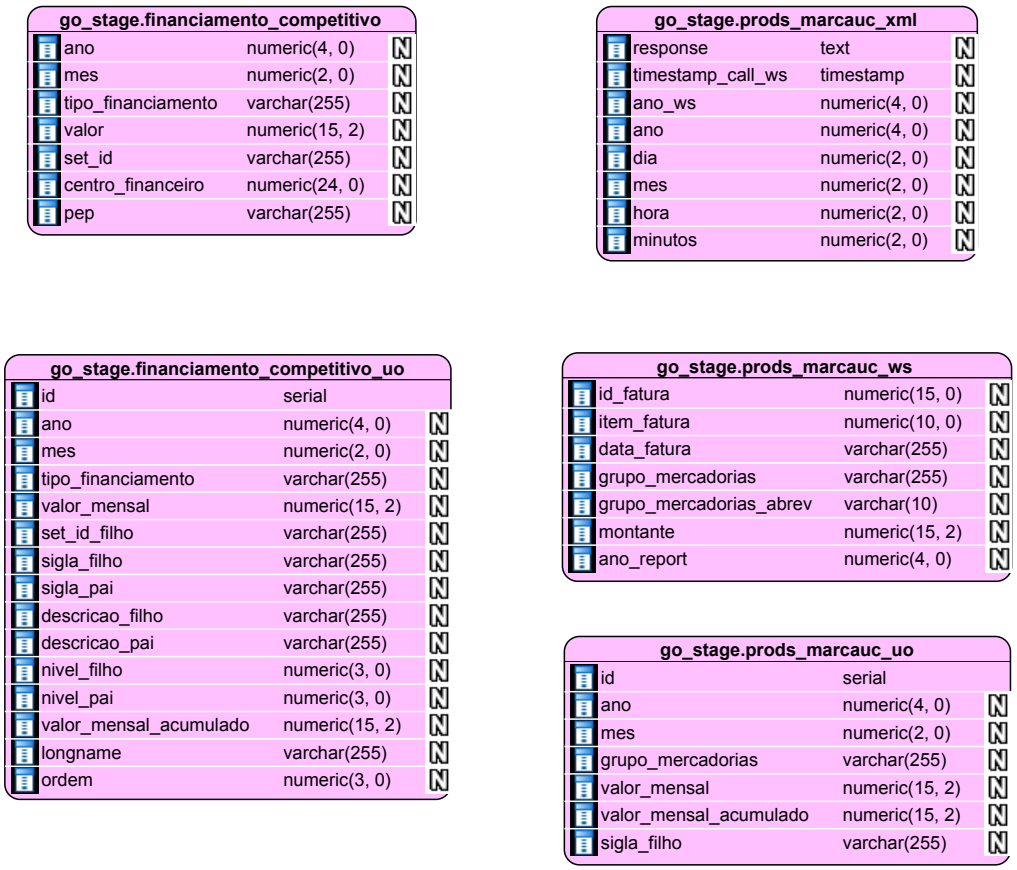


Figura C.2: Modelo concetual da área temporária (detalhado) – parte 2/3.

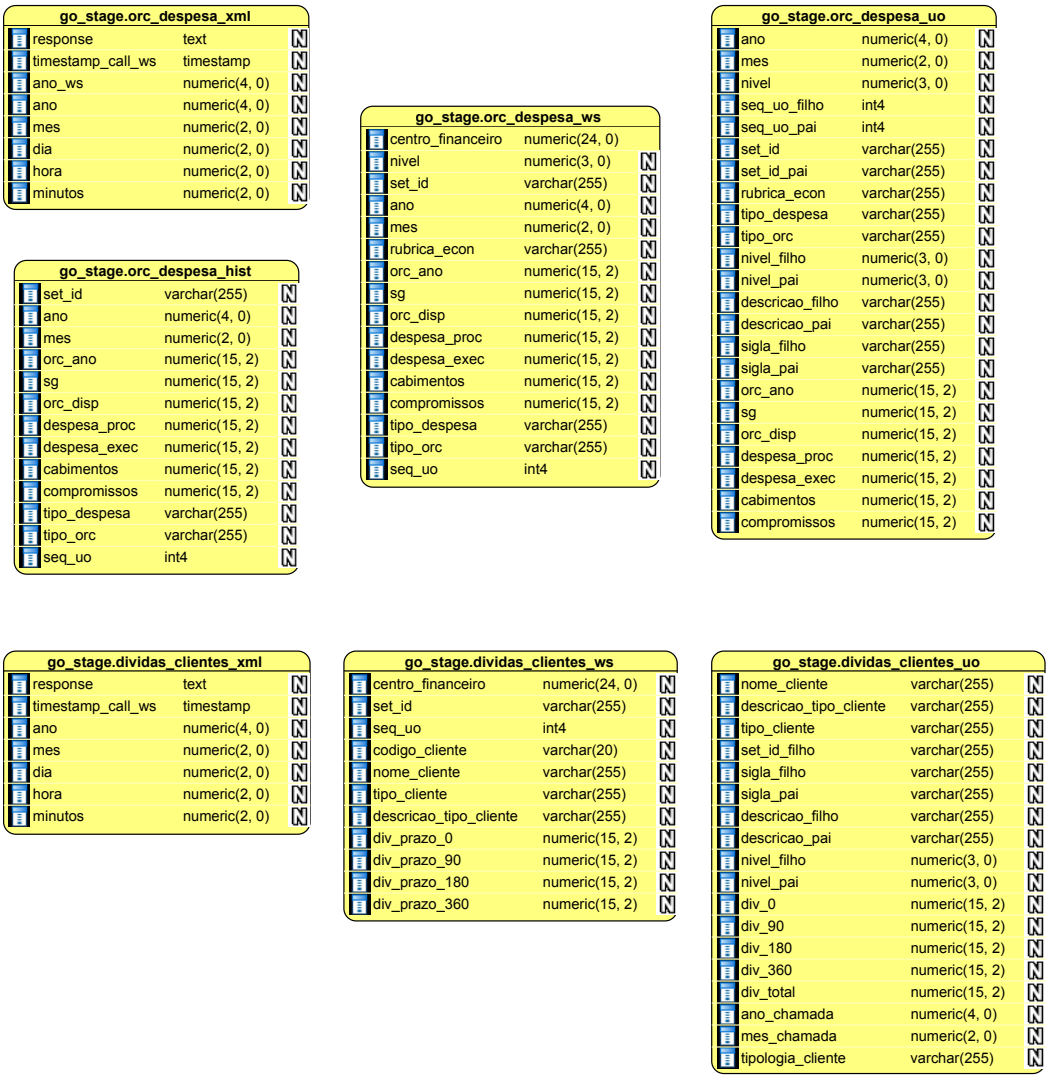


Figura C.3: Modelo concetual da área temporária (detalhado) – parte 3/3.

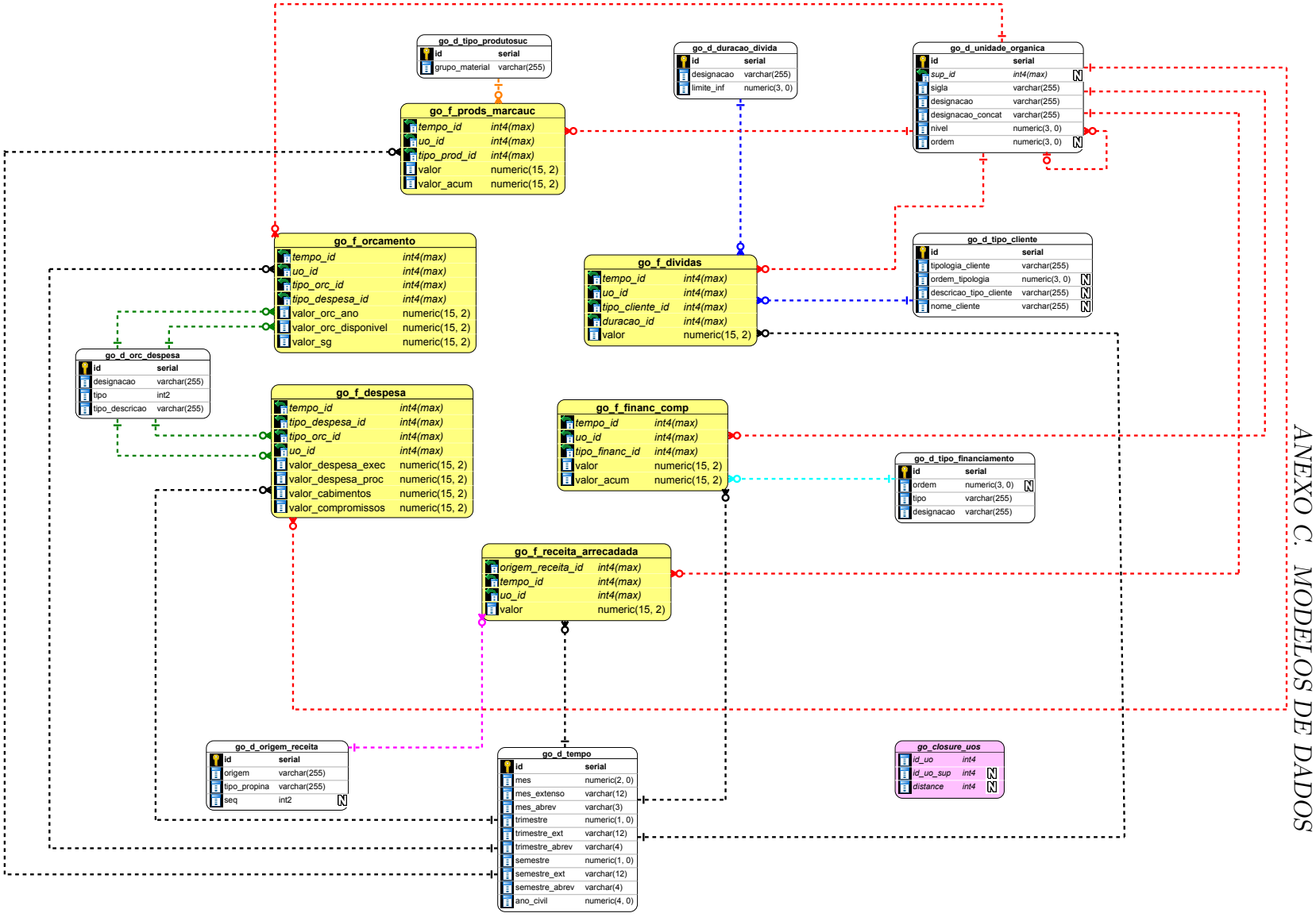


Figura C.4: Modelo multidimensional do *data mart* (detalhado).

Anexo D

Anexos digitais

Os anexos aqui listados são fornecidos em formato digital e encontram-se na pasta `ANEXOS`:

(1) UC-NUM_ARQ_arquitetura_guias.pdf

(2) UC-NUM_DSG_especificacao_design.pdf

(3) UC-NUM_ETL_GO-transformações_PDI

(4) UC-NUM_REQ_GO_fichas-indicadores.pdf

(5) UC-NUM_TST_GO_fichas-validacao-exemplo.pdf

(6) GO-modelo_datamart.pdf

Os primeiros dois documentos foram elaborados por toda a equipa e são indispensáveis a uma boa gestão e desenvolvimento do projeto.
Já os documentos de (3) a (6) são de autoria exclusiva da estagiária e têm conteúdo de apoio a esta dissertação, conforme referido em algumas secções.