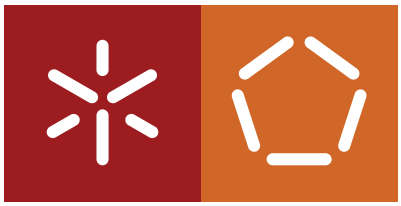


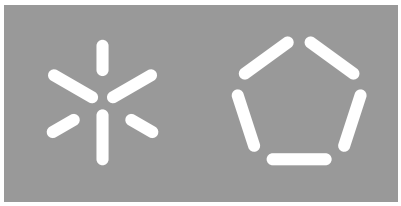
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Vasco António Pinheiro da Costa Abelha

Interoperabilidade Semântica

Caso de estudo em torno do ciclo de desenvolvimento
e produção de soluções multimédia avançadas para automóvel

30 de Outubro de 2015



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Vasco António Pinheiro da Costa Abelha

Interoperabilidade Semântica

Caso de estudo em torno do ciclo de desenvolvimento
e produção de soluções multimédia avançadas para automóvel

Dissertação
Mestrado em Engenharia Informática

Trabalho supervisionado por
Professor Doutor José Manuel Ferreira Machado

30 de Outubro de 2015

D E C L A R A Ç Ã O

Nome: Vasco António Pinheiro da Costa Abelha
Endereço eletrónico: vascoabelha91@gmail.com
Cartão de Cidadão: 13771713

Título da Dissertação: Interoperabilidade Semântica - Caso de estudo em torno do ciclo de desenvolvimento e produção de soluções multimédia avançadas para automóvel

Orientador: Professor Doutor José Manuel Ferreira Machado

Ano de conclusão: 2015

Designação do Mestrado: Mestrado em Engenharia Informática

Universidade do Minho, __/__/__
Assinatura: _____

AGRADECIMENTOS

Gostaria de agradecer em primeiro lugar ao meu orientador, Professor Doutor José Machado pela disponibilidade prestada e pela ajuda na elaboração desta dissertação.

Não menos importante, quero prestar agradecimento ao Professor Doutor Paulo Martins, responsável pela Bolsa de Investigação, pelo tempo e pela paciência disponibilizada durante os 9 meses.

Gostaria também de agradecer aos meus amigos de longa data e aos novos amigos que conheci durante a Bolsa de Investigação. Não sei o que seria de mim sem os jogos de futebol à terça-feira. Não vou estar a enumerar, porque vocês sabem quem são.

Não posso deixar de agradecer à minha família: aos meus pais, aos pais da Sara, aos meus irmãos, às minhas avós por terem participado, aturado e ajudado, de uma maneira ou outra, no desenvolvimento deste projeto.

Obrigado Rebeca pela tua companhia nas longas noites dedicadas ao desenvolvimento deste projeto.

Como o melhor fica sempre para último, obrigado Sara Jorge Soares Carneiro. Obrigado por me ajudares a ser um ser humano melhor, tanto a nível profissional como pessoal. Obrigado por tudo.

ABSTRACT

Over the last few decades, there has been a huge evolution of the complexity of technological systems. Currently, these systems are composed by thousands of different components that trade, between them, colossal amounts of information. Nevetheless and in most of the cases, such complexity leads to a decrease in terms of efficiency, sustainability and persistance of data. If all components were fully autonomous agents of the context in which they operate, then there would be no problem. However, this is far from reality, since there will always exist an interdependence between the components of a particular system, residing, still, the necessity to transmit a set of data from one entity to another completely different.

Given this, we question how communication maybe possible between the different agents. This paradigm can be metaphorically represented by the game of Broken Telephone, in which the message, transmitted between the various participants of different nationalities, reaches the end of the game completely distorted.

Inevitably, these distorted messages may eventually cause problems during decison-making and procedures where the level of costs, efficiency and reliability of information are of utmost importance. This can range from a simple act of caring for a patient - Medicine - to the production of thousands of radios circuit board per hour.

Thus, it can be seen that this is a problem which overwhelms daily the technological world in which we live. Urging the need to obtain a solution - Interoperability.

RESUMO

Nas últimas décadas tem-se verificado uma grande evolução da complexidade dos sistemas tecnológicos. Atualmente, estes apresentam milhares de componentes completamente diferentes que trocam quantidades colossais de informação. Tal complexidade leva, na maioria dos casos, a um decréscimo da eficiência, sustentabilização e persistência de dados. Se todos os componentes fossem agentes completamente autónomos do contexto em que estão inseridos, então não haveria qualquer problema. Porém, isto está muito longe da realidade, uma vez que haverá sempre uma interdependência entre os componentes de um sistema, residindo a necessidade de transmitir um conjunto de dados de uma entidade para uma outra completamente diferente.

Perante isto, questiona-se de que forma a comunicação poderá ser possível entre os variados agentes. Este paradigma poderá ser metaforicamente representado por um jogo do telefone estragado, no qual a mensagem transmitida entre os vários elementos participantes, de nacionalidades diferentes, chega ao fim do jogo completamente deturpada.

Inevitavelmente, estas mensagens deturpadas acabam por acarretar malefícios aquando de tomadas de decisão e procedimentos em que os níveis de custo, eficiência e fiabilidade da informação sejam de extrema importância. Isto pode ir desde o simples ato de cuidar de um paciente - Medicina - até à produção de mil placas de rádio por hora, produção em escala.

Sendo assim, pode-se verificar que isto é uma problemática que avassala diariamente, e cada vez mais, o mundo tecnológico em que vivemos, urgindo a necessidade de se obter uma solução para a mesma - Interoperabilidade.

CONTEÚDO

LISTA DE FIGURAS

Lista de Figuras

TERMINOLOGIA

Notação Geral

- Itálico - Termo, expressão ou palavras estrangeiras;
- Negrito - Salientar conceito.

Acrónimos

AS Análise Semântica

BLOB Binary Large Object

CLOB Character Large Object

DCMI Dublin Core Metada Initiative

DQL Doctrine Query Language

ERP Enterprise Resource Planning

FLEX Fast Lexical Analyzer

FTP File Transfer Protocol

FOAF Friend of a Friend

GB Gigabyte

GC Gestão de Conhecimento

GM Gestão de Memória

HTTPS Hyper Text Transfer Protocol Secure

IS Interoperabilidade Semântica

ISO International Organization for Standardization

IT Interoperabilidade Técnica

JSON JavaScript Object Notation

JSON-LD JavaScript Object Notation - Linked Data

LIA Linha de Inserção Automática

LOB Large Object

MB Megabyte

MOO Modelo Orientado ao Objeto

NCLOB National Character Large Object

NLS National Language Support

OIB Operations Information Broker

OSI Open Systems Interconnection

OWL Web Ontology Language

PCB Printed Circuit Board

PDA Personal Digital Assistant

PIA Preparação de Inserção Automática

PL/SQL Procedural Language/Structured Query Language

RDF Resource Description Framework

RDFA Resource Description Framework in Attributes

RDFQ Resource Description Framework Query

RDFS Resource Description Framework Schema

SAP System, Applications and Products

SI Sistema de Informação

SIE Sistema de Informação Executiva

SIIA Sistema de Informação de Inserção Automática

SIPIA Sistema de Informação de Programação de Inserção Automática

SKOS Simple Knowledge Organization System

SMC Smart Manufacturing Control

SMD Surface-mount Device

SMT Surface-mount Technology

MIME Simple Mail Transfer Protocol/Multi-Purpose Internet Mail Extensions

SOAP Simple Object Access Control

SOL Sistema de Operações Logísticas

SPARQL SPARQL Protocol and RDF Query Language

SPT Sistema de Processamento de Transações

SQL Structured Query Language

SSL Secure Sockets Layer

SSD Sistema de Suporte à Decisão

TCP/IP Transmission Control Protocol/Internet Protocol

VPN Virtual Private Network

W3C World Wide Web Consortium

XML eXtensible Markup Language

YACC Yet Another Compiler-Compiler

YAML YAML Ain't Markup Language

1

INTRODUÇÃO

A Seleção Natural, processo proposto por *Charles Darwin* e *Alfred Wallace*, identifica a diversificação como o principal fator evolutivo das espécies. Atualmente, a diversidade e a evolução, epígrafes da seleção natural, têm um sentido mais lato na nossa sociedade, estando implícitas em tudo o que nos rodeia, desde a simples caneta até ao robô Philae.

Nos dias de hoje, o diferente, o único, o "fora da caixa" é idolatrado, reverenciado e recompensado. No âmbito da tecnologia, a diversificação peca pelo excesso, sendo múltiplas vezes esquecido o contexto da população-alvo e do verdadeiro significado da Informática - ajudar o ser humano.

A maioria das empresas elabora equipamentos individualizados com caraterísticas muito próprias. Contudo, quando é necessário existir comunicação entre as diferentes ferramentas inerentes de cada um destes equipamentos, tal não se torna possível. Desta forma, surge uma necessidade extrema de criação de ferramentas de alto nível abstrato que permitem a interoperabilidade entre as mais diversas entidades, facilitando e ajudando os consumidores de vários produtos.

Aqui os consumidores não representam somente um simples indivíduo. Esta palavra adquiriu um sentido mais lato, estendendo-se a empresas multinacionais, bem como a setores da nossa sociedade, p.e. Saúde e Finanças. Hoje em dia, um profissional de qualquer área, minimamente informatizada, utiliza 20 produtos de *software* diferentes para realizar 20 tarefas que estejam inter-relacionadas. Tornando-se, assim, no verdadeiro Homem dos Sete Ofícios.

O foco desta dissertação passará por clarear o *modus operandi* do mundo industrial, iluminar o leitor para os problemas que daí advém e apresentar e discutir soluções ao nível da Interoperabilidade. Resultados e conclusões que possam auxiliar no desenvolvimento futuro de novas ferramentas que integrem um conjunto de componentes heterogéneos e na sua posterior implementação. Esta tese está inserida no âmbito da Bolsa de Investigação da Universidade do Minho, *HMI EXCEL-I&D crítica em torno do ciclo de desenvolvimento e produção de soluções multimédia avançadas para automóvel*. Mais concretamente, um sistema de controlo de produção inteligente.

1.1 CONTEXTUALIZAÇÃO E ENQUADRAMENTO

A indústria já não se resume à exploração dos metais e ao carvão. Atualmente, a indústria está presente em todas as áreas da nossa sociedade, desde a agricultura até às ciências farmacêuticas, afetando direta ou indiretamente o nosso quotidiano, o avanço tecnológico, *et cetera*. Chega a ser o motor da nossa sociedade. No entanto e tendo como pano de fundo o mundo e a conjetura tecnológica em que nos encontramos, podemos constatar que as roldanas, as ferramentas utilizadas, na maioria dos casos, pela indústria carecem de organização, modernização e inovação necessária para resolver com eficácia os obstáculos que se apresentam. Não obstante ao facto de já estarmos há mais de meio século na Terceira Revolução Industrial. Revolução responsável pela inserção de novas tecnologias e aprimoramento constante das mesmas de modo a melhorar a qualidade do produto e eficiência dos processos.

É importante ressalvar que a evolução tecnológica está presente na industria. O problema reside na organização da mesma. Isto pode-se facilmente verificar ao nível dos Sistemas de Informação.

Decompondo a definição de Sistema de Informação que se segue [1]:

Sistema de Informação: *conglomerado de fontes de informação digitais, interoperáveis e acessíveis por protocolos específicos.*

podemos verificar que esta não se aplica inteiramente no mundo empresarial, no desenvolvimento de sistemas informáticos industriais. [2]

Em regral geral, as fontes de informação coadunam-se com a realidade tecnológica. O papel, as capas, a informação física foram substituídas por documentos de texto, folhas de cálculo e base de dados. Aumentando, assim, a robustez, a persistência e a contextualização dos próprios dados.

As ferramentas de acesso estão inerentes às fontes de informação. Tanto pode ser um webservice baseado em *SOAP*, como o uso de *SQL Developer* no caso da base de dados *Oracle*, ou até, a utilização do *Microsoft Office* aquando da consulta de dados em folhas de texto ou de cálculo. Por norma, existe sempre uma ferramenta de acesso para cada fonte de informação, mas tal pode nem sempre acontecer, como será explicado mais tarde aquando do estudo da arquitetura da *Bosch* Braga.

A interoperabilidade, o cruzamento de dados de vários pontos de acesso diferentes, é a raiz dos problemas de organização previamente referidos. Atualmente, e na maioria dos casos, o cruzamento de dados, a extração de conhecimento está assente em processos manuais, onde, em casos extremos, o próprio resultado é uma folha de cálculo enviada por *mail* para o interessado em questão. Além de ostentar uma extrema ineficiência ao nível de processos/protocolos empresariais, isto acarreta também uma maior predesposição para a falha humana, o que pode trazer grandes malefícios à empresa, tanto a nível económico como de ambiente de trabalho.

Aplicando a mesma definição de sistema de informação ao Universo *Bosch*, podemos constatar que não se denota alterações relevantes. A informação encontra-se digitalizada e dispersa por vários

suportes de armazenamento de dados. Cada fonte de informação tem, também, o seu respetivo canal de acesso. E por fim o problema da interoperabilidade mantêm-se.

Estas temáticas ainda ganham mais relevância e importância quando inseridas num projeto de grande dimensão, onde o fluxo de informação é constante e enorme, como por exemplo um sistema de controlo de produção baseado em *e-kanban*. O estudo deste exemplo no âmbito da bolsa de investigação adquire ainda um maior destaque, visto que o objetivo da mesma passa pelo desenvolvimento de um sistema de suporte ao planeamento, à programação e ao controlo de produção. Um *software* de gestão de produção baseada na filosofia *e-kanban* - *SMART Manufacturing Control* - incluindo, também, a monitorização dos postos nas linhas de produção, aramazéns e *milkruns*.

Facilmente se pode, então, verificar que um projeto desta complexidade requere quantidades colossais de informação. Um fluxo contínuo de dados a fim de serem manipulados e processados tendo em conta o contexto em que estão inseridos. Isto acaba também por exigir, uma estrutura que consiga copular a maior quantidade e variedade possível de informação originária de diferentes pontos de origem. Assim sendo, torna-se necessário, e de máxima importância, o estudo e mapeamento do sistema informático da *Bosch*-Braga. Mais concretamente, o sistema que serve de suporte à produção, a fim de se encontrarem soluções para facilitar e agilizar todo este processo de comunicação - fluxo de dados - entre os vários pontos de informação e o sistema de gestão de produção inteligente - *SMC*. Posteriormente, já com um conhecimento extenso do sistema informática da unidade *Bosch*-Braga, pode-se dar início à discussão e desenvolvimento de soluções de modo a garantir a persistência do fluxo de dados e desempenho da plataforma.

1.2 MOTIVAÇÃO

A motivação não se prende ao percurso académico, ao estudo de novas tecnologias e produtos de *software*, nem de novos princípios de arquiteturas. Graças à tecnologia, a industria tem vindo a tornar-se mais complexa e competitiva. Um mundo repleto de vários sistemas, equipas e profissionais a trabalhar em conjunto para não serem ultrapassados. Um mundo em que o conhecimento, a posse de informação e a rapidez com que essa informação é tratada, é cada vez mais importante e fulcral no sucesso de uma empresa - maximização de processos de produção. Já não basta produzir e vender. Hoje pensa-se, testa-se e só depois se equaciona em produzir, verificando se traz vantagens a curto e longo prazo.

Assim sendo, pode-se afirmar que a motivação estende-se à curiosidade do mundo industrial e empresarial, bem como ao estudo, criação e desenvolvimento de soluções que possam potenciar a maximização de processos. Um maximização ao nível da extração de conhecimento, da interoperabilidade entre sistemas de informação e componentes tecnológicos.

Numa primeira fase, começou-se por estudar o *modus operandi* da Unidade da *Bosch* em Braga. O estudo dos vários sistemas de informação e a forma como estes atuam entre si e com os colaboradores

da *Bosch*, por outras palavras, o fluxo de informação. A identificação, caracterização e classificação do fluxo de informação existente no sistema informático da *Bosch* Braga.

Embora tenha participado nas diferentes fases deste projeto, após a auditoria inicial realizada ao sistema informática, e ,de acordo, com os objetivos, a situação real, e o próprio contexto da bolsa de investigação, procedeu-se à ponderação de soluções realistas para os problemas existentes, bem como à distruição de tarefas pela equipa da bolsa de investigação. O resultante da organização de tarefas, culminou na criação de uma ferramenta de conversão de base de dados *Firebird* para *Oracle* - um 'compilador' de base de dados - já que um dos principais problemas aquando do desenvolvimento da plataforma *SMART* prendeu-se com o desempenho e funcionalidades do motor *Firebird*.

1.3 OBJETIVOS

Ao contrário do que se possa pensar, a motivação não é o início de uma etapa, nem muito menos o percurso. A motivação é o fim, o objetivo final que se pretende alcançar. O próprio ato de realizar um sonho é uma motivação, mas como é que se sabe se o sonho foi alcançado?

Através de condições, requisitos, da delineação de um conjunto de objetivos que possam afirmar que a motivação foi alcançada. No âmbito desta dissertação, a questão principal que se coloca é:

É possível replicar uma base de dados Firebird em Oracle através da utilização do tradutor desenvolvido?

Porém, a só utilização deste requisto não é suficiente para verificar o sucesso desta dissertação. Todavia ainda é necessário verificar se:

1. De que forma o estudo realizado no sistema informático da Unidade da *Bosch* permitiu melhorar o Sistema de Informação utilizado na plataforma *SMART Manufacturing Control*?
2. Quais as vantagens do desenvolvimento e utilização da ferramenta de tradução em comparação à conversão manual?
3. De que forma a utilização deste tradutor melhora a plataforma *SMART*?
4. Quais são as funcionalidades que distinguem este tradutor dos milhares publicados na *internet*?
5. Verificou-se um decréscimo no tempo necessário para a replicação e conversão da dase de dados *Firebird*?

1.4 ESTRUTURA

De modo a facilitar a compreensão deste documento, optou-se por o organizar em 7 capítulos:

1. Introdução;
2. Estado da Arte;
3. Universo Braga-*Bosch*;
4. Materiais e Métodos;
5. Tradutor;
6. Apresentação e Discussão dos Resultados;
7. Conclusão.

Além disso, como se poderá reparar, a organização dos capítulos e da respetiva informação foi preparada de modo a facilitar a leitura, a compreensão e o relacionamento dos conhecimentos. Em vez de restringir a apresentação de conteúdo teórico ao capítulo 2, **Estado da Arte**, optou-se por sistematizar a enumeração dos conhecimentos conforme a sua apresentação, a sua utilização e necessidade. Isto permite uma leitura mais fluída e contextualizada

Por fim, é importante ressalvar que a organização dos capítulos segue um fio condutor, uma cadência própria do geral para o particular. Denota-se uma maior especificação e descrição de conhecimentos à medida que se vai aprofundando a leitura.

O presente capítulo, **Introdução**, está fundamentando na enumeração dos objetivos e motivos que serviram de impulso à escolha do tema e posterior desenvolvimento desta dissertação, bem como o enquadramento e contextualização da mesma.

O capítulo 2, **Estado da Arte**, focar-se-á, principalmente, no estudo e apresentação de conhecimentos teóricos relevantes para a compreensão desta dissertação. Numa primeira etapa, os conceitos apresentados estão relacionados com o estudo e posterior mapeamento do sistema de informático presente na Unidade de Produção da *Bosch* em Braga, mais concretamente o conceito de Interoperabilidade e os seus respetivos níveis e soluções. Por fim, torna-se também importante a descrição do que é um sistema de informação e os seus respetivos tipos, visto que o próprio sistema informático da *Bosch* é um aglomerado de sistemas de informação.

Por conseguinte, o capítulo 3, **Universo Braga-Bosch**, consiste no estudo do sistema informático, referido anteriormente. Esta é, de certa forma, a base da dissertação, já que o objetivo principal advém do levantamento e mapeamento do sistesma informático - a criação de uma ferramenta de conversão de base de dados *Firebird* para *Oracle*. Aqui será descrito, na íntegra, o sistema informática envolvido no desenvolvimento da plataforma *SMART Manufacturing Control*, bem como os seus respetivos componentes.

No capítulo 4, **Materiais e Métodos**, o objetivo da dissertação está clarificado, logo já se pode proceder à descrição e enumeração dos materiais e princípios a serem utilizados aquando do desenvolvimento do compilador. De certa forma, isto vai de encontro ao que se referiu anteriormente, a sistematização de conhecimentos conforme o contexto da informação. Este capítulo pode assemelhar-se ao Estado da Arte, uma vez que, também é realizada uma descrição e explicação das ferramentas utilizadas aquando do desenvolvimento do compilador e de outros conceitos fundamentais para a compreensão dattradução realizada ao nível dos motores de base de dados *Firebird* e *Oracle*.

Aprofundando-se na leitura, o nível de especificidade aumenta e, assim sendo, o capítulo 5, **Tradutor**, consiste na especificação do conceito de compilador - a apresentação e exposição do nível de tradução. Este capítulo resume-se à decomposição do processo de tradução em 3 fases e à posterior explicação de cada uma delas e as respetivas ferramentas usadas.

O capítulo 6, **Apresentação e Discussão de Resultados**, compreende a utilização do tradutor no contexto da dissertação - bolsa de investigação - mais concretamente fase de testes. Aqui proceder-se-á à exposição e elucidação dos vários exemplos realizados na fase de testes. Ou seja, de uma forma sucinta, serão expostos os objetivos pretendidos e a forma como estes foram alcançados. No entanto, não foi possível apresentar o resultado da utilização do tradutor num meio de produção devido ao término do contrato na bolsa de investigação.

Por fim, o capítulo 7, **Conclusão**, terá como mote principal, um pequeno resumo do cômputo geral desta dissertação e da bolsa de investigação, bem como uma listagem e apreciação sobre as conclusões principais que se podem aferir do desenvolvimento deste projeto. Devido à modularidade e potencialidade do tradutor, também se achou relevante um delineamento do trabalho a realizar para o futuro no âmbito desta ferramenta.

2

ESTADO DA ARTE

Aqui se fará uma apresentação cuidada e detalhada dos vários conhecimentos técnicos e essênciais para a compreensão desta dissertação. Desde a definição genérica de interoperabilidade, passando pelos seus respetivos níveis e acabando nos sistemas de informação. Esta ordem permite, assim, uma leitura fácil e organizada dos elementos nucleares deste documento: Interoperabilidade e Sistemas de Informação.

2.1 INTEROPERABILIDADE

Genericamente, a Interoperabilidade pode ser traduzida pela possibilidade de comunicação entre diferentes sistemas tecnológicos com o intuito de, autonomamente, trocar informação e processar esses dados de acordo com a tarefa que lhes foi apresentada. Por outras palavras, a Interoperabilidade consiste na capacidade de vários sistemas independentes conseguirem trabalhar em conjunto tendo em conta o contexto presente. No caso da indústria, o contexto refere-se à junção de protocolos de comunicação, fontes de informação e sistemas de *software*, uma relação lógica de suporte ao sistema informático existente. Agora, conscientes desta definição, facilmente se consegue perceber o principal objetivo da implementação de camadas de interoperabilidades ao comparar duas realidades distintas:

- Universo *Bosch*-Braga sem Interoperabilidade (atual);
- Universo *Bosch*-Braga com princípios de Interoperabilidade.

2.1.1 *Universo Bosch-Braga sem Interoperabilidade*

Como se poderá observar mais à frente, a Unidade da Bosch-Braga não denota grandes desenvolvimentos ao nível de interoperabilidade. Os protocolos atuais de comunicação não são automáticos e requerem supervisão humana a fim de serem completados. A interoperabilidade existente advém da utilização de várias ferramentas para extrair a informação produzida seguida de *inserts* manuais em *tables*. Não há um fluxo contínuo e um processamento automático da informação que é criada num dia de atividade da Linha de Inserção Automática - *LIA*. Genericamente isto pode ser descrito pela imagem que se segue.

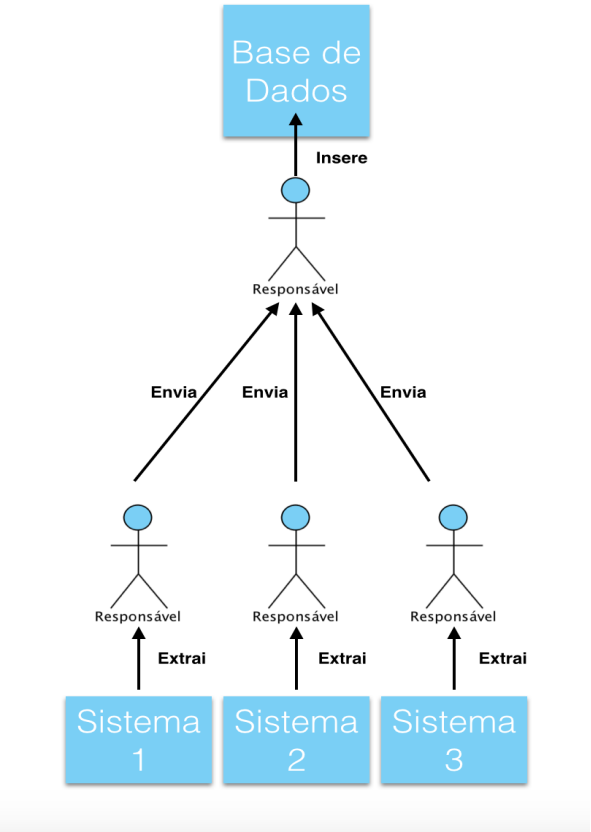


Figura 1: Universo *Bosch*-Braga sem interoperabilidade.

2.1.2 *Universo Bosch-Braga com princípios de Interoperabilidade*

O Universo *Bosch*-Braga interoperável resultaria num fluxo contínuo de informação em que o próprio processamento dessa informação seria independente de entidades responsáveis - processamento automático. O raio de acção do responsável resumir-se-ia à supervisão dos processos automáticos. Jamais teria que ser o próprio a realizar a extração de informação dos vários sistemas e a inserir no sistema de informação principal. O responsável tornar-se-ia num supervisor.

Este estado utópico é o que se pretende atingir com o desenvolvimento do *SMART Manufacturing Control*.

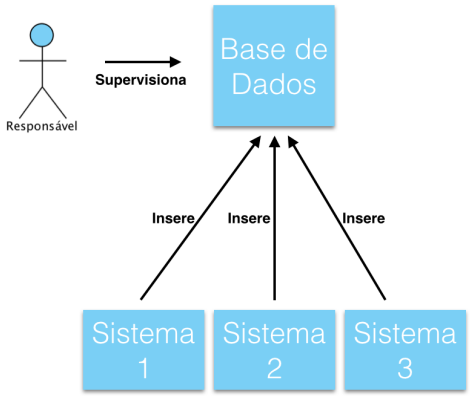


Figura 2: Universo *Bosch*-Braga com princípios interoperabilidade.

Neste caso, a interoperabilidade permite uma maior rapidez e eficácia através de uma partilha de dados contextualizados entre os vários sistemas de informação. Porém, esta eficiência só é possível com o último nível de Interoperabilidade - a Semântica. Por sua vez, a Semântica é referente ao estudo do significado, contextualização da informação. Assim sendo, podemos afirmar que a Semântica é 'Interoperabilidade de alto nível'. Uma Interoperabilidade em que não existe qualquer tipo de ambiguidade ou incerteza sobre o significado dos dados transmitidos. Facilitando assim o processamento e utilização dessa mesma informação. Todavia, não se pode menosprezar os restantes níveis, pois estes compreendem funções e formalidades necessárias tanto no panorâma geral, como, também, no sucesso da implementação da Interoperabilidade Semântica.

Com base na literatura, pode-se concluír que a Interoperabilidade pode ser dividída em 3 níveis, em que um nível *n* engloba as filosofias, princípios e funcionalidades do conjunto de níveis inferiores. Os 3 níveis de Interoperabilidade podem ser divídidos por [3, 4]:

- Interoperabilidade Semântica (nível 3);
- Interoperabilidade Sintática (nível 2);
- Interoperabilidade Técnica (nível 1).

2.1.3 *Interoperabilidade Técnica*

Interoperabilidade Técnica é responsável por todos os problemas relativos à comunicação, protocolos, ligação, serviços, *middleware* e segurança dos sistemas interoperáveis. Este paradigma acaba por incluír todas as camadas referentes ao modelo *OSI* [5].

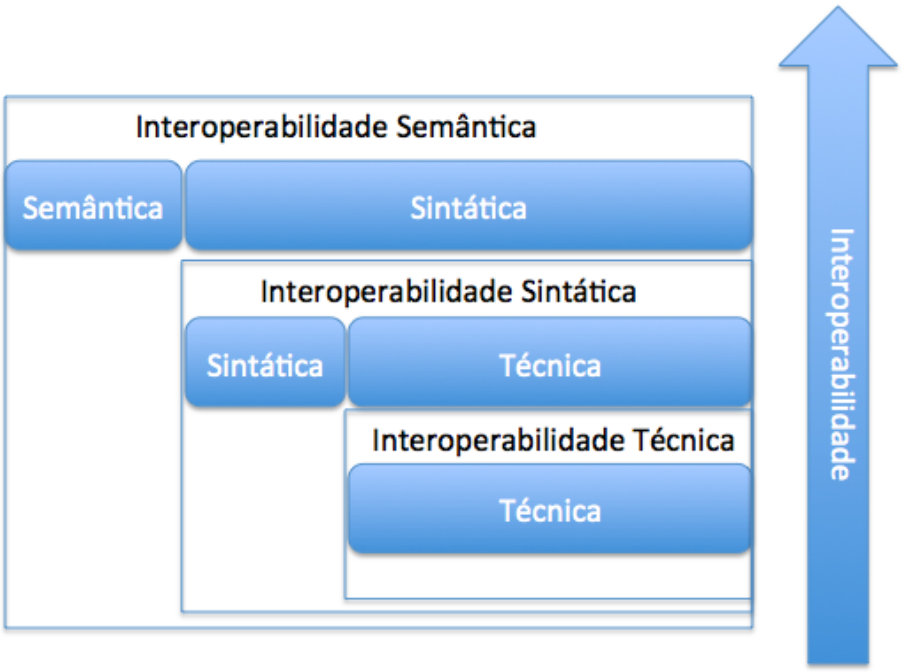


Figura 3: Níveis de interoperabilidade.

Open Systems Interconnection (OSI): *modelo conceptualizado pela ISO cujo principal objetivo é servir de modelo 'standard', framework para a criação de protocolos de comunicação. Permitindo assim conexão entre redes heterogéneas.*

A Interoperabilidade Técnica não apresenta qualquer obstáculo aquando da sua implementação. Atualmente existem uma panóplia de protocolos de comunicação ao nosso dispor, tais como:

- *TCP/IP*;
- *HTTPS*;
- *SMTP/MIME*;
- *FTP*, *SSL*;
- *et cetera*.

Ao nível da segurança e performance, a *IT*, também não apresenta qualquer tipo de lacuna:

- *VPN*;
- *Extranet*;

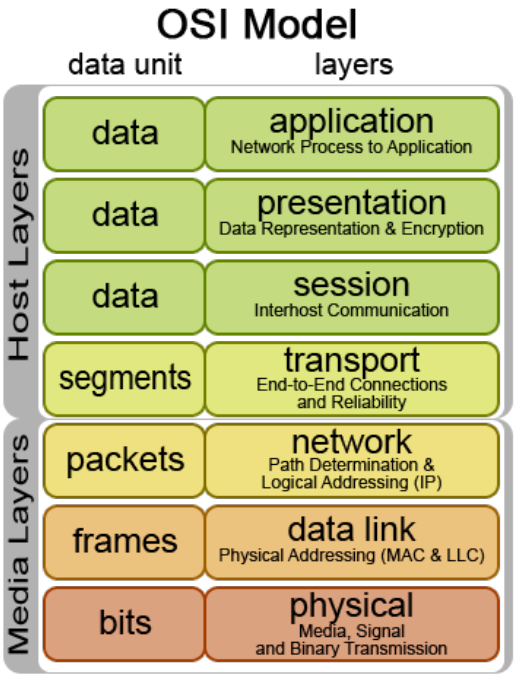


Figura 4: Modelo de *OSI*.

- *et cetera*.

Por fim, é importante reter que este tipo de interoperabilidade só é usado para garantir que a informação é transmitida corretamente sem qualquer tipo de precalços. Não existe qualquer tipo de relação com o conteúdo da informação. Esta funcionalidade pertence à camada seguinte: Sintática. [6, 7]

2.1.4 *Interoperabilidade Sintática*

Esta camada de Interoperabilidade está associada ao formato da informação, à sintaxe. A Interoperabilidade Sintática consiste na definição de estruturas imutáveis para organização de informação.

Em termos leigos, e fazendo anologia à linguagem humana, este tipo de interoperabilidade só confere mecanismos de leitura e escrita - uma gramática - ao sistema. A interpretação e processamento dos dados é remetido para o terceiro nível - Semântica. No entanto, esta etapa é crucial para a implementação de maiores níveis de interoperabilidade.

No cômputo geral podemos verificar que o *XML - eXtended Markup Language* - tem sido a linguagem mais escolhida no momento da organização de informação. [8] Tal escolha deve-se ao facto deste protocolo:

- Contextualizar informação através de *tags* - identificadores;

- Grande número de ferramentas *open source* - *XPath*, *XQuery*;
- Regras de validação - *XML Schema*;
- Orientada à informação;
- Estrutura Hierárquica;

bem como, por ter sido desenvolvida pela organização *World Wide Web Consortium*, cujo objetivo consiste em desenvolver protocolos e *guidelines* que asseguram o bom funcionamento e crescimento da *internet*. Porém, ultimamente, o *XML* tem vindo a ser substituído por soluções mais simples e promissoras, tais como:

- *JSON (JSON-LD)*;
- *YAML*.

Estas últimas duas opções distinguem-se pela sua simplicidade e manipulação dos dados. Simplicidade ao nível de escrita e leitura por parte do ser humano e da máquina. E, manipulação de dados na medida em que facilmente e rapidamente se pode criar e moldar a estrutura que melhor se coaduna com o contexto em que estámos inseridos e a sua respectiva informação.

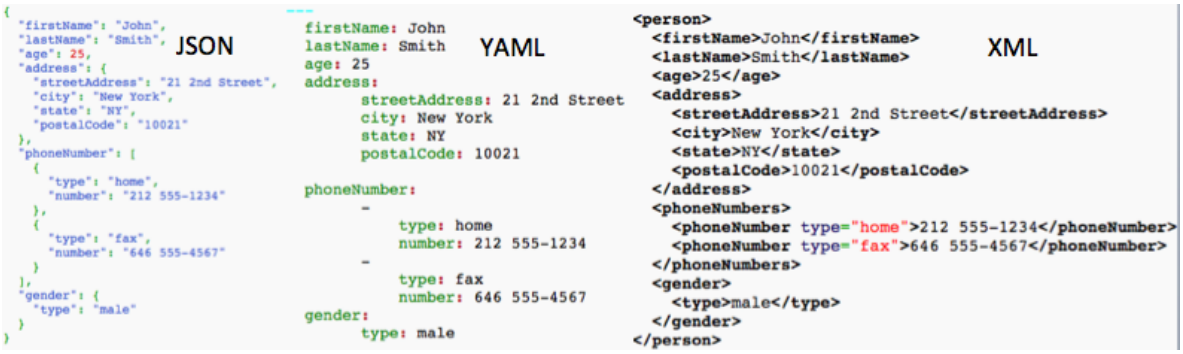


Figura 5: Comparação entre *JSON, YAML, XML*.

2.1.5 *Interoperabilidade Semântica*

Antes de mais, é importante ressalvar a pouca importância dada ao longo das últimas décadas ao campo da interoperabilidade semântica. A pesquisa e posteriores implementações, até então, têm-se focado em casos muito específicos e, alicerçados a ferramentas que não se coadunam com a conjetura tecnológica atual. Porém, esta tendência tem vindo a desaparecer graças aos efeitos colaterais da utilização da *internet* por parte dos utilizadores, *Pegada Digital*. Cada vez mais se verifica um aumento exponencial ao nível da informação gerada e posteriormente guardada ou partilhada, chegando ao

ponto de se estimar que 90% da informação existente na *Web* é datada dos últimos 2-3 anos [9]. Para além disso, a *internet* tem-se tornado num espaço de aprendizagem, conhecimento, globalização, onde num segundo podemos estar a investigar remotamente em Nova Iorque e ao mesmo tempo presentes numa conferência no Dubai. Por fim, as empresas tentam moldar e criar novas aplicações que se adaptem aos utilizadores. [10]

Mudam-se os tempos, mudam-se as vontades

Mudam-se os tempos, mudam-se as vontades,
Muda-se o ser, muda-se a confiança.
Todo o mundo é composto de mudança,
Tomando sempre novas qualidades.

Luís Vaz de Camões(1524-1580) in *Sonetos*.

Por mais difícil, por mais inevitável que seja o avanço, é necessário mudar face aos novos desafios, às novas vontades. Hoje em dia, esses desafios passam pela eficiência. O ser humano, devido ao contexto socio-económico, necessita de soluções mais viáveis em termos de tempo e custo. Soluções estas que passam pela automatização de processos ou um aumento em termos de eficiência desses mesmos processos, tais como:

- processamento de pedidos;
- pesquisa de informação específica;
- entre outros, dependendo do contexto em que estão inseridos.

A *internet* é o meio mais proliferativo para o avanço e estudo da Interoperabilidade Semântica, bem como o que mais precisa. Não há 'sistema' que tenha maior diversidade de informação e dados heterógeneos. Não há surpresa nenhuma ao afirmar que empresas como a *Google, Microsoft, Yahoo* e até mesmo o *Facebook*, não querendo ficar para trás, já se começaram a reforçar e adaptarem-se às mudanças que aí vêm, chegando ao ponto de alterarem profundamente o *modus operandi* das suas principais ferramentas, p.e. motor de pesquisa do *Google* [11]. As necessidades mudaram, os utilizadores mudaram, a exigência aumentou, a competividade aumentou. Um utilizador quando faz uma pesquisa em qualquer motor de buscar tenta ser o mais eficiente possível, escrevendo menos e querendo, ao mesmo tempo, receber mais informação E como é que isto será possível? *matching keywords* são suficientes? Qual é a relação entre a pesquisa de informação e interoperabilidade semântica?

É importante reter que a interoperabilidade semântica não se resume ao domínio da *internet*. [12] A *IS* torna-se fundamental em qualquer meio que esteja repleto de diversos sistemas heterogéneos e que exija um grande fluxo de informação, como por exemplo:

- espaço empresarial;
- espaço industrial;
- espaço financeiro;
- saúde.

Nestes últimos casos, a interoperabilidade semântica tem como objetivos [13, 14, 15]:

- reduzir custos;
- aumentar eficiência;
- auxílio aos utilizadores;
- automatizar processos;

É impossível definir interoperabilidade semântica sem se homenagear *Sir Tim Berners-Lee*, criador da *Web* e diretor do *W3C*. O termo semântica aplicado à computação e interoperabilidade remonta a Maio de 2001, altura em que *Lee*, inventou o termo *Semantic Web*. [16]

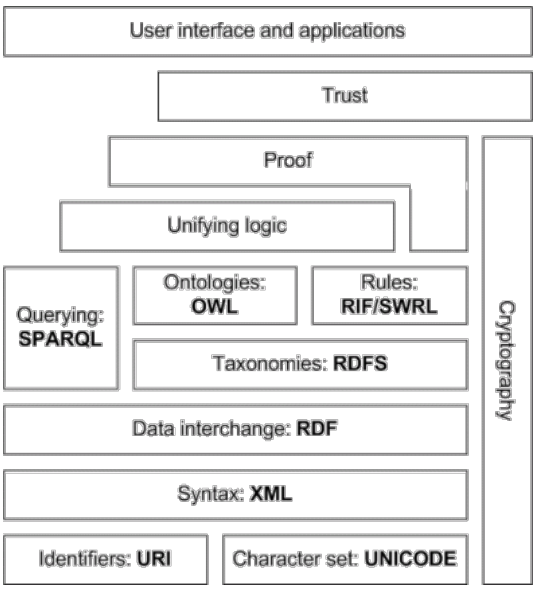


Figura 6: Layers referentes à *Semantic Web*.

Semantic Web: *é considerada uma extensão à internet dos dias de hoje, em que a informação é publicada com contexto e significado explicito, facilitando assim a leitura e compreensão da informação, bem como a cooperação entre humanos e computadores.*

Pode-se concluír que a interoperabilidade semântica é um conceito resultante da necessidade de implementação da *Web* Semântica. Assim sendo, verificamos que o ato de conferir o tal contexto e significado à informação é denominado de interoperabilidade semântica [13] . E tal ato só é exequível através da utilização das seguintes ferramentas principais:

- *RDF*;
- *RDF/XML*, *JSON-LD*;
- *RDFS*;
- *OWL*;
- *SPARQL*;

Resource Description Framework

RDF (*Resource Description Framework*) originalmente desenvolvido para organização de metadata, tem vindo cada vez mais a ser utilizado como ferramenta de descrição ou modelação de informação. Aliás, a própria *W3C* já define o *RDF* como uma *framework* para expressar informação sobre *resources*, dados. Estas *resources* podem ser qualquer coisa, tanto podem caracterizar uma pessoa, como um documento, ou mesmo um conceito abstrato. [17]

Antes de se aprofundar o tema, é essencial que já se tenha uma perspetiva dos diferentes usos do *RDF*, bem como as suas vantagens:

- Possibilidade de leitura de informação em páginas *Web* e indução do seu significado, por parte de sistemas externos, através da inserção de vocabulário que auxilie na interpretação dessa mesma informação. É crucial que este vocabulário seja universal - *schema.org*;
- Enriquecimento de *datasets* existentes na *internet* através da facilidade com que se pode interligar e partilhar informação entre um *dataset* e entidades externas - documentos, outros *datasets*, *et cetera*.
- Dogma do *RDF* está assente no princípio de tudo estar conectado. As pesquisas tornam-se mais eficientes. A agregação de dados em tópicos torna-se uma realidade. Melhor formatação e organização da informação aquando da visualização.

Atualmente, o *RDF* está orientado à *internet* e é fundamental em cenários onde informação existente na *Web* precisa de ser processada por aplicações. Além disso, e cada vez mais, numa tentativa de normalizar os dados e aumentar a possibilidade de interoperobilidade semântica, o *RDF* tem vindo a ser utilizado aquando da publicação de informação na *internet*. Assim, lentamente vai-se construíndo uma estrutura em *RDF* da *internet*, criando continuadamente relações e interligando as várias entidades existentes, páginas *web*, documentos, pessoas, *resources*, *et cetera - Linked Data* [18].

É importante reter que a *Web* Semântica não se resume ao simples ato de disponibilizar informação com uma estrutura previamente acordada. É necessário inter-relacionar a informação, de modo a melhorar a contextualização da informação e por consequência a interpretação dos dados. Isto só é possível através da criação de links - pontes - entre as várias entidades, bloco de dados.[19]

O sucesso da inter-ligação entre entidades, resources, deve-se ao facto do *RDF* utilizar expressões como sujeito-predicado-objeto, denominadas de *triplets*, aquando da descrição dos respetivos dados. O sujeito representa um conjunto de dados e o predicado (denominado também de propriedade) expressa uma relação, associação, a um objeto. Em suma, o sujeito e o objeto consistem em duas entidades que estão relacionadas através de uma propriedade, predicado. [20]

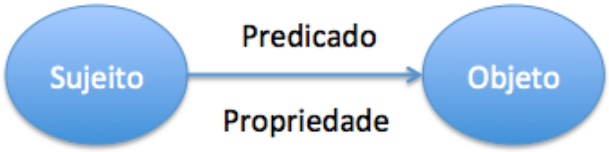


Figura 7: Definição de *RDF Triple*.

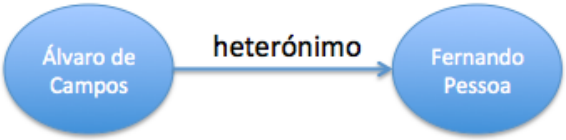


Figura 8: Exemplo de *RDF Triple*.

Como se pode ver através das figuras 7 e 8, o *RDF* está assente em grafos - um *triple*, e por si só, já se pode considerar um grafo. Uma modelação dos dados baseada em grafos, além de suportar heterogeneidade ao nível da informação, é a opção mais simples e fiél para representar a expressividade, as relações entre entidades em comparação a outro de modelos de base de dados existentes, tais como [21]:

- Modelos Físicos;
- Modelos Relacionais;
- Modelos Semânticos;
- Modelos Orientados aos objetos;
- Modelos de dados Semi-estruturados.

MODELO FÍSICO

Este tipo de base de dados encontra-se, atualmente, obsoleto. Embora eles permitam guardar quantidades enormes de informação, o seu nível de abstração é quase nulo e a estruturação de dados é inflexível, razões pelas quais, se torna impossível a modelação de grafos neste paradigma.

MODELO RELACIONAL

Modelo proposto por *Edgar Frank Codd* tinha como ponto principal a introdução do conceito de abstração - separação clara entre nível físico e lógico - onde esta última camada é responsável pela criação de relações entre conjunto de dados (*primary* e *foreign keys*). Ambas as perspetivas, Relacional e *RDF*, podem apresentar semelhanças aquando da criação de relações entre dados, porém, os restantes princípios do modelo relacional não se aplicam à filosofia da organização de dados baseada em grafos, por exemplo:

- A estrutura da base de dados têm que estar pré-definida;
- O *schema* do modelo relacional é fixo, o que dificulta em muito a integração de novos *schemas* -conteúdos homogéneos;
- *SQL* não permite representar caminhos(predicados em *RDF*), vizinhança. Além disto, o tipo de conetividade existente entre dados restringe-se à transititivade.

MODELO SEMÂNTICO

A criação e desenvolvimento deste modelo deve-se à necessidade de encontrar soluções para conferir uma maior expressividade na altura da descrição da informação organizada e as suas respetivas relações. Uma base de dados assente num modelo semântico já permite aos utilizadores representar objetos e as suas relações de uma maneira simples e natural através da utilização de conceitos abstratos, tais como: agregação, classificação, *sub* e *super* classes, heranças e hierarquias. Do ponto de vista do *RDF*, este tipo de modelo, tem algumas semelhanças já que ambas tentam ressalvar, representar e salientar as relações entre as várias entidades a serem modeladas.

MODELO ORIENTADO AO OBJETO

Estes modelos são baseados na programação orientada a objetos, onde a informação é representada através de colecções/grupos de objetos organizados por classes com as suas respetivas variáveis e métodos associados. Pode-se dizer que a organização da informação segue uma abordagem inspirada nos grafos, devido ao facto de que os objetos também se relacionam com outros objetos, o que leva a expressões do tipo objeto-método-objeto. No entanto, ainda se verifica diferenças muito significativas no âmbito da organização em comparação ao *RDF* e a sua abordagem em grafos. O *M.O.O* assume que o mundo à sua volta é constituído por objetos que interagem com outros através de métodos. Por sua vez, o *RDF* dá enfâse à intercomunicação da informação, às relações entre os dados e as propriedades dessas mesmas.

MODELO DE DADOS SEMI-ESTRUTURADO

De todos os modelos existentes na literatura, este é o que mais se assemelha aos principios do *RDF*. Os modelos semi-estruturados foram desenvolvidos para suportar blocos de informação onde a estrutura se encontra irregular, implícita e parcial. Além disso, o *schema* deste tipo de modelos está em constante evolução, o que leva a um aumento no tipo de informação que podem conter. Atualmente, o estandarte deste universo é o *XML*. Porém, ainda existem muitas diferenças significativas que impossibilitam um mapeamento direto entre informação expressa em *RDF* para o *XML*, tanto ao nível da abstração, onde o *RDF* se encontra numa camada superior, a nível da organização dos dados, o *XML* está alicerçado numa estrutura de árvore ordenada e por fim a nível da semântica, já que este último é *self-describing* - a informação sobre os dados encontra-se nos mesmos - enquanto que o *RDF* expressa explicitamente a informação dos dados através da criação de relações entre as várias entidades descritas.

MODEL	LEVEL	DATA COMPLEX.	CONNECTIVITY	TYPE of DATA
Network	physical	simple	high	homogeneous
Relational	logical	simple	low	homogeneous
Semantic	user	simple/medium	high	homogeneous
Object-O	logical/physical	complex	medium	heterogeneous
XML	logical	medium	medium	heterogeneous
RDF	logical	medium	high	heterogeneous

Figura 9: Sumário da comparação entre os vários modelos de base de dados.

Como já foi dito anteriormente, o *Resource Description Framework* é uma ferramenta conceptual focada na descrição da informação. Por si só, o *RDF* não consegue inferir o significado da informação que lhe é apresentada. É neste preciso momento que o *RDFS* entra em cena.

Resource Description Framework Schema

RDFS - Resource Description Framework Schema - é uma extensão do *RDF* que se foca em conferir significado à informação, através de mecanismos e vocabulários que permitem descrever tanto objetos como as suas relações. Por outras palavras, o *RDFS* é uma extensão semântica do RDF que permite definir caracteristicas semânticas dos dados *RDF*. [22]

De modo a conseguir interoperabilidade entre fontes de informação, sistemas e aplicações, o vocabulário usado tem que ser consistente. Em termos leigos, é necessário que os sistemas envolvidos na interoperabilidade criem e desenvolvam uma linguagem universal para o universo em que se encontram. Transpondo para o nosso quotidiano, um ser humano muito dificilmente consegue interagir e cooperar com outro pessoa se não houver elos de comunicação, concordância do significado dos conceitos expressos, *et cetera*. É esta mesma abordagem que é necessária implementar no mundo digital - a criação de um vocabulário reutilizável e universal que permite descrever sem qualquer tipo de ambiguidade e exprimir o que a informação representa. Além disso, uma maior reutilização de vocabulário, equivale a uma maior ligação de *resources*, o que leva a uma maior interoperabilidade, a uma maior contextualização da informação, *et cetera*. [22, 23, 24]

Alguns exemplos de grupos de vocabulários, ontologias:

- *Foaf*;
- *Dublin Core Metadata Initiative*;
- *Schema.org*;
- *SKOS*.

FOAF

Friend of a friend foi uma das primeiras ontologias adoptadas em todo o mundo para descrever pessoas, as suas atividades e relações que possam ter com outras pessoas ou objetos. [25]

```
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#JW>
    a foaf:Person ;
    foaf:name "Jimmy Wales" ;
    foaf:mbox <mailto:jwales@bomis.com> ;
    foaf:homepage <http://www.jimmywales.com/> ;
    foaf:nick "Jimbo" ;
    foaf:depiction <http://www.jimmywales.com/aus_img_small.jpg> ;
    foaf:interest <http://www.wikimedia.org> ;
    foaf:knows [
        a foaf:Person ;
        foaf:name "Angela Beesley"
    ] .

<http://www.wikimedia.org>
    rdfs:label "Wikimedia" .
```

Figura 10: Exemplo do uso de *Foaf* em formato *RDF/Turtle*.

DCMI

Dublin Core Metada Initiative - framework desenvolvida com o objetivo de criar e estabelecer normas que facilitem a implementação de interoperabilidade. Disponibiliza um leque geral de conceitos - vocabulários - que podem auxiliar na semântica dos dados. [26]

SCHEMA.ORG

Esta ontologia resulta de uma parceira de empresas como a *Google*, *Microsoft*, *Yahoo*, entre outras. Como já foi referido anteriormente a pesquisa está a mudar, cada vez mais os utilizadores normais querem uma pesquisa mais eficiente e que de uma certa forma tente deduzir o que eles querem agora e o que eles vão querer. Por exemplo: O António está curioso com o último filme da trilogia do *Hobbit* e, ao escrever apenas o título do filme no motor de busca, conseguirá obter várias informações acerca do mesmo, entre as quais:

- Conteúdo do filme;
- Salas de Cinema;
- Horários;
- *et cetera*.

Em suma, este vocabulário, ontologia, permite adicionar às páginas *web* meta-dados sobre o significado das mesmas, facilitando assim a procura e agregação de temas e entidades. [27, 28]

SKOS

Desde 2009 que o *Simple Knowledge Organization System* é uma recomendação do *W3C*. Este foi desenvolvido para facilitar a publicação e representação de *thesauri*, taxonomias, *classification schemes*, *subject-heading system* entre outros. Atualmente, a sua utilização restringe-se à área bibliográfica. [29]

Estes exemplos só representam uma pequena percentagem de um mundo que está em constante evolução e expansão. Todos os dias surgem novas ontologias e novos vocabulários com novas especifidades, moldados a uma restrita área de discurso.

Serialização de RDF

O *RDF* e *RDFS* são ferramentas abstratas, ou seja, normas a seguir aquando da organização e caracterização de informação de modo a facilitar a interoperabilidade semântica. Por si só, não são suficientes para transmitir qualquer tipo de informação entre os diversos sistemas. É necessário uma sintaxe concreta que suporte este nível de abstração bem como os seus principios. [28]

A solução para este problema reside na serialização do *RDF* e *RDFS* em formatos que possibilitem essa transposição, neste caso, Modelos de dados Semi-Estruturados. Estes, como já foi verificado anteriormente, são os que mais se assemelham à filosofia destas ferramentas e, além disso, são, em comparação aos restantes modelos, os mais fáceis de manipular e moldar ao universo do Resource Framework Description.

Antes de mais, a serialização consiste no processo de guardar entidades *resources* em modelos de armazenamento. No âmbito do *RDF*, a serialização consiste em representar a estrutura abstrata criada seguindo os princípios do *RDF* num formato concreto e real que pode ser lido pelas máquinas ou seres humanos . Por outras palavras, extensões do *RDF* capazes de descrever e suportar grafos através de uma sintáxe.

Como se pode observar na literatura, nos dias de hoje existem várias extensões do *RDF* capazes de servir este propósito. Aqui, ao contrário do paradigma das ontologias, uma variade de formatos e sintaxes não significam necessáriamente ambiguidade ou imprecisão no momento de interpretação da informação, já que no fim de contas, os dados, a semântica é imutável; só a sintática é que se altera. Só a organização da informação é que muda, o resto permanece constante.

Exemplos de Serializações de RDF:

- *N-Triples*, *Turtle* e *N-Quads*;
- *JSON-LD*;
- *RDFa*;
- *RDF/XML*.

Desta listagem, só é relevante salientar dois tipos de serialização - *JSON-LD* e *RDF/XML*. Estes são, de momento, os mais escolhidos para descrever as entidades de um grafo e as suas relações, todavia as razões da escolha diferem. A popularidade do *RDF/XML* deve-se ao facto do *XML* ter sido a única sintaxe disponível aquando da criação do *RDF*. Ultimamente tem sido substituída por uma vertente mais simples e recente - *JSON-LD*. Esta abordagem em *JSON* permite uma representação menos verbosa e mais personalizada de um grafo. [17, 30, 31, 32]

SPARQL

Tudo o que foi referido anteriormente consiste na modelação e armazenamento dos dados de modo a possibilitar a interoperabilidade semântica. Contudo isto não é suficiente para inferir ilações sobre os dados. Para atingir este patamar, é necessário a utilização de ferramentas e linguagens próprias que possibilitam a criação de queries segundo a abordagem *RDF*. Neste universo, já existem vários tipos de linguagens e que permitem a realização deste tipo de queries, tais como:

- *DQL*;
- *Versa*;
- *RDFQ*;
- entre outras.

Porém, é sobre o *SPARQL* - **S***PARQL* **P***rotocol* **a***nd* **R***DF* **Q***uery* **Language** - que recai a maioria das queries semânticas, chegando ao ponto de *Sir Tim Berners-Lee* qualificar esta ferramenta como um dos pontos fulcrais para o sucesso da *Semantic Web* - '*SPARQL will make a huge difference*'. [18, 33]

A sua popularidade deve-se ao facto de ter sido desenvolvida pelo *W3C*, bem como pelo seu poder de abstração, isto é, o *SPARQL* adapta-se a qualquer fonte de informação, quer esta seja *Turtle* ou *JSON-LD*. Além disso, o *SPARQL* proporciona ao utilizador um conjunto vasto de queries e operações, tais como *SELECT, CONSTRUCT, ASK e DESCRIBE* em que:

- *SELECT* - extrai *raw values* e retorna os resultados no formato desejado: *JSON*, *XML*, *et cetera*;
- *CONSTRUCT* - extrai informação e recria uma serialização de *RDF* com o resultado obtido;
- *ASK* - utilização restringe a queries cujo resultado seja verdadeiro ou falso;
- *DESCRIBE* - retorna um único grafo *RDF* que vai de encontro às restrições usadas.

2.2 SISTEMAS DE INFORMAÇÃO

Tecnologias de Informação e Sistemas de Informação são dois conceitos codependentes. A primeira, de uma perspetiva aplicacional e pedagógica, compreende todas as atividades e estudo de soluções relativas à informação, no caso da procura, processamento e manipulação de informação. A segunda é o resultado do estudo das soluções. Um Sistema de Informação é uma estrutura que possibilita essas atividades. Um sistema de

- armazenamento;
- organização;
- processamento;
- disseminação

de informação. Já há muito tempo que o sucesso de uma empresa deixou de ser ditado pelo os produtos que oferece e pelos preços que pratica. Hoje em dia, a oferta é tanta, que um consumidor para um mesmo produto tem uma vasta panóplia de opções à sua disponibilização. Tal saturação de oferta e de competividade, estimularam uma alteração de paradigmas. Agora, as principais ferramentas de diferenciação do sucesso entre empresas passam pelo:

- armazenamento;
- organização;
- conhecimento;

e, isto, só é possível com sistemas que possibilitam armazenar e processar grandes quantidades de informação devidamente organizadas. [34] Com o papel isto não é possível. Em projetos de pequena escala talvez, porém, no presente atual, tal coisa já não acontece com frequência. Pensa-se global. Pensa-se em grande escala. Cada vez mais, as empresas começam a cortar no uso do papel, de capas, *post-its*, para guardar e organizar informação persistente e crucial, substituindo-as por base de dados, folhas de cálculos, entre outras ferramentas de armazenamento persistente. Porém, o sistema de informação não se resume a guardar. Os *SI* têm como principal objetivo o suporte à tomada de decisão, bem como facilitar o controlo e organização de uma empresa, de uma entidade, de informação. O armazenamento por sí só não chega. Um *SI* inclui todos os protocolos/regras de obtenção de informação, de tratamento de informação, de disseminação de informação e de extração de conhecimento. [35]

Esta figura demonstra o raio de acção normal de um sistema de informação no mundo industrial. Facilmente se consegue perceber que uma aposta sólida num programa de Tecnologia de Informação e num Sistema de Informação robusto acaba por cobrir todo o fluxo de trabalho existente, toda a logística e ajudar em todos os nívels de operações empresarias - desde uma gravação de uma simples transação até à utilização do conhecimento existente para ajudar na tomada de decisão dos diretores executivos.

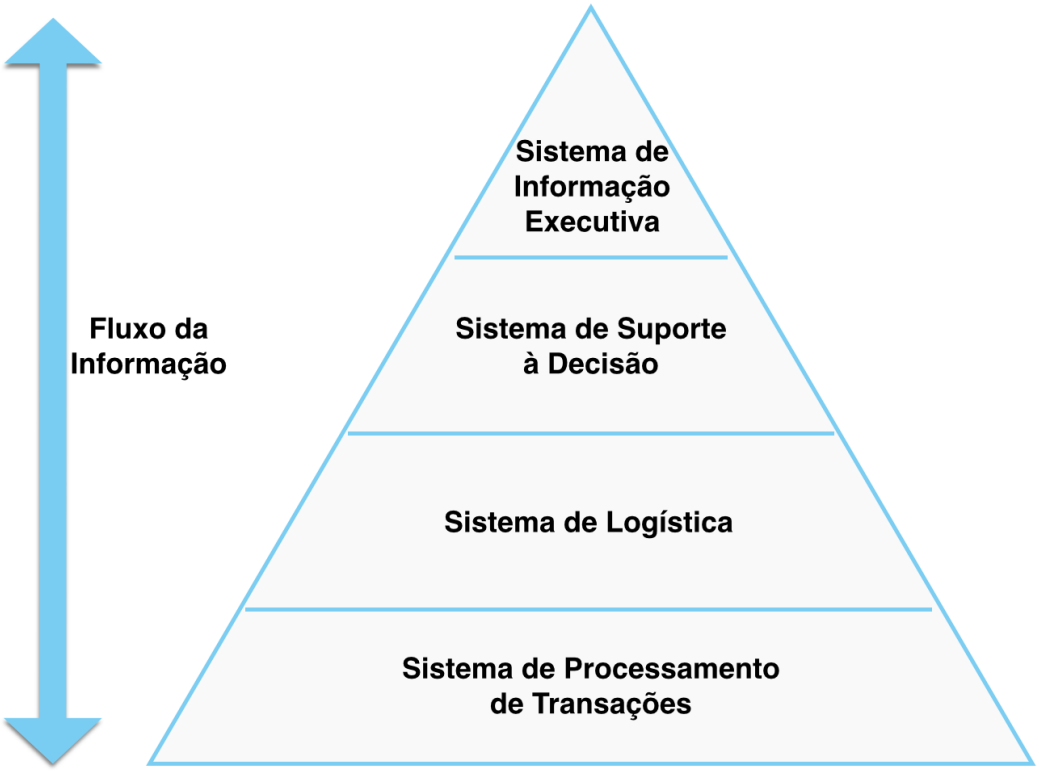


Figura 11: Overview do fluxo de informação nos diversos níveis de um Sistema de Informação.

2.2.1 *Sistemas de Processamento de Transações*

A população de base de dados, o armazenamento de informação, o ponto de entrada de informação no Sistema de Informação está assente no primeiro piso da pirâmide. Tudo que seja transação, registo industrial ou comercial, passível de ser útil, e necessário para o futuro, acaba por ser guardada em estruturas persistentes, tais como, base de dados, folhas de cálculo, *XML*, entre outros. É importante não esquecer que atualmente devem ser feitas milhões de transações por dia, muitas em simultâneo, sendo então necessário realizar um grande investimento num sistema de processamento de transacões para conseguir agregar quantidades enormes de dados e tratá-los de acordo com o contexto. Os requisitos de um *SPT* cingem-se a:

- Performance;
- Disponibilidade;
- Integridade dos Dados;
- Facilidade;
- Modular.

Ao nível da Performance, o contexto força o aumento de rapidez de resposta. O *software* tem de conseguir suportar e processar quantidades gigantes de informação o mais rápido possível, de forma a não atrasar o fluxo de trabalho. Além do curto tempo de resposta,a prontidão do sistema é fulcral. O *SPT* nunca pode estar indisponível, *offline*. O simples ato de não poder registar novas transações pode levar a interrupções do fluxo de trabalho, ou até mesmo à paragem da empresa. Não obstante ao fato de originar incongruências de dados num nível geral. Ou seja, a integridade dos dados é também posta em causa. O estudo da usabilidade do sistema de processamento de transacções não deve ser esquecido. É importante que os utilizadores do sistema consigam utilizar o *software* de registo sem dificuldades. Isto leva a uma diminuição do erro humano aquando do registo de transacções. Melhorando assim a integridade dos dados. Por fim, o modular resume-se à possibilidade de permitir que o sistema esteja susceptível a mudanças. Adicionar, remover ou alterar *software* dependendo das mudanças do contexto empresarial ou industrial. Assim, sempre que é necessário fazer qualquer tipo de alteração, não se precisa de criar um novo sistema de processamento de transações, poupando tempo e dinheiro.

2.2.2 *Sistemas de Logística*

O sistema de logística, como o próprio nome indica, está encarregue da logística de uma empresa. Este sistema serve de suporte para:

- Controlo de *stocks*;
- Controlo de equipamentos;
- Tratamento de informação.

Graças ao registo de todo o tipo de transacções do fluxo de trabalho de uma empresa, o responsável pela logística consegue facilmente verificar se é necessário comprar matérias primas, qual a quantidade existente de materiais no armazém da empresa, se há produtos em falta para dar início a planos de produção, *et cetera*. Também, e desde que haja interoperabilidade entre os equipamentos e o *SPT*, o logista pode averiguar se existe alguma máquina que não esteja a funcionar a 100% dando início a reparação do equipamento. Finalmente, o encarregado é responsável, em certa parte, por tratar mais uma vez a informação e os dados que têm, compilando-os e registando essas conclusões a fim de aprimorar a qualidade e integridade dos dados no Sistema de Informação.

2.2.3 *Sistemas de Suporte à Decisão*

À medida que vamos subindo na pirâmide, vamos, também, subindo na escala hierárquica da empresa. Este Sistema está restringido aos responsáveis primários pela gestão da empresa, pela produção. Através de extração de conhecimento e de outros técnicas de inteligência artificial, é possível inferir padrões e conclusões em grandes quantidades de informação ao nível da eficiência de planos de

produção; comparação de trimestres no âmbito da produção e de vendas; bem como a previsão de vendas tendo em conta o contexto em redor. O sistema de suporte à decisão tende mais a ser usado como uma ferramenta de ajuda numa perspetiva mais comercial e de negócios.

2.2.4 *Sistemas de Informação Executiva*

Geralmente este é considerado uma especificação do sistema de suporte à decisão. Não há alterações profundas ao nível dos objetivos em relação aos *SSD*. Simplesmente, este foca-se mais no panorâma global da empresa, nas diretrizes, no modo como a instituição funciona como um todo. Um sistema de informação executiva só é utilizado pela direção. Este permite fácil acesso a informação externa à instituição, bem como uma visualização compilada e organizada do aspeto geral da empresa, tanto a nível de produção, economia e de *marketing*.

Claro que estes níveis variam de empresa para empresa, de negócio para negócio, porém, todos seguem o mesmos princípios aqui descritos - divisão do sistema de informação por camadas tendo em conta o raio de acção e o contexto em que serão usadas. Sendo que a camada superior recebe ou utiliza informação proveniente dos níveis inferiores e, para que esta troca de dados aconteça, é fundamental que haja interoperabilidade a todos os níveis. A possibilidade de receber e tratar informação entre os vários sistemas existentes.

Assim sendo, pode-se verificar que uma aposta forte no desenvolvimento de *SI* não pode descurar a interoperabilidade. Uma maior interoperabilidade traduz-se num sistema de informação mais robusto, mais eficaz, mais forte. Atualmente, as Tecnologias de Informação e os Sistemas de Informação são consideradas, na literatura, como uma arma de combate das empresas, da indústria, no século XXI.

3

UNIVERSO BRAGA-BOSCH

Este capítulo corresponde ao estudo da arquitetura informática presente na Unidade da *Bosch* em Braga. Como foi referido anteriomente, este levantamento e mapeamento de informação é necessário para o sucesso do desenvolvimento e posterior implementação do *software* de gestão de produção - *SMART Manufacturing Control*. Recapitulando, o *SMART Manufacturing Control*, é um sistema inteligente que possibilita uma gestão genérica de artigos, desde o planeamento e controlo de produção até à sua programação na Linha de Inserção Automática da Unidade da *Bosch* de Braga. A gestão genérica implica que o software esteja pronto a responder e a processar qualquer tipo de pedido por parte do utilizador. O *SMART* tem que ser capaz de representar qualquer tipo de matéria prima, recurso ou informação necessária para o planeamento e desenvolvimento de um produto. Resumidamente, pode-se dizer que este sistema vai servir de suporte à produção de inserção automática de *SMD*. Ou seja, o estudo do sistema informático da *Bosch* concentrar-se-á, essencialmente, nos departamentos responsáveis pela produção de inserção automática orientada aos *SMD*, na análise das fontes e sistemas de informação necessárias para a representação do fluxo de trabalho existente, bem como, uma verificação das práticas de interoperabilidade utilizadas.

3.1 DOMÍNIO DA PRODUÇÃO DE INSERÇÃO AUTOMÁTICA DE SMD

Antes de se começar a discriminar as várias fontes de informação e os respetivos protocolos de interoperabilidade, é necessário que se tenha uma perspetiva mais geral do fluxo de trabalho na produção de inserção automática. Quais são os departamentos responsáveis e como é que eles se relacionam entre si?

Atualmente a produção de *SMD* engloba os departamentos de logística, planeamento de produção, o Supermercado e dois departamentos de produção. O fluxo de informação, de trabalho, pode ser descrito assim:

O Departamento de Logística, *LOG1*, é responsável pelo planeamento de toda produção de acordo com as encomendas que tem de momento. Esse planemanto é entregue ao *MOE14* - departamento principal pelo planeamento e produção. Por sua vez, *MOE14*, utiliza a nova informação obtida pelo *LOG1* e dá início à elaboração do plano final de produção que é entregue no *MOE11*. Este está encarregue pela produção dos *SMT* que serão vendidos a clientes ou utilizados pela própria fábrica na

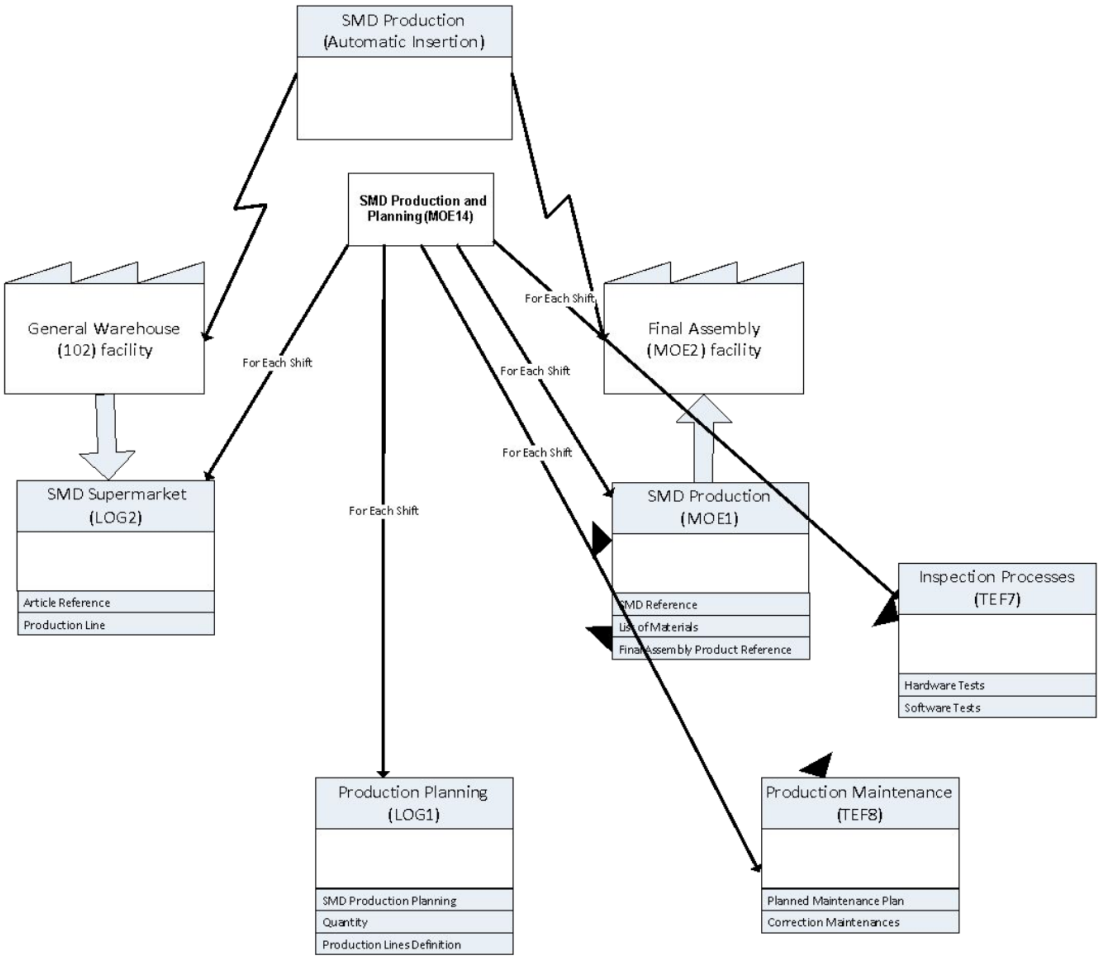


Figura 12: Overview do fluxo de trabalho na produção de inserção automática de *SMD*.

criação de componentes específicos. Por outro lado, o resto dos departamentos servem de suporte ao controlo de qualidade de produção (*MOE12* e *MOE18*,*TEF7* e *TF8*) e ao abastecimento de matérias primas necessárias - *LOG2*. Mais facilmente se consegue compreender o fluxo de trabalho através da figura que se segue:

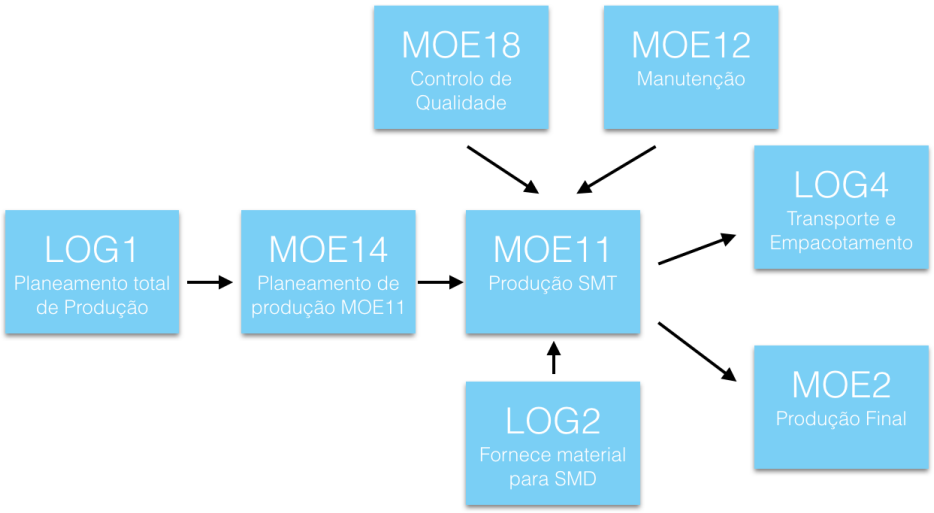


Figura 13: Resumo do fluxo de trabalho.

3.1.1 *Departamento de Logística*

A Logística não se limita a garantir a manutenção dos stocks, das matérias primas, a verificar se estas estão no sítio certo à hora certa, se há ruptura de stock, *et cetera*. O departamento de logística e os seus respetivos subdepartamento são responsáveis pela recepção e expedição de material, pelo fornecimento de produtos, pela organização de matérias primas necessárias para montagem afinal. De modo a responder eficazmente a todos os pedidos, a Unidade de Braga, o departamento de Logística, está dividido em:

- *LOG1* - Controlo de encomendas e o planeamento de produção;
- *LOG2* - Gestão do fluxo de trabalho e logísticas internas;
- *LOG3* - Fornecimento;
- *LOG4* - Transporte;
- *LOG-P* - Outras logísticas relacionadas com a gestão;

Todavia, os vários subdepartamentos acabam por utilizar o mesmo sistema informático a fim de causar o menos entropia possível aquando da cooperação. Os subdepartamentos estão interligados e trabalho em conjunto, compilando a informação necessária para mais tarde entregar ao *MOE11 e MOE14*, como se pode observar nas figuras 12 e 13.

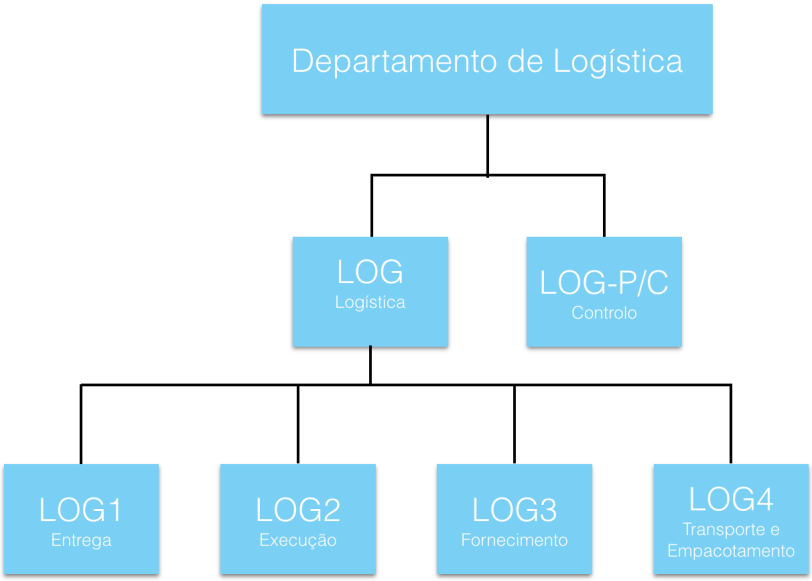


Figura 14: Organização do departamento de Logística.

Ao nível dos Sistemas de Informação, pode-se dizer que a ponte de ligação entre todos os subdepartamentos de logística é a junção dos sistemas *SOL*, *SAP* e *CMTracer*. Este são responsáveis por responder a:

- Pedidos de fornecimento de matérias primas resultantes da informação proveniente do LOG4 - Transporte;
- Recepção e expedição de materiais;
- Inventário do *stock* existente nos armazéns;
- bem como todas as responsabilidades circunscritas pelos diversos subdepartamentos.

SOL

O *SOL* é constituído por uma aplicação reservada ao *back office* das fontes de informação - adicionar, alterar e remover informação - e por uma aplicação *front end* de consulta e acessível por computador ou *PDA*. O *SOL* está encarregue de controlar os pedidos de fornecimentos de materiais através do *milk run* - o transporte.

Este sistema está assente numa base de dados *MySQL*. O resto da topologia encontra-se descrita na figura que se segue - figura 15.

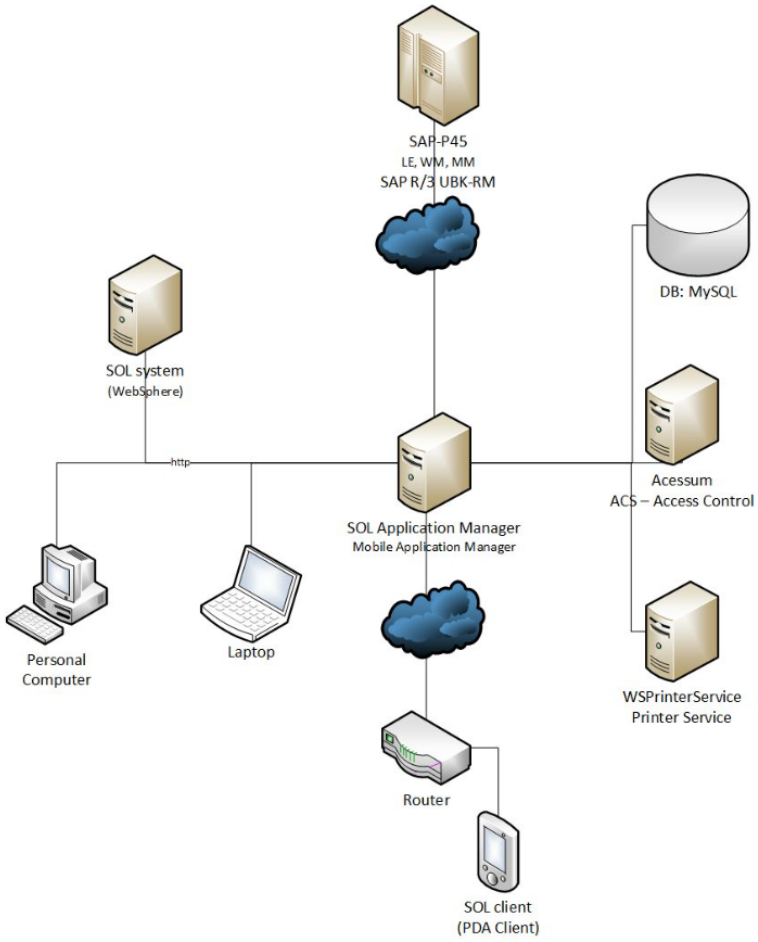


Figura 15: Arquitetura do sistema *SOL*.

SAP R/3 ERP System

SAP R/3 ERP System é um sistema baseado em *ERP*. Sucintamente um sistema *ERP - Enterprise Resource Planning* - permite armazenar, processar, interpretar e visualizar transações industriais ou comerciais realizadas em tempo real. Transações que correspondem, por exemplo, a custos de produção, entrega de produtos ou serviços, controlo dos armazéns, encomendas e pagamentos, entre outros. Na Unidade de Logística de Braga, este *software* acaba por ter a seu encargo:

- gestão de armazéns;
- controlo de matérias primas;
- execução logística;
- recepção de material;

CMTracer

Por sua vez, o *CMTracer* é uma aplicação de suporte às linhas de produção. Resume-se à localização dos produtos e matérias primas, ou seja, vai guardando os percursos das várias matérias ao longo da produção e a sua localização atual. Este sistema, além de estar assente num motor *Oracle* de base de dados, também interage com outros *softwares* e fontes de informação. Porém, essa interoperabilidade é manual e repleta de restrições.

Atualmente, o *CMTracer* comunica com:

- *SIIA*, *PIA*, *SOL* - através de *web services*;
- *SIPIA*, *Siplace Pro DB*, *BPW1* - base de dados;
- *Pana Pro* - através de *XML* próprios para um módulo específico do *CMTracer* - *CMTracer Pana Pro*
- *Viscom*, *Koh Young*, *WSPrinter* - ficheiros *XML*.
- *RLaser*, *Rehm*, *et cetera*.

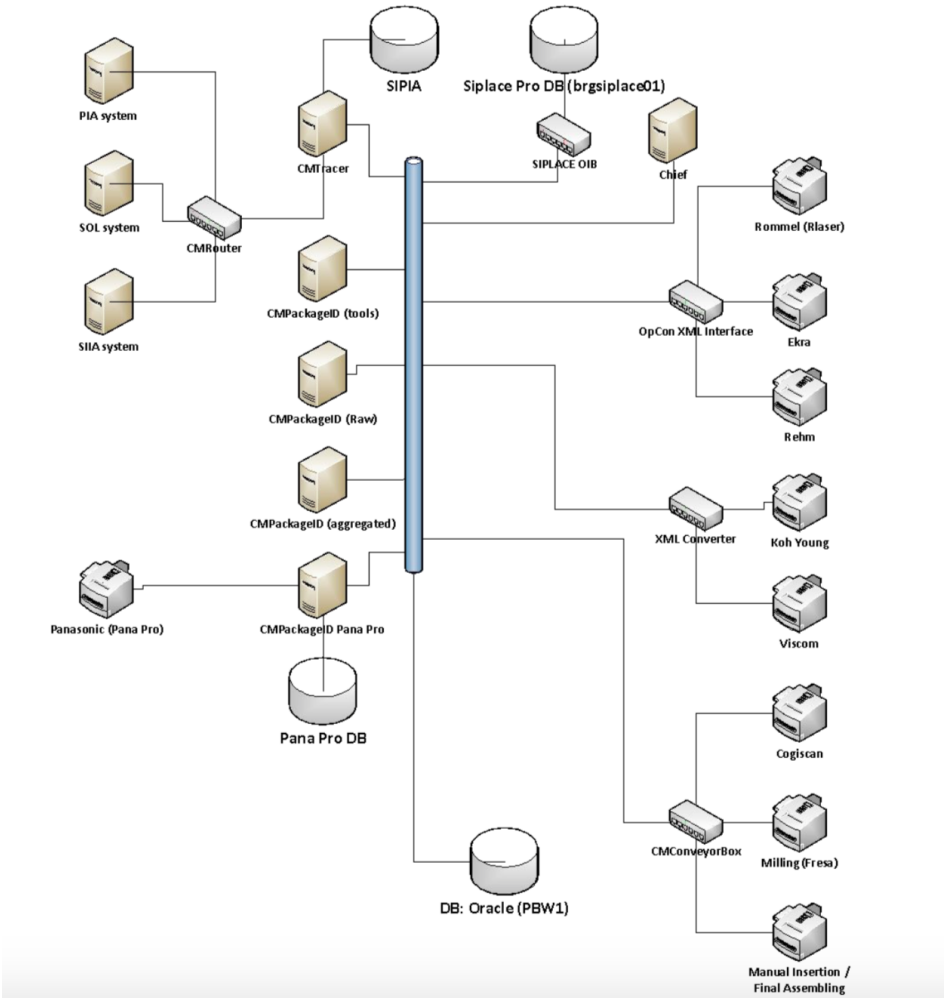


Figura 16: Arquitetura do sistema *CMTracer*.

3.1.2 *Departamento de Produção*

A área de Produção em Braga está dividida, tal como a de Logística, em subdepartamentos: *MOE1*, *MOE2*, *MOE-Q*. Porém, o único que está dentro no raio de acção do produto de *software* - *SMART Manufacturing Control* - é o *MOE1*. Este é responsável pela produção dos *SMT* e, por sua vez, o *MOE1* está dividido em mais 5 secções:

- *MOE11* - produção *SMT*;
- *MOE12* - controlo de qualidade;
- *MOE14* - planeamento e programação de produção;
- *MOE18* - equipa de manutenção, prevenção e correcção.

- *MOE1-P* - outros projetos.

Ao nível do sistema informático que suporta estas equipas e secções, pode-se verificar que estes têm à disposição uma panóplia de aplicações de *software*, fontes e sistemas de informação diferentes e, aparentemente, sem grandes desenvolvimentos ao nível de protocolos de interoperabilidade.

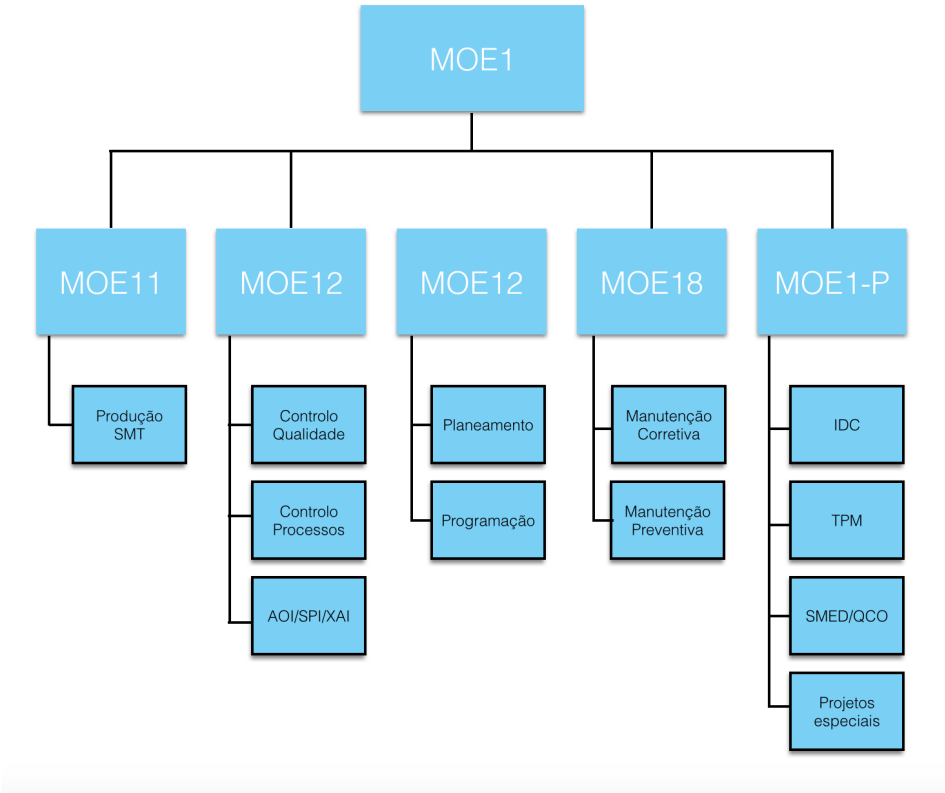


Figura 17: Organização do departamento *MOE1*.

SAP R/3 ERP System

Ao contrário do que se verifica no Departamento de Logística, a utilização deste sistema difere no caso de produção. Aqui, o *SAP* é responsável pela:

- preparação e validação das várias fases de produção;
- preparação da pasta de soldar (quando uma pasta é transportada do armazém para o outro, a transação é registada no sistema *SAP*);
- inserção automática de componentes;
- planeamento de produção.

PIA

O sistema de Preparação de Inserção Automática serve de suporte à fase de preparação. Retorna a informação necessária para a validação e preparação das diversas fases de produção. Por um lado, este sistema está assente em dois motores de base de dados - *MySQL* e *Oracle* - por outro lado, o *sistema PI.* também troca informação com várias aplicações existentes, tais como:

- *SIPIA*;
- *SAP P45*;
- *Acessum (Access Control)*;
- *WSCMTracer*;
- *WSPrintServer*;

Do ponto de vista aplicacional e estrutural, o PIA assemelha-se ao SOL. É constituído por uma aplicação de *back office*, uma aplicação *web* instalada num servidor *WebSphere* e, por fim, uma aplicação de *front end* para o *PDA*.

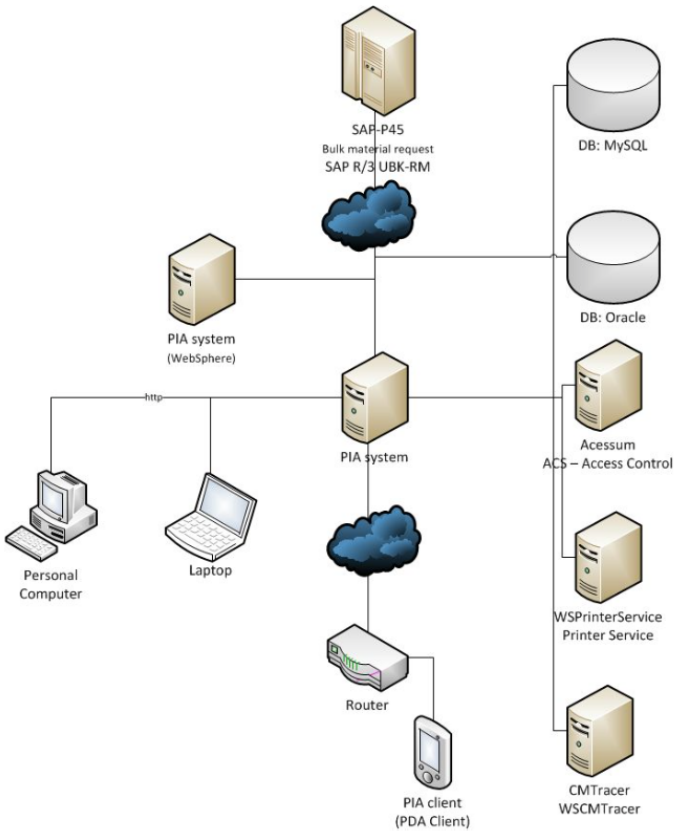


Figura 18: Arquitetura do sistema *PIA*.

SIIA

SIIA - Sistema de Informação de Inserção Automática - é responsável pelo registo e compilação todas as transações criadas ao longo das linhas de produção de inserção automática. Ao nível da arquitetura, o *SIIA* não foge à regra da maioria dos sistemas. Este é, inicialmente, composto por:

- aplicação *web* de gestão de *back office*;
- aplicação *web* de *front end*;
- aplicação móvel de *front end*;
- base de dados *Oracle*;
- diversas interfaces pré-existentes;

Em relação à primeira, facilmente se consegue inferir o objetivo da mesma. As restantes aplicações compreendem a ponte entre o utilizador e o sistema de informação de inserção automática. A única diferença significativa reside no acesso de cada uma. A ligação à aplicação *web*, instalada no *WebSphere*, só é conseguida através da rede interna da Unidade do Departamento de Produção. Por fim, a aplicação móvel foi desenvolvida no âmbito dos dispositivos *PDA*.

No caso das interfaces utilizadas, estas resumem-se a um conjunto de sistemas de informação já descritos ou ainda por descrever, tais como:

- *SIPIA*, um cliente para obter os programas de produção de equipamentos *Siemens*;
- *Acessum*, sistema de autenticação;
- *ProdMonitor*, indicadores de produção;
- *Asterisk*, estrutura de suporte às chamadas, assemelha-se a um sistema *VoIP*;
- *SAP*, registo das quantidades produzidas;
- *CMTracer*, visualização de transações efetuadas e do panorama geral de produção.

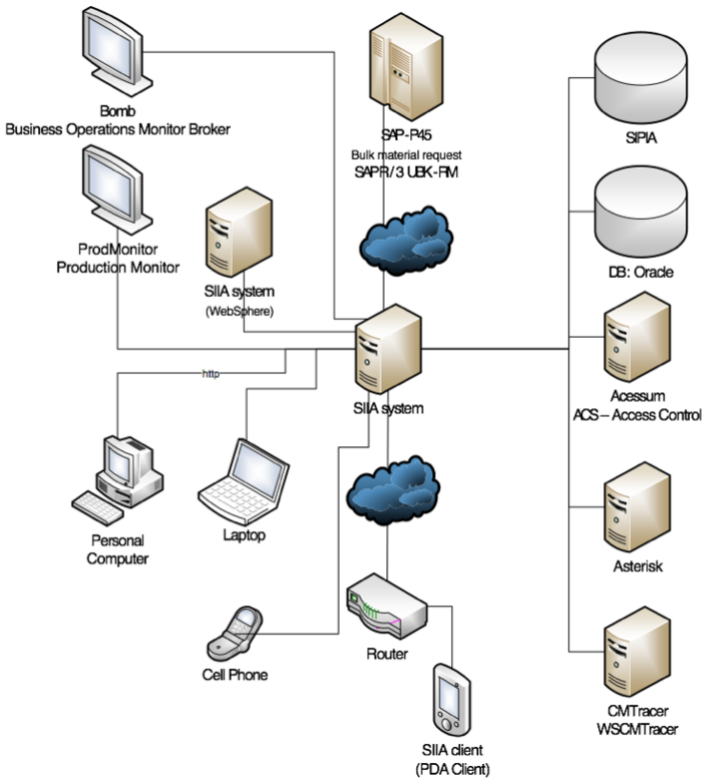


Figura 19: Arquitetura do sistema *SIIA*.

Viscom System

Este não tem grande importância no cômputo desta dissertação. Não está inserido no espetro de utilizador do *SMART Manufacturing Control*. Porém, a descrição e análise deste *SI* não pode ser descurada já que o estudo da arquitetura do sistema computacional da Unidade da *Bosch* engloba todos os sistemas de informação.

O *Viscom System* é um *software* de processamento de imagem industrial. A sua finalidade visa o controlo de qualidade do produto. Isto é feito através da realização em série de exames ópticos, raios-x em *2D* e *3D* ao nível da pasta, da soldadura e da disposição de placas.

ProdMonitor

O *ProdMonitor* é um sistema baseado em *Andon* que permite a visualização e monitorização em tempo real de todas as linhas de produção de inserção automática existentes no departamento *MOE1*. A monitorização integra:

- linhas de produção;
- monitorização e contagem de *PCBs*;

- monitorização e contagem de produção defeituosa;
- comparação entre eficiência desejada e a real;
- atraso do tempo de produção;

Além disto, o *ProdMonitor* também possibilita a análise dos dados obtidos através da criação de gráficos e diagramas.

A estrutura deste sistema está assente em duas temáticas complementares - núcleo e o cliente. O núcleo é responsável pela implementação das regras e protocolos de produção, ou seja, o modo como o Sistema de Informação processa e obtêm a informação proveniente das várias linhas de produção. Por sua vez, o cliente é uma interface aplicacional. Resume-se à mostragem dos dados processados. Ao nível da base de dados, o *ProdMonitor* está assente num motor *SQL Server* e *mySQL*.

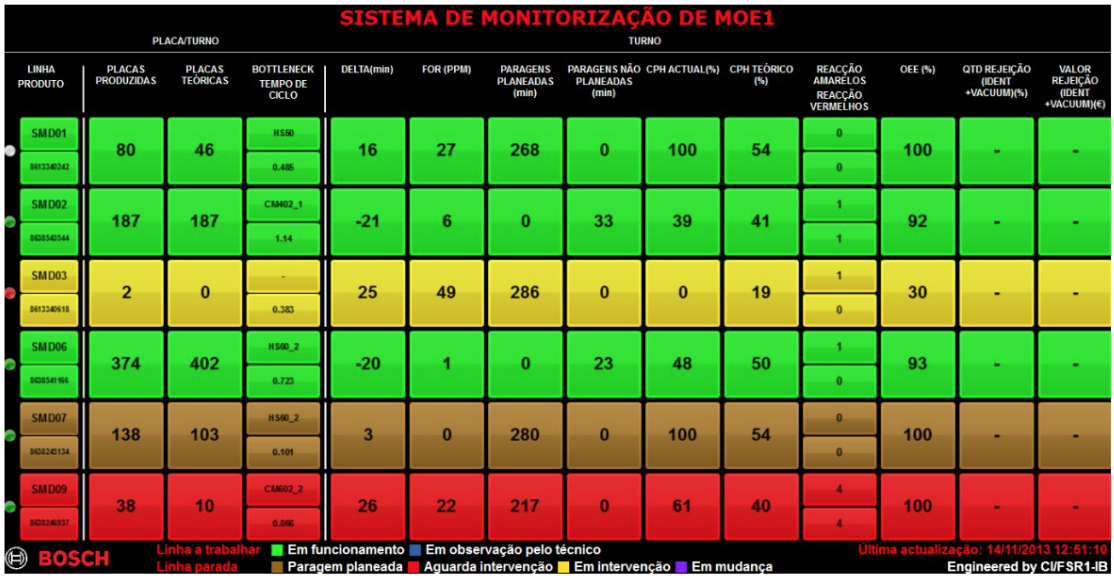


Figura 20: Exemplo do sistema *ProdMonitor* em acção.

Rommel RLaser

Rommel é o indicativo da empresa que produziu este sistema. *RLaser* remete-nos para um software que está inserido em máquinas de *laser typing* a fim de registar todo o tipo transações deste nível.

A separação e posterior definição dos nomes deste sistema não foram feitas ao acaso. Anteriormente, no capítulo de Introdução, foi referido que a maioria dos problemas de interoperabilidade advém das próprias entidades de desenvolvimento de software; e este caso não foge à regra. Atualmente, para transferir a informação originada das transações do *RLaser* para o *CMTracer*, é necessário utilizar mais dois produtos de software para conseguir essa interoperabilidade:

- *OpCon XML Interface*;
- *CMConveyorBox*;

Antes de se disponibilzar a informação no *CMTracer*, precisa-se de convertar a mesma para *XML* através do *OpConXML Interface* e, assim, após a conversão, já se pode aceder à mesma informação através do *CMConveyorBox*.

Ekra, Chief, Koh Young, Rehm

Este conjunto de sistemas encontra-se na mesma situação que o sistema de informação *Rommel RLaser*. Cada um foi desenvolvido por uma empresa diferente mas com o mesmo objetivo - registo de transações.

- *Ekra* - registo de transações em máquinas de inserção de pastas de soldar;
- *Chief* - visualização do plano de produção
- *Koh Young* - registo de transações em máquinas de inspeção de pastas de soldar;
- *Rehm* - registo de transações em máquinas de forno.

Ao nível da interoperabilidade e do transporte dos dados para o módulo principal de registo de transações, pode-se verificar que estes também não diferem do sistema *RLaser*. O transporte e atualização de dados no CMTracer só é possível com a conversão dos dados para *XML* através do *OpCon XML Interface*.

Siplace Pro System

Como já se viu anteriormente, a maior parte dos componentes da linha de inserção automática têm os seus próprios fabricantes e este também não foge à regra. O sistema *Siplace Pro* corresponde ao software embutido nas máquinas *Siemens* de inserção automática de componentes. Estas máquinas são responsáveis pela junção, montagem, de componentes em placas *PCB*.

A montagem é realizada através dum braço robótico e de acordo com a informação existente, ou seja, a 'receita de construção'. Assim sendo, pode-se assumir que este Sistema é de extrema importância no sucesso da linha de montagem e, além disso, é também, um dos mais problemáticos ao nível da partilha de informação e interoperabilidade, visto tratar-se de um sistema isolado e complexo em que a única ponte de ligação da base de dados com o exterior resume-se a um módulo chamado *Siplace Operations Information Broker*. Não se pode interagir e manipular diretamente a própria base de dados a fim de extrair conhecimento. O próprio módulo de comunicação, *Siplace OIB*, só retorna a lista de componentes presentes na máquina aquando do pedido de informação. Esta lista acaba por ser processada e guardada pelo *CMTracer*. Embora que inacessível, não se pode deixar de referir a configuração da base de dados. Neste caso, o motor da fonte de informação é *SQL Server*.

Pana pro System

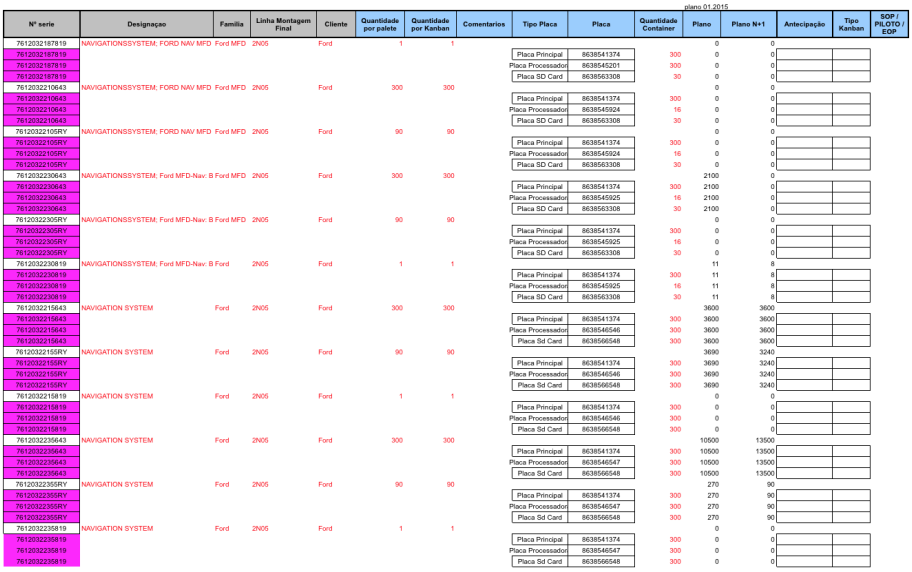
Pana pro System, fabricado pela *Panasonic*, serve o mesmo propósito do *Siplace* - inserção automática de componentes através de um braço robótico que encaixa os componentes no sítio correto de um placa, com ajuda de uma receita, um plano de montagem. No entanto, a problemática da extração e processamento da informação resultante deste sistema mantêm-se, embora por razões diferentes. No espectro do Siplace, o problema residia no facto da base de dados estar isolada do exterior a não ser pela utilização de um módulo, OIB. Aqui, não há qualquer problema de conexão com a base de dados, isto é, se a consideramos uma base de dados. Na realidade não existe uma base de dados convencional. A informação está guardada em ficheiros de texto, com uma estrutura própria, com delimitadores próprios e com semelhanças às *tables* de base de dados relacionais. Temos na mesma o conceito de *table*, de *column*, de linha, *et cetera*. Porém, está tudo representado em texto, em ficheiros *.pcb* e *.mix*.

Atualmente, numa fase de produção real, não há um único programa que faculte a consulta, a transmissão de informação importante. Ou seja, a transmissão de informação recai sobre o responsável pela linha de inserção automática em questão. Manualmente, o responsável processa e retira as próprias ilações propícias a erros humanos. Este Sistema não peca só pela falta de interoperabilidade, como também pela predisposição para o erro humano e posterior atraso de processos consequentes da extração de conhecimento por parte do funcionário. Por fim, os ficheiros resultantes são guardados, na íntegra, no sistema de informação *CMTracer*.

Tables de Excel

Na conjetura atual, as folhas de cálculo também podem ser consideradas Sistemas de Informação. O planeamento inicial de produção semanal é definido e atualizado em folhas de cálculo. Aqui o processo é completamente manual e desprovido de qualquer tipo de interoperabilidade.

3.1. Domínio da produção de inserção automática de SMD



plano 01.2015

Nº serie	Designaçao	Familia	Linha Montagem Final	Cliente	Quantidade por palete	Quantidade por Kanban	Comentarios	Tipo Placa	Placa	Quantidade Container	Plano	Plano N+1	Antecipação	Tipo Kanban	SOP / PILOTO / EOP
7612032187819	NAVIGATIONSSYSTEM; FORD NAV MFD	Ford MFD	2N05	Ford	1	1					0	0			
7612032187819								Placa Principal	8638541374	300	0	0			
7612032187819								Placa Processador	8638545201	300	0	0			
7612032187819								Placa SD Card	8638563308	30	0	0			
7612032210643	NAVIGATIONSSYSTEM; FORD NAV MFD	Ford MFD	2N05	Ford	300	300					0	0			
7612032210643								Placa Principal	8638541374	300	0	0			
7612032210643								Placa Processador	8638545924	16	0	0			
7612032210643								Placa SD Card	8638563308	30	0	0			
7612032210RY	NAVIGATIONSSYSTEM; FORD NAV MFD	Ford MFD	2N05	Ford	90	90					0	0			
7612032210RY								Placa Principal	8638541374	300	0	0			
7612032210RY								Placa Processador	8638545924	16	0	0			
7612032210RY								Placa SD Card	8638563308	30	0	0			
7612032230643	NAVIGATIONSSYSTEM; Ford MFD-Nav. B	Ford MFD	2N05	Ford	300	300					2100	0			
7612032230643								Placa Principal	8638541374	300	2100	0			
7612032230643								Placa Processador	8638545925	16	2100	0			
7612032230643								Placa SD Card	8638563308	30	2100	0			
7612032230SRY	NAVIGATIONSSYSTEM; Ford MFD-Nav. B	Ford MFD	2N05	Ford	90	90					0	0			
7612032230SRY								Placa Principal	8638541374	300	0	0			
7612032230SRY								Placa Processador	8638545925	16	0	0			
7612032230SRY								Placa SD Card	8638563308	30	0	0			
7612032230819	NAVIGATIONSSYSTEM; Ford MFD-Nav. B	Ford	2N05	Ford	1	1					11	8			
7612032230819								Placa Principal	8638541374	300	11	8			
7612032230819								Placa Processador	8638545925	16	11	8			
7612032230819								Placa SD Card	8638563308	30	11	8			
7612032215643	NAVIGATION SYSTEM	Ford	2N05	Ford	300	300					3600	3600			
7612032215643								Placa Principal	8638541374	300	3600	3600			
7612032215643								Placa Processador	8638546546	300	3600	3600			
7612032215643								Placa Sd Card	8638566548	300	3600	3600			
7612032215RY	NAVIGATION SYSTEM	Ford	2N05	Ford	90	90					3690	3240			
7612032215RY								Placa Principal	8638541374	300	3690	3240			
7612032215RY								Placa Processador	8638546546	300	3690	3240			
7612032215RY								Placa Sd Card	8638566548	300	3690	3240			
7612032215819	NAVIGATION SYSTEM	Ford	2N05	Ford	1	1					0	0			
7612032215819								Placa Principal	8638541374	300	0	0			
7612032215819								Placa Processador	8638546546	300	0	0			
7612032215819								Placa Sd Card	8638566548	300	0	0			
7612032235643	NAVIGATION SYSTEM	Ford	2N05	Ford	300	300					10500	13500			
7612032235643								Placa Principal	8638541374	300	10500	13500			
7612032235643								Placa Processador	8638546547	300	10500	13500			
7612032235643								Placa Sd Card	8638566548	300	10500	13500			
7612032235SRY	NAVIGATION SYSTEM	Ford	2N05	Ford	90	90					270	90			
7612032235SRY								Placa Principal	8638541374	300	270	90			
7612032235SRY								Placa Processador	8638546547	300	270	90			
7612032235SRY								Placa Sd Card	8638566548	300	270	90			
7612032235819	NAVIGATION SYSTEM	Ford	2N05	Ford	1	1					0	0			
7612032235819								Placa Principal	8638541374	300	0	0			
7612032235819								Placa Processador	8638546547	300	0	0			
7612032235819								Placa Sd Card	8638566548	300	0	0			

Figura 21: Exemplo de uma folha de excel.

OpCon

Este produto é o elo de ligação entre os vários de sistemas de produção - *Ekar*, *RLaser* e *Siplace Pro* - e o *software* de registo de transações *CMTracer*. O *OpCon* é, por assim dizer, o conversor de dados em *XML* (*OpenCon XML Interface*) para depois serem, mais tarde, processados pelo *CMTracer* através do *CMConveyorBox*.

XML Converter Interface

O *OpCon* só resolve o problema do software de alguns fabricantes. Para os restantes, *Pana Pro*, *Viscom*, *Koh Young*, é utilizada uma ferramenta que serve o mesmo propósito - *XML Converter Interface*. Como o próprio nome indica, esta ferramente processa e transforma os dados de informação em ficheiros *XML* a fim de serem armazenados no *CMTracer*. Quanto muito, o *OpCon* e o *XML Converter Interface* são mais pontes de interoperabilidade do que sistemas de informação. No entanto, ambos não divergem da definição anteriormente definida de Sistema de Informação.

SIPIA

SIPIA, Sistema de Informação Programável de Inserção Automática, consiste em supervisionar e, se necessário, efetuar alterações nas máquinas de inserção automáticas presentes nas diversas linhas de produção. Os utilizadores podem criar, alterar e apagar programas, configurações e planeamentos de

produção. Além disso, o *SIPIA* é o responsável por fornecer informação crucial acerca da organização de produção:

- tempo de ciclos dos *PCBs*;
- toda a produção existente numa linha;
- distribuição de receitas *PCB* pelas linhas de inserção automática.

Ao nível do acesso à informação proveniente do *SIPIA*, não se pode levantar qualquer tipo de objeção. Esta pode ser acessível diretamente através de um *browser* ou de exportação dos dados para uma folha de cálculo *Excel*. Em relação à comunicação com outros sistemas, interfaces, o *SIPIA* utiliza o:

- *Access Control (Acessum)*;
- *PIA*;
- *SIIA*.

O primeiro, *Access Control*, é utilizado para autenticação dos utilizadores no sistema. Os restantes servem como fontes de informação para, assim, conseguir supervisionar o panorâma geral das diversas linhas de produção. Por fim, é importante ressalvar que este produto de *software* está alicerçado numa base de dados *MySQL*.

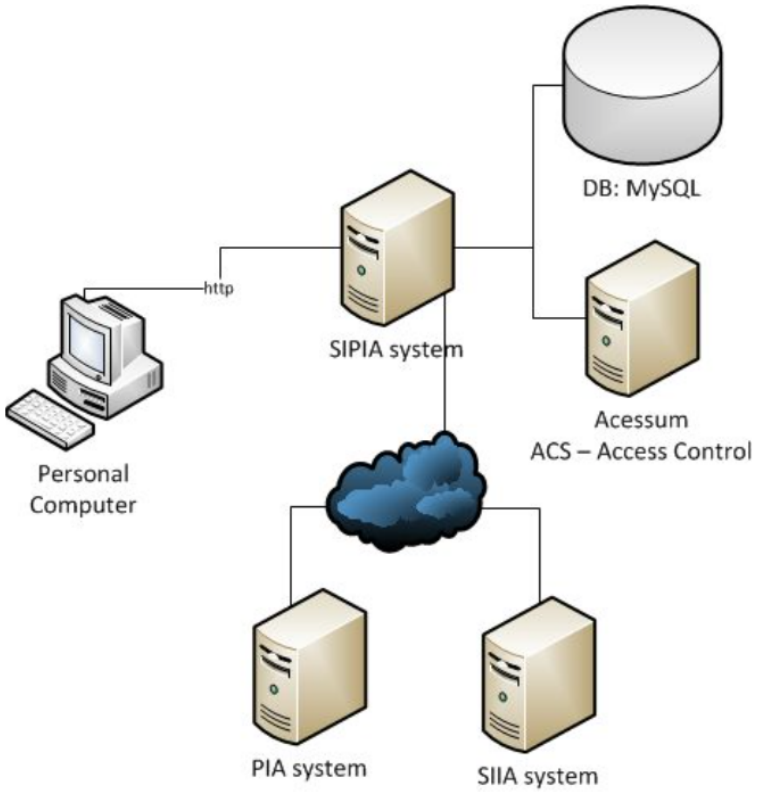


Figura 22: Arquitetura do sistema *SIPIA*.

3.2 TOPOLOGIA DO SISTEMA DE INFORMAÇÃO DA UNIDADE BOSCH

Após a leitura do subcapítulo anterior, *Domínio da produção de inserção automática de SMD*, a primeira ilação que se deve constatar é a panóplia de equipamentos de fabricantes diferentes. A relação de equipamento para fabricante chega quase a atingir o rácio de um para um, ou seja, os equipamentos e as máquinas utilizadas têm o seu próprio universo, as suas próprias regras e protocolos, a sua própria estrutura de dados, a sua própria arquitetura. Enquanto Ser Humano à superfície somos todos diferentes, mas no fundo no fundo somos todos iguais, mesmas necessidades, mesmo objetivos, mesmo ato de exprimir e comunicar. Aqui não se verifica isso. A única alternativa dos sistemas de informação exprimirem-se, comunicarem entre si, só é possível através outros Sistemas de Informação, outros produtos de *software* - o tal jogo do telefone. A maioria dos sistemas informáticos existentes acabam por descarregar as suas transacções no *CMTracer* a fim de serem consultadas por outros sistemas. Não há cooperação em tempo real. Não há interoperabilidade. Aqui nenhum *software* trabalha verdadeiramente em 'conjunto' com outro para atingir um determinado fim. O 'conjunto' é o *CMTracer* ou *SIPIA*. Os Sistemas de Informação principais da Unidade da *Bosch*. Um guarda e processa as transacções da maioria das linhas de produção de inserção automática, *CMTracer*, e o outro supervi-

siona e controla as máquinas e linhas de produção - *SIPIA*. Em suma, esta é a segunda ilação. A falta de verdadeira interoperabilidade entre os vários sistemas informáticos. A falta de eficiência no fluxo normal de trabalho e de processos, a predisposição para o erro humano.

O *SMART Manufacturing Control* visa combater e resolver estes problemas. De um modo geral, tenta substituir a maioria dos sistemas aglutinando-os a todos. Um supervisor inteligente e automático. Uma camada aplicacional superior que 'interliga' estes sistemas todos num só. [36, 37]

O verbo interligar adquire vários sentidos neste contexto. Em alguns casos, o interligar passa por normalizar a informação e disponibilizar essa mesma informação noutra estrutura, noutro motor de base de dados. Isto insere-se, principalmente, no âmbito da dissertação. O processo de normalizar, alterar a estrutura e a disponibilização só se consegue com a conversão correta de base de dados. Neste caso de *Firebird* para *Oracle*, e não, *Firebird* não serve de suporte a nenhum sistema descrito anteriormente, porém, a aplicação *SMART* foi desenvolvida ao longo dos anos em cima deste tipo de motor. Logo, de modo, a facilitar a sua implementação a nível de produção e testes reais, e também de aumentar os índices de normalização dos tipos de dados, tornou-se essencial a conversão de toda a estrutura de *Firebird* para motor *Oracle*. O próprio *CMTracer*, o responsável pelo registo da maioria das transacções na linha de produção, está assente numa base de dados *Oracle*.

Noutra perspetiva, o interligar é literal. Os sistemas estão conectados à plataforma *SMART* através de uma camada média aplicacional - *INTER*. Aqui, o *INTER* é responsável por automaticamente processar a informação proveniente dos vários Sistemas Informáticos e guardar essa mesma informação na base de dados do *SMART* através de agentes inteligentes. [38]

A topologia do SI encontra-se no **anexo A**.

4

MATERIAIS E MÉTODOS

Este capítulo apresenta e compreende todas as tecnologias estudadas e utilizadas para o sucesso deste projeto. O conjunto de todas as ferramentas ou conceitos que direta ou indiretamente ajudaram no desenvolvimento do conversor de base de dados *Firebird* para *Oracle*, o compilador. Todavia, antes de dar início à listagem e descrição das ferramentas, é preciso entender o que é um compilador, uma base de dados e o que o próprio *Firebird* e *Oracle* representam. Só com assimilação destes conceitos é que se pode perscrutar os restantes conceitos, tais como, gramática, *Yacc*, *Flex*, símbolos terminais e não terminais, expressões regulares, entre outros.

4.1 COMPILADOR

O que é um Compilador? Facilmente se consegue entender o porquê e o que é através desta definição [39]:

Compilador: *é um programa ou conjunto de programas que transforma uma linguagem de programação em outra diferente.*

De um certo modo, e fazendo uma comparação ao mundo real, o 'compilador' assemelha-se a um tradutor. Da mesma maneira que um tradutor, traduz frases e textos de uma linguagem para a outra; um compilador converte operações, códigos de uma linguagem de programação para outra. Porém, não é assim tão linear, visto que as linguagens de programação são incomparáveis à linguagem humana.

As linguagens de programação estão separadas por níveis de abstração - Linguagem de Alto Nível e Baixo Nível. As linguagens de baixo nível resumem-se a instruções de processador, de *hardware* - código máquina. Estas estão diretamente relacionadas com a arquitetura do computador. De uma certa maneira, são os 'átomos' da programação. Tudo o que é executado pelo computador tem que ser previamente convertido em código máquina. Em relação ao nível abstrato, pode-se dizer que é baixo ou quase nulo, já que se tem programar tudo desde o controlo de memória até às instruções do processador. No lado oposto, temos então as linguagens de alto nível de abstração. Aqui o programador já não se precisa de preocupar com organização da memória e instruções do processador, não precisa de ir ao detalhe. Este nível de processamento e de controlo já está implícito no código, na semântica usada, o que acaba por simplificar assim a vida do programador. No entanto, este tipo de linguagem

não pode ser executada diretamente pelo computador. Precisa-se de converter essa linguagem para código máquina a fim de ser interpretada e executada pelo computador. É aqui que entra o compilador. [40]

Sendo assim, podemos constatar que o compilador converte linguagens de alto nível em linguagens de baixo nível de abstração. E, da mesma maneira que há o complicado e o descomplicado, também existe um descompilador. Este corresponde à operação inversa, à passagem de uma linguagem de baixo nível para uma de alto nível de abstração. Por fim, temos o tradutor.

No contexto da compilação, um compilador pode ser um tradutor quando este processa uma linguagem de alto nível e retorna também uma linguagem do mesmo nível de abstração. Agora que já se compreende os vários níveis de compilação, podemos então estudar qual é a informação que vai ser processada e traduzida - base de dados.

Antes de mais, a conversão de base de dados é o ato de replicar uma estrutura existente de suporte à persistência de dados de um motor, uma linguagem noutro motor e noutra linguagem. No âmago da base de dados, a estrutura que guarda os dados e os próprios dados, resumem-se a linhas de código, a uma linguagem de programação. As *tables*, os dados, as *columns*, as *primary keys*, os *procedures*, funções, sequências, tudo é e pode ser descrito em texto, código e na linguagem de programação correspondente ao motor da base de dados. Geralmente são variantes de *PL/SQL*.

No âmbito desta dissertação, o programa de compilação desenvolvido assume, assim, a função de tradutor, já que a linguagem de programação, neste caso *PL/SQL*, quer do *Firebird*, quer do *Oracle*, é considerada uma linguagem de alto nível. Ou seja, o tradutor tem como função processar as *tables*, os dados e a restante estrutura da base de dados em *Firebird* e reescrever o equivalente em *Oracle PL/SQL*. [41]

É importante ressalvar o fato de um tradutor por mais simples que pareça, não deve somente arranjar correspondência entre palavras de linguagens diferentes. O Tradutor não deve descurar a arquitetura dos dois motores de base de dados, bem como optimização do código ou o próprio contexto das operações que estão a ser processadas.

4.2 BASE DE DADOS

Já se sabe que o tradutor focar-se-á na conversão de Base de Dados, mais concretamente de *Firebird* para *Oracle*, Base de Dados Relacionais. Genericamente, o objetivo e o significado do conceito Base de Dados pode-se inferir através do nome. Assim sendo, podemos dizer que uma Base de Dados é o que o próprio nome indica - uma estrutura de dados, um banco de dados [42]. Isto é, de um ponto de vista abstrato, ‘qualquer coisa’ que guarda dados, que os torna persistentes, arquivando para organização ou posterior consulta, e essa ‘coisa’ depende do tipo de Base de Dados:

- Modelo em Rede - Hierárquico;
- Modelo Relacional;

- Modelo Orientado ao Objeto;
- Modelo Semi-Estruturado;

Neste contexto, o objeto, a 'coisa' representa *tables*, *rows*, *columns*,*primary* e *foreign keys* - um Modelo Relacional. Na maioria dos casos, os conceitos primários e fundamentais referentes ao Modelo Relacional permanecem constantes nos dois motores de base de dados - *Firebird* e *Oracle*. Estas definições, quanto muito, só variam na sintaxe usada para descrição ou nos próprios tipos de dados usados para suportar a informação, como se pode constatar na figura que se segue.

Firebird

```
CREATE TABLE CODIGO_PAG_AUX (
    DOC         VARCHAR(50) NOT NULL,
    CAMPO       VARCHAR(50) NOT NULL,
    LINHA       INTEGER NOT NULL,
    TIPO        VARCHAR(10),
    CHAVE       VARCHAR(1),
    ORIGINAL    BLOB SUB_TYPE 1 SEGMENT SIZE 80,
    CORRIGIDO   BLOB SUB_TYPE 1 SEGMENT SIZE 80
);
```

Oracle

```
CREATE TABLE CODIGO_PAG_AUX (
    DOC         VARCHAR2(50) NOT NULL,
    CAMPO       VARCHAR2(50) NOT NULL,
    LINHA       NUMBER NOT NULL,
    TIPO        VARCHAR2(10),
    CHAVE       VARCHAR2(1),
    ORIGINAL    BLOB,
    CORRIGIDO   BLOB
);
```

Figura 23: Exemplo de sintaxe de criação de *Tables - Firebird vs Oracle*.

Além dos tipos de dados, não se denota mais nenhuma diferença. Porém, o caso muda de figura quando se começa a lidar com *functions*, *procedures*, *triggers*, objetos, entre outras entidades. Em suma, os obstáculos e a diferenciação entre as linguagens surgem quando se começa a aprofundar na complexidade e programação de operações para o motor de base de dados em questão.

4.2.1 *Firebird*

Como já foi referido anteriomente, a aplicação desenvolvida no âmbito da Bolsa de Investigação em que esta dissertação está fundamentada - o *SMART Manufacturing Control* - está assente numa Base de Dados *Firebird*. Isto deve-se principalmente ao facto do:

- Historial da aplicação;
- *Open source*;
- *Delphi*;

É importante salientar que o *Firebird* utilizado encontra-se na versão 1.5. Ademais, o desenvolvimento da aplicação é *a priori* ao início da Bolsa de Investigação. Os criadores e mentores do projetos optaram pelo *Firebird* em virtude do passado académico - especializados em *Delphi* - e dos custos

de licença. Em comparação com o resto dos motores de base de dados disponíveis no mercado, o *Firebird* em termos de custos e portabilidade para o *Delphi* separa-se da concorrência, já que há uma grande diversidade de bibliotecas de *Firebird* no *Embarcadero*; este último é, atualmente, o grande responsável pela linguagem *Delphi*. Além disso, a utilização do *Firebird* não implica qualquer tipo de custos, visto esta ser *open source*. No entanto, embora esta plataforma seja *open source*, a sua documentação é parca e deixa a desejar em relação à sua concorrência de mercado. A documentação não acompanha na íntegra as várias alterações que fazem ao longo das várias versões do *software*.

Resumidamente, o *Firebird* é uma base de dados relacional open source, disponível para *Linux, Microsoft Windows e OSX*. Atualmente está na versão 2.5 e ao nível das funcionalidades do motor também pode-se afirmar que na maior parte dos campos o *Firebird* fica aquém das espectativas em relação à sua competição - *Oracle, Postgres*. Como se pode verificar na figura que se segue, o *Firebird* está desprovido da maior parte das funcionalidades que se espera de um motor de base de dados. A restante comparação entre motores de base de dados encontra-se no **anexo B**.

Programming	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird
Stored procedures(*)	Yes	(Yes)(*)	Yes	Yes	Yes	Yes
Table functions(*)	Yes	Yes	Yes	No	Yes	Yes
Custom aggregates(*)	Yes	Yes	No(*)	No	No	No
Function overloading(*)	Yes(*)	Yes	No	No	Yes	No
User defined operators(*)	No(*)	Yes	No	No	No	No
Statement level triggers(*)	Yes	Yes	Yes	No	Yes	No
Row level triggers(*)	Yes	Yes	No	Yes	Yes	Yes
Before triggers(*)	Yes	Yes	(No)(*)	Yes	Yes	Yes
Dynamic SQL in functions(*)	Yes	Yes	No(*)	No	Yes	No
Dynamic SQL in triggers(*)	Yes	Yes	No	No	No	No
Delete triggers fired by cascading deletes(*)	Yes	Yes	Yes	No	Yes	Yes
Built-in scheduler	Yes	No	Yes	Yes	Yes	No

Figura 24: Comparação no âmbito do *PL/SQL*.

Continuando, a principal aposta do *Firebird* reside na concorrência, na performance e nos *stored procedures*. Os dois primeiros podem dar azo a discussão, visto que, durante o perído da Bolsa de Investigação, procedeu-se a uma série de testes em pequena escala e, até aí, o próprio motor já estava a encontrar problemas a nível de concorrência - *locks* - e de performance. No âmbito dos *stored procedures* e de um ponto de vista conceptual, os *procedures*, como se poderá observar mais tarde, são *tables*. Facilmente se pode fazer *select* ao *output* de *procedures*, retornar elementos, agregar *outputs* através de *joins*, *et cetera*. Isto permite uma maior manipulação e nível abstracional em relação à competição, como no caso do motor *Oracle*.

Além disso, de um ponto de vista relacional, também se pode constatar que o *Firebird* contêm algumas falhas no campo das relações entre *tables*. Após um estudo da base de dados utilizada durante a Bolsa de Investigação, *Firebird* 1.5, verificou-se que é possível a criação de *primary keys* com valores nulos. Isto vai contra as regras fundamentais de normalização dos dados e de base de dados relacional. Mais tarde, durante o desenvolvimento desta dissertação, verificou-se que isto consiste

num *bug*, chegando a ser reconhecido pela própria empresa responsável do *Firebird*. Porém o *bug* não foi corrigido e a própria base de dados está assente, na maior parte, neste *bug*.

Tipos de Dados

Ao nível dos tipos de dados não se encontra uma grande disparidade em relação a outros motores de base de dados, os próprios nomes não variam muito. No entanto não se deve deixar de referir que os tipos de dados do *Firebird* podem denotar uma certa ambiguidade, já que um tipo de dado em *Firebird* pode suportar e guardar, o que por norma, noutros motores de base de dados, são tipos diferenciados. Por outras palavras, não há uma especificidade na declaração de tipos, o que pode aumentar a predesposição do código para erro em *run time*. Por exemplo, um tipo *Char* do *Firebird* pode corresponder no *Oracle* a:

- *Char*;
- *Nchar*;
- *Raw*;
- *Long raw*.

Atualmente, na versão *Firebird* 1.5, os respetivos tipos cinge-se ao:

- *BigInt* - Inteiro de 64 *bits*;
- *Char(n)* - Cadeia de *n* caracteres;
- *Date* - Inteiro de 32 bits;
- *Decimal* - Precisão:1-18 e Escala:1-18;
- *Double Precision* - *Float* de 64 *bits*;
- *Float* - *Float* de 32 *bits*;
- *Integer* - Inteiro de 32 *bits*;
- *Numeric* - igual ao *Decimal*;
- *Smallint* - Inteiro de 16 *bits*;
- *Time* - Inteiro de 32 *bits*;
- *Timestamp* - Inteiro de 64 *bits*;
- *Varchar(n)* - Cadeia de *n* caracteres;
- *Blob Subtype (0,1)* - Variável dinâmica usada para guardar grandes quantidades de informação.

Daqui é também importante salientar que, ao contrário do *Oracle* em que o *varchar2(n)* é *size-dynamic*, ou seja, só ocupa o espaço que usa mesmo que declare um *varchar2* com 500 caracteres, no *Firebird* iss não se verifica. Ao declarar um varchar com 500 caracteres e desses se só se utiliza 5, o Varchar vai ocupar, na mesma, os 500 caracteres anteriormente declarados. No caso do *Blob*, este tipo de dado têm dois subtipos - 0 e 1. Ao contrário dos outros motores de base de dados, o *Firebird* não tem um tipo específico de *Lob* de texto, *Clob*. Este é um subtipo 1 do tipo *Blob*.

Tables, Constraints & Indexes

Como já foi aludido anteriormente, ao nível das *Tables* não se denota grandes alterações ao nível da arquitetura. A única diferença significtiva reside na declaração dos tipos de dados. De resto há, como se verificará mais à frente, somente, algumas diferenças ao nível da sintaxe da declaração de:

- *Constraints*;
- *Index*;
- *Primary Keys*;
- *Foreign Keys*;
- *Table & Columns Descriptions*.

No cômputo das relações, a única diferença, já enumerada, consiste na declaração de *primary keys* nulas.

Procedures

Os *stored procedures* são considerados um dos pontos fortes do *Firebird*. O *Firebird* tem uma abordagem diferente dos restantes motores de base de dados - um *procedure* também pode ser considerado uma *table* - o que facilita em grande parte o *nesting de procedures*. Muito facilmente se pode invocar *procedures* dentro de *procedures* e processar esses mesmos valores. Essa invocação é feita através de um simples select em que:

- O nome da *table* corresponde ao nome do *procedure*;
- O nome das *columns* corresponde ao nome das variáveis que retorna.

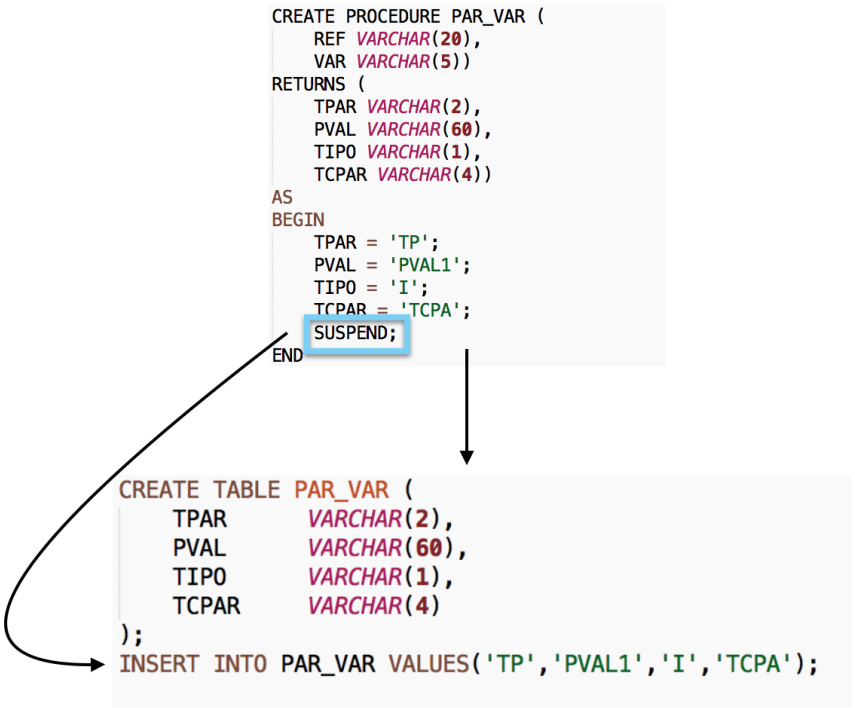


Figura 25: Exemplo do *Firebird* na abordagem dos *stored procedures*.

Através da figura 25 e tendo este exemplo em mente, facilmente se consegue interpretar e compreender a filosofia inerente aos *stored procedures* em *Firebird*. Aquando da execução do *procedure*, este imita o comportamento da *table* representada e, preenche-a através do comando *suspend*. Por sua vez, a instrução realiza um *insert* nessa *table* com os valores atuais das variáveis explícitas no *return* - as *columns*.

Por fim, a invocação de um *procedure* dentro de outro *procedure* resume-se a um *select*:

```
select TPAR from par_var('Par1','Par2') where TIPO = 'I';

<Resultado> 'TP'
```

Figura 26: Exemplo de invocação de um *nested procedure*.

Contudo, nem sempre é assim. No *Firebird*, há dois grupos de *procedures*, os que retornam variáveis e os que não retornam. Os *procedures* que seguem a estruturação de *tables* e de consulta através de *selects* - exemplo anterior - são os que retornam 'qualquer coisa'. No outro grupo encontram-se os *procedures* que não têm qualquer tipo de retorno. Estes não podem ser representados por *tables*. São *procedures* no sentido real da definição e a sua execução também não difere de outros motores de base de dados.

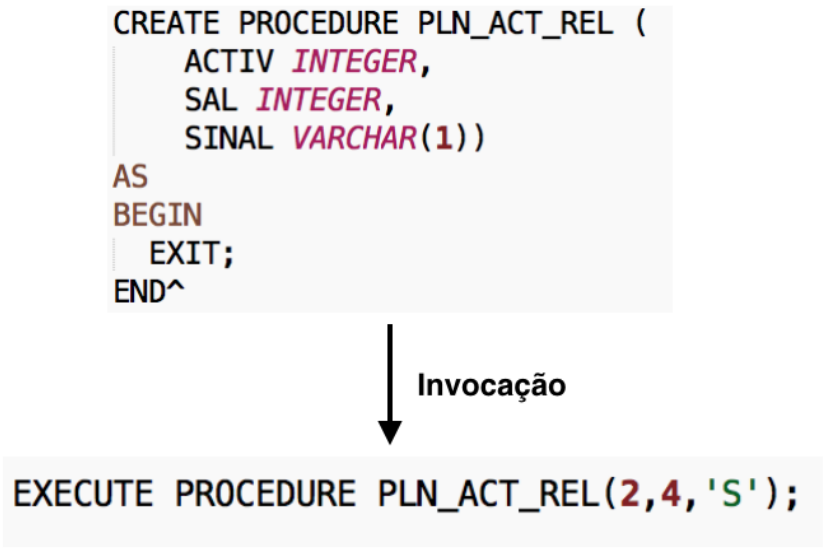


Figura 27: Exemplo de invocação de *procedure* sem *return*.

Functions

Como qualquer base de dados, o *Firebird* tem as suas próprias *functions* de sistema e suporta também a programação de *functions* de utilizador. Neste espectro, as *functions* também não diferem muito em relação a outras Base de Dados. Aqui não há nenhuma abordagem inovadora. O que se pode salientar é a redução da utilização de *functions*. Graças às funcionalidades dos *stored procedures*, em muitos casos, a utilização de *functions* é substituída pelos *procedures*. Isto também se pode verificar ao longo da Bolsa de Investigação, já que há 1280 *procedures* para 32 *functions*. As únicas *functions*, de utilizador, criadas em *Firebird*, ao longo da Bolsa de Investigação, equivalem a *functions* já existentes no sistema do motor Oracle, *functions* ao nível do processamento e conversão de tipo de dados e output. Em suma, o *Firebird* está aquém do motor *Oracle* a nível de *functions* de sistema.

Domains

Em comparação ao motor de base de dados *Oracle* - o único motor relevante no âmbito da Dissertação - os *domains* representam mais um obstáculo na conversão. O mais parecido que existe no *Oracle* denomina-se de Objetos. Ao contrário do *Oracle*, como se verificará mais tarde, no *Firebird* os *domains* só são usados como métodos de organização e simplificação da estrutura de base de dados. Permite a criação de 'tipos de dados' mais customizáveis.

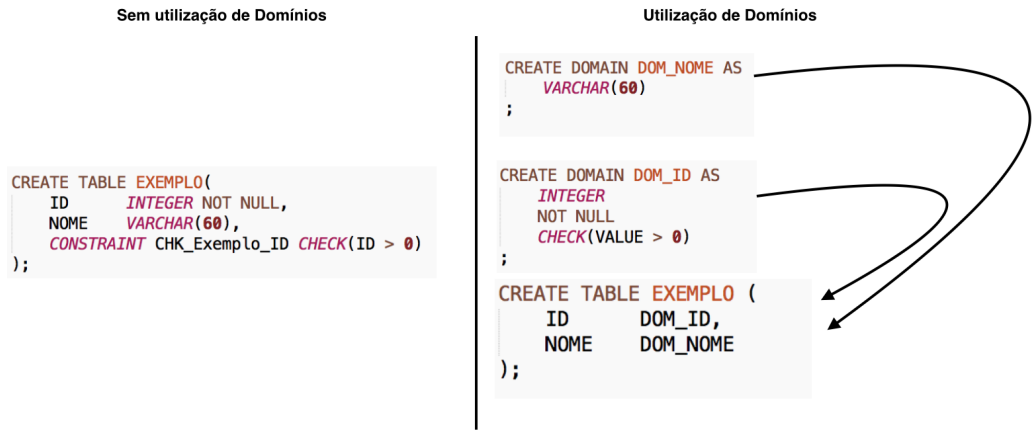


Figura 28: Exemplo de uma boa utilização de *domains*.

Como se pode constatar nesta figura, a utilização de *domains* permitiu uma redução na sintaxe necessária para a construção da *table*, bem como uma melhor organização das *columns*. Isto também permite a reutilização de código, já que o mesmo *domain* pode ser usado em *tables* e *columns* diferentes. Todavia, esta potencialidade não foi utilizada corretamente na bolsa de investigação, já que os *domains* cingiram-se em 100% dos casos a um estatuto de *aliases*.

```
CREATE DOMAIN DOM_CAM AS
NUMERIC(15,4);

CREATE DOMAIN DOM_CLTCOD AS
VARCHAR(3);

CREATE DOMAIN DOM_CORCOD AS
VARCHAR(5);

CREATE DOMAIN DOM_CPGCOD AS
VARCHAR(3);

CREATE DOMAIN DOM_CPSCOD AS
VARCHAR(3);

CREATE DOMAIN DOM_CVDCOD AS
VARCHAR(3);

CREATE DOMAIN DOM_DES20 AS
VARCHAR(20);
```

Figura 29: Exemplo de uma má utilização de *domains*.

Triggers

Em relação aos *Triggers* de outros tipos de base de dados relacionais existentes no mercado, não existe uma grande disparidade no *Firebird* ao nível da sintaxe e da filosofia intrínseca. Para todos os efeitos, é um *Trigger* genérico com poucas alterações na altura da declaração.

```
CREATE TRIGGER Nome_Trigger for Nome_Tabela
   [ACTIVE | INACTIVE]
   {BEFORE | AFTER} <acção>
   [POSITION number]
   AS
   <Bloco de Código>

<acção>          ::= <átomo_acção> [OR <átomo_acção> [OR <átomo_acção>]]
<átomo_acção>    ::= INSERT | UPDATE | DELETE
[.*]             ::= Não é um requirimento primário tudo o que está entre [].
```

Figura 30: Exemplo de sintaxe de um *trigger*.

4.2.2 *Oracle*

O motor da base de dados *Oracle* surge principalmente da necessidade de:

- Normalizar a informação;
- Resolver problemas de concorrência;
- Manutenção posterior;

No caso da normalização da informação, esta é essencial para o fluxo, para a consistência e persistência dos dados criados, processados e utilizados aquando da execução da aplicação *SMART*. Como já se referiu anteriormente, o *SMART Manufacturing Control* será a principal ferramenta de supervisão e utilização nas linhas de inserção automática da unidade de produção da *Bosch*. Esta amplitude de supervisão implica que haja uma quantidade enorme de informação, disseminada por várias sistemas de informação, que precisa de ser organizada e relacionada entre si. Mais tarde, verificou-se que o *Firebird* não conseguia, com a rapidez necessária, dar resposta ao enorme fluxo de informação criada durante a produção nas linhas de inserção automática. Além disso, as principais fontes de informação do sistema informático da *Bosch* estão assentes em motores de base de dados iguais ou provenientes da mesma empresa:

- *CMTracer - Oracle*;
- *SIPIA - MySQL*;

Logo, a disparidade de tipo de dados, do processamento e execução de *functions*, *procedures*, *triggers*, entre outros, é quase nula ou baixa em comparação com o *Firebird*. Isto não só simplifica a

complexidade aquando da programação da plataforma *SMART*, bem como reduz consideravelmente os problemas que possam surgir por correspondência de tipo de dados, output de *procedures* ao nível da programação. No âmbito da concorrência, verificou-se também, em fase de testes, que o *Firebird* não está dotado de boas práticas, regras ou motor para a resolução dos conflitos de concorrência que possam surgir das enormes quantidades de informação recebidas e da sua posterior manipulação. Neste campo, o motor Oracle provou-se ser eficiente por observação do caso *CMTracer*. No âmbito da produção, o *CMTracer* é o responsável por receber e processar todas de transações de variados tipos.

A base de dados *Oracle*, ao contrário da *Firebird*, não é *open source*. Esta pertence a uma das maiores empresas de *software* do mundo, *Oracle Corporation*, sendo necessário comprar uma licença para posterior utilização no mundo empresarial. É importante também referir que esta licença não se traduz somente numa permissão de utilização. Esta confere, um apoio tecnológico por parte da *Oracle*, bem como um acesso a uma maior quantidade de documentação. Não obstante ao facto de mesmo sem licença e as consequentes regalias, a quantidade de documentação disponível e exemplos na *internet* sobrepassa o *Firebird*. Isto deve-se em grande parte a uma maior utilização, por parte da comunidade científica e tecnológica, de produtos *Oracle*.

É importante referir que a base de dados *Oracle* a ser utilizada, na bolsa de investigação, encontra-se na versão *Oracle 11gR2*.

Tipos de Dados

Ao contrário do *Firebird*, no *Oracle* existe uma maior discriminação e separação dos vários tipos de dados. Não há tanta ambiguidade como no *Firebird*. Na versão 11gR2, a informação pode ser caracterizada como:

- *Char(n)* - Cadeia de caracteres, sem espaços e com tamanho *n* fixo;
- *NChar(n)* - Cadeia de caracteres NLS, sem espaços e com tamanho *n* fixo;
- *NVarchar2(n)* - Cadeia de caracteres NLS com um tamanho *n* e variável;
- *Varchar2(n)* - Cadeia de caracteres com um tamanho *n* e variável;
- *Long* - Cadeia de caracteres com um tamanho máximo de 2GB;
- *Raw* - Cadeia de caracteres binários;
- *Long Raw* - Cadeia de caracteres binários com um tamanho máximo de 2GB;
- *Number e Numeric (p,e)* - Precisão:1-38 e Escala: -84-127;
- *Float - Float* de 32 *bits*;
- *Dec e Decimal (p,e)* - Precisão:1-38 e Escala: -84-127;

- *Integer*;
- *Int*;
- *SmallInt*;
- *Real*;
- *Double Precision*;

Até agora ainda só foram descritos os tipos relacionados com caracteres e números. Os últimos 5 não têm descrição devido ao facto de se puderem agrupar no mesmo tipo de dados - Number. O *Integer*, o *Int*, o *SmallInt*, o *Real*, o *Double precision* são todos abstrações do *Number*, em que só a precisão e a escala é que diferem. Por exemplo, ao se declarar um *Integer*, este vai ser guardado ou processado como um *Number* de precisão 38. Isto acaba por ajudar em termos de performance, já que é muito mais eficiente utilizar um *Number* do que um inteiro, e, da perspetiva do utilizador, pode se tornar útil caso este não tenha muita experiência com motores de base de dados. Ao nível das Datas e *Lobs*, o *Oracle* divide-se por:

- *Date*;
- *Timestamp*;
- *Interval year*;
- *Interval day*;
- *Bfile*;
- *Blob*;
- *Clob*;
- *NClob*;
- *Rowid*;
- *Urowid*;

Daqui é importante distinguir, mais uma vez, a maior oferta de tipos e funcionalidades que um utilizador tem à sua disposição. O *timestamp* está dividido em 3 subtipos - sem *timezone*, com *local timezone* e *session timezone*. O *interval year and day* são dois tipos que não existem de todo, nem aproximações, no *Firebird*. Estes últimos tipos de datas podem ser representadas por intervalo de anos ou dias. Ao nível dos *Lobs*, o *Oracle* disponibiliza uma maior gama de opções, desde *file locators*, no caso do *Bfile* até 3 tipos de *Blob*, dependendo se estes *lobs* estão representados em binário - *Blob* - caracteres - *CLob* ou até mesmo *Unicode* - *NClob*. Os últimos dois tipos, *Rowid* e *Urowid*, iam servir de

auxílo na conversão do motor de Base de dados *Firebird* para *Oracle*. No entanto, descobriu-se uma solução mais simples e robusta. Como já foi mencionado anteriormente, os *procedures* em *Firebird* são, na maior parte, *tables* - um conjunto de *rows* e *columns*. Por sua vez, o *Rowid* e *Urowid* são tipos de dados que representam as *rows* de uma *table*.

Tables, Constraints & Indexes

Tal como foi descrito na secção do *Firebird*, as diferenças prendem-se com os tipos de dados das *columns* e com a sintaxe de certos objetos: *constraints* e *indexes*. Ao nível da declaração de chave primárias, o motor *Oracle* não permite a declaração destas como *null*.

Procedures

Este é o principal obstáculo na conversão. Os *procedures* representam a maior discrepância ao nível da estrutura e do seu funcionamento. Voltando ao *Firebird*, pode-se verificar que estes têm a possibilidade de retornar variáveis - neste caso os *procedures* funcionam como *tables*. No *Oracle*, só as funções é que têm a possibilidade de retornar verdadeiramente variáveis. Em *procedures*, o mais parecido com o retorno - *return* - denomina-se de parâmetro de saída. Isto consiste em variáveis cujo o *scope* não está circunscrito ao *procedure* que as recebe. Qualquer alteração realizada nessa variável é guardada:

```
CREATE OR REPLACE PROCEDURE IncrementaNum (num_out OUT number)
begin
    num_out:= num_out+1;
end;
```

```
CREATE OR REPLACE PROCEDURE Procedimento_PAI
is
    xnumber number;
begin
    xnumber := 0;
    IncrementaNum(xnumber);
    DBMS_OUTPUT.put_line(xnumber);
end;
```

xnumber = 1

Figura 31: Exemplo de um *procedure* - parâmetro out.

A figura 31 representa um *procedure* em *Oracle*. Assim sendo, pode-se então verificar que os *procedures* não têm um *return* genérico. Estes usam uma *flag* **OUT**, aquando da declaração dos

parametros, de modo a indicar que este parâmetro tem que ser devolvido com o valor resultante da execução. Neste caso, observa-se que o *xnumber* está a 0 quando declarado como parâmetro do *nested procedure* e acaba por ser devolvido ao *procedure* principal já com o novo valor - 1. Os parâmetros que são declarados num *procedure* com a *flag* **IN** representam parâmetros de entrada, em que os novos valores obtidos, aquando da execução desse *procedure*, não são trespassados para outros *procedures*. No capítulo seguinte poder-se-á aferir que, de um ponto de vista programacional, será impossível a conversão literal de todos *procedures Firebird* para *Oracle*. Somente os *procedures* em *Firebird* que não têm qualquer tipo de return serão replicados em *procedures* Oracle. Os restantes, em termos leigos - os *procedures tables* - serão traduzidos para funções *Oracle* em que o retorno consiste num objeto do tipo *table*.

Functions

Após a conversão, a maior parte das funções existentes na base de dados *Oracle* serão *procedures* de *Firebird*. No entanto, em termos do número de funções disponíveis para o utilizador o *Oracle* ultrapassa o *Firebird*. Um grande número das funções de *Firebird*, criadas ao longo da bolsa da investigação, já existem por defeito no motor Oracle. Assim sendo, só se precisa de proceder à correspondência de funções entre os dois motores, ao invés da tradução inicialmente considerada. No geral, as funções de *Firebird* incidem-se na temática do processamento de *output* ou conversão de tipos.

Firebird Funções de Utilizador	Oracle Funções de Sistema
ASCII_CHAR	ASCIISTR
ARREDONDA	ROUND
B_LONGSUBSTR	SUBSTR
CALCULAEXP	EXP
CEILING	CEIL
DIV	'/'
FLOATTOSTR	CAST/TO_CHAR
FLOOR	FLOOR
STRLEN	LENGTH
STRREPLACE	REPLACE

Figura 32: Correspondência entre algumas funções de *Firebird* e *Oracle*.

Aqui pode-se clarificar o avanço do *Oracle* em relação à base de dados *Firebird*. Do lado esquerdo estão funções que deviam ser genéricas de um sistema de base de dados, no entanto, estas foram todas definidas pelo utilizador. Em relação ao *Oracle*, estas já estão definidas e prontas para utilização.

Domains

O título desta subsecção - *Domain* - não tem qualquer representação literal no *Oracle*. Não existe. Todavia, este existe no *Firebird* e, como tal, é necessário conciliar este conceito com a arquitetura do motor *Oracle*. O mais próximo em relação aos *domains Firebird* são os Objetos do *Oracle*. Porém, é importante ressalvar que ambos não têm o mesmo objetivo. Enquanto que os *domains* têm uma propósito organizacional e de simplificação, os objetos em *Oracle* vão de acordo à programação orientada a objetos.

Consequentemente, a utilização destes objetos permite 'partir' um sistema complexo em entidades lógicas, mais simples. Ou seja, os objetos facultam a criação de código modular, reusável e de fácil manutenção. Um objeto é, e indo de acordo com a programação orientada a objetos, um tipo de dados que permite a declaração de:

- Variáveis;
- Funções;
- Métodos.

Em que estes últimos são utilizados para manipular as variáveis - atributos - declarados no objeto. Assim, acaba-se por encapsular blocos de códigos, operações e funções com os seus respetivos atributos. Levando a uma maior simplificação do código *PL/SQL* - grande parte do código de *procedures* pode ser substituído por invocações de métodos de objeto; isto também permite uma representação mais organizada e fiel do contexto em que a base de dados está inserida.

Como já foi referido anteriormente, os *domains* de *Firebird* não serão verdadeiramente convertidos, visto que eles só estão a ser usados como 'aliases' de tipo. Então onde é que os objetos se enquadram na conversão? Como se verificará no capítulo seguinte, os objetos do *Oracle* serão utilizados para imitar o comportamento dos *procedures-table* do *Firebird*.

Triggers

Este é mais um tópico em que não se verifica uma grande disparidade de sintaxe - escrita - ou princípios aquando da comparação com o *Firebird*. Contudo, após uma observação da informação disponível, pelo Oracle, acerca dos *triggers* poder-se-á verificar que estes são mais complexos e oferecem uma maior customização e configuração. Como a base de dados em *Firebird* da bolsa de investigação já está assente em *triggers* simples, torna-se impossível a utilização do imenso potêncial da customização dos *triggers* do *Oracle* após a tradução.

5

TRADUTOR

Já se sabe o que é um compilador, o *Firebird* e o *Oracle*. Estes são os três pilares da criação desta dissertação. A gênese, os objetivos, a motivação desta tese 'orbitam' em torno destas 3 entidades. A gênese está inserida no âmbito de uma bolsa de investigação existente, *HMIEXCEL - I&D crítica em torno do ciclo de desenvolvimento e produção de soluções multimédia avançadas para automóvel*. O objetivo proposto enquadrava-se no ato de replicar o suporte à persistência dos dados, base de dados, de *Firebird* para *Oracle*:

- 612 - *Tables*;
- 1280 - *Procedures*;
- 32 - *Functions*;
- 113 - *Triggers*;
- 10 - *Views*;
- 60 - *Domains*;
- 153 - *Indexes*;
- 580 - *Constraints*;
- 283 - *Descriptions*.

Agora somando tudo e tendo em conta que o prazo limite para realizar a conversão era de 9 meses, seria possível uma pessoa estudar cada uma das arquiteturas da base de dados e manualmente reescrever centenas de milhares de linhas de código? Mais de 1 milhão de operações? Não se sabe, talvez conseguisse antes de dar entrada numa ala psiquiátrica, mas e no fim? Será que a conversão estaria 100% correta? Muito dificilmente não. O ser humano erra e, neste contexto, a margem de erro de aumentaria com o passar dos dias. Além disso, a fase de testes e de correção da conversão seria reduzida tanto em tempo como em eficácia. Já que a maior parte dos 9 meses seriam dedicados à reescrita da base de dados.

E qual seria o fruto da bolsa de investigação, que novas competências teria a pessoa, responsável pela conversão, aprendido ou aprofundado? Somente a arquitetura das duas base de dados e um aumento significativo na tolerância, na paciência e na rapidez de escrita?

Após uma ponderação dos objetivos com o contexto da dissertação, a motivação passou pelo desenvolvimento de uma ferramenta capaz de realizar essa conversão da base de dados de forma automática. O resultado é um tradutor (nível de compilador), que permite a conversão das base de dados em *Firebird*, utilizadas no âmbito da bolsa de investigação, para *Oracle*.

Por fim, como se verificará mais tarde, é importante ressalvar o modo como este tradutor foi arquitetado, visto que esta ferramenta permite facilmente a alteração do *output* da conversão. Rapidamente se consegue alterar a linguagem para a qual o tradutor converte, p.e. *Firebird* para *MySQL*. O tradutor não se restringe somente à conversão literal de *Firebird* para *Oracle*. O Tradutor realiza, em primeiro lugar, um estudo e uma aprendizagem da estrutura total da base de dados *Firebird* em que processa e relaciona os vários tipos de informação:

- *Tables*;
- *Columns*;
- *Domains*;
- *Functions*;
- *Procedures*;
- *Triggers*;
- *Indexes*
- *Constraints*;

e só depois procede à conversão. Como o total dos dados já estão processados e guardados, só é preciso alterar a forma como escreve. Uma simples alteração de 'prints' permite a alteração de como representamos essa informação. Assim sendo, facilmente se consegue alterar a linguagem, a estrutura de output, o motor de Base de Dados para o qual se quer converter. A partir daqui compilador e tradutor serão usado como sinónimos. Porém, ainda falta explicar e descrever os vários mecanismos, ferramentas e algoritmos utilizados para a construção de um tradutor, bem como os próprios mecanismos de um tradutor genérico.

5.1 COMO É QUE FUNCIONA?

Como já foi referido anteriormente, um compilador recebe informação, dados, texto numa linguagem específica e retorna o equivalente noutra linguagem. Leigamente o ato da compilação pode ser

demonstrado, por agora, de um modo simples, através da linguagem humana. Hipoteticamente, estamos perante um Senhor que pediu ao filho para traduzir a frase - *Olá! Sou o José e moro em Braga* - para inglês. Em primeiro lugar, o filho só consegue traduzir se compreender o que está escrito em Português. Precisa de entender tanto o significado de cada palavra, como a gramática, a estruturação inerente à palavras, a construção da frase. Abstratamente, de um lado podemos dizer que as palavras podem ser representadas por peças de um puzzle, a própria gramática é o encaixe das peças e a frase é o puzzle.

Além disso, o filho do Senhor José também precisa de dominar o significado das palavras e da gramática em inglês a fim de traduzir a frase. Só assim é que o filho consegue traduzir a frase em português para inglês.

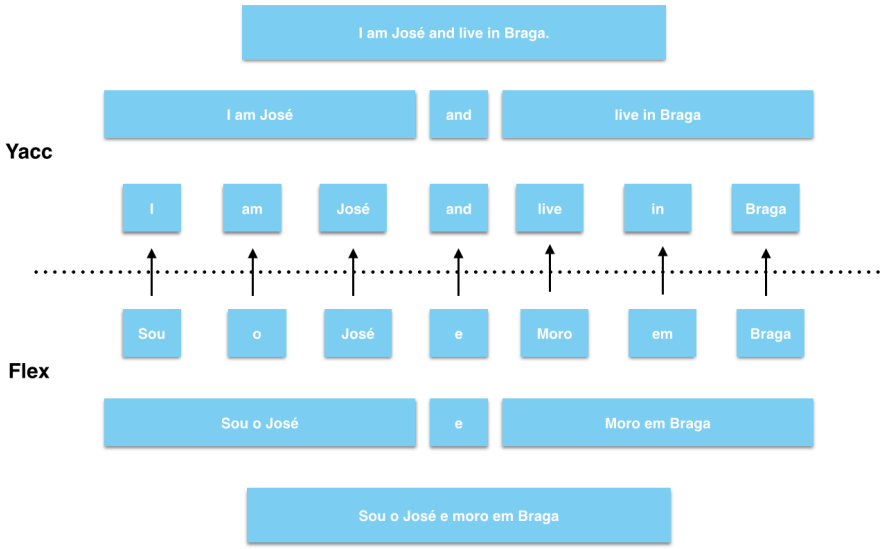


Figura 33: Relação entre *Flex* e *Yacc*.

Na perspetiva de um tradutor informático, a filosofia é igual. A conversão só é possível se houver um 'dicionário' com uma listagem de todo o campo léxico existente e que possa surgir na altura da leitura da informação, bem como um conjunto de regras gramaticais que vá processando, decompondo e validando sintáticamente essas mesmas frases a fim de compreender o que a própria frase representa. Só após a decomposição e processamento das frases é que o tradutor pode dar início à validação semântica, sintática e a posterior conversão. Essa conversão é realizada através de algoritmos em *C* com o auxílio do *Yacc*.

Concluíndo, pode-se afirmar que o tradutor é constituído por 3 fases principais:

- Estudo Lexical - Dicionário;

- Estudo e validação gramatical - Regras Gramaticais;
- Tradução - Algoritmos;

e cada fase foi desenvolvida, respetivamente, com ferramentas e arquiteturas próprias:

- *Flex*;
- *Yacc*;
- Código em *C*.

5.2 FLEX

Flex - Fast Lexical Analyzer - é um analisador léxico, uma ferramenta que possibilita a criação do tal 'dicionário', a representação e organização do campo lexical [43]. Por outras palavras, é o sítio onde ensinamos ao tradutor, através de expressões regulares, a constituição do campo lexical, como por exemplo:

- Conceito de letra - um caracter que estejo situada entre *aA e zZ - [aA-zZ]*;
- Conceito de digito - um caracter descrito entre *0, 1, 2, 3, .., 9 - [0-9]*;
- Conceito de Palavra - um conjunto finito de letras - *[aA-zZ-]+*;
- Conceito de Número - um conjunto finito de digitos - *[0-9]+*;
- Conceito de String - Conjunto de palavras ou digitos entre aspas ou plicas - *('—")[aA-zZ0-9 -.,(...)]+('—")*;

entre outros. Inicialmente, é assim que se contrói um analisador léxico, porém só isto ainda não é suficiente. A representação do campo léxical ainda está muito ambígua. Como na Língua Portuguesa ou em qualquer outra língua, existem palavras reservadas, palavras fundamentais para o sentido da frase, da comunicação, tais como *e, mas, porque, meu, eu, et cetera*. No caso das linguagens de programação, estas também não diferem: *int, function, select, where* são palavras reservadas do sistema, da língua. Conferem o contexto, o sentido e a definição à instrução. Por conseguinte, é também necessário proceder à declaração e agrupamento destas palavras, destas definições no analisador léxico, o *Flex*, a fim de aumentar a robustez, a solidez e consistência do tradutor, não obstante ao fato de ajudar na modularidade do desenvolvimento do mesmo.

```
"UPDATE"|"update"                    {return UPDATE;}
"UPPER"|"upper"                      {return UPPER;}
"VALUE"|"value"                      {return VALUE;}
"VARIABLE"|"variable"                {return VARIABLE;}
"VALUES"|"values"                    {return VALUES;}
"VARCHAR"|"varchar"                  {return VARCHAR;}
"WHEN"|"when"                        {return WHEN;}
"WHERE"|"where"                      {return WHERE;}
"WHILE"|"while"                      {return WHILE;}
"WRITE"|"write"                      {return WRITE;}
"''''"                               {yylval.nome = strdup(yytext); return string;}
"'''NULL'''"                         {yylval.nome = strdup(yytext); return string;}
[0-9]+                               {yylval.num = atoi(yytext); return number;}
[aA-zZ0-9_\$]+                       {yylval.nome = strdup(yytext); return word;}
```

Figura 34: Exemplo da analisador léxico - *Flex*.

Nesta figura já se pode encontrar a declaração de algumas palavras chaves e definições de conceitos do campo léxical. Porém o importante a reter da imagem consiste no facto de uma declaração ter a sua respetiva acção. Ou seja, sempre que o analisador léxico encontra essa palavra ou expressão regular na informação a traduzir, este executa a respetiva acção. Geralmente esta acção consiste no envio de um 'alerta', um *token* único e representativo da palavra encontrada para o *Yacc* - o responsável pelo estudo e validação gramatical. Em seguida, o *Yacc* começa a juntar ordernadamente os vários *tokens* de acordo com as regras.

Em suma, o *Flex* é a primeira etapa do tradutor, o ponto de entrada de informação. Começa a decompor o texto tendo em conta as expressões regulares e declarações presentes no dicionário e à medida que vai encontrado *matches* para essas expressões, o *Flex* envia-as através de um sinal, um *token*, para o *Yacc* a fim de ser analisadas sintaticamente de acordo com as regras gramaticais. [44]

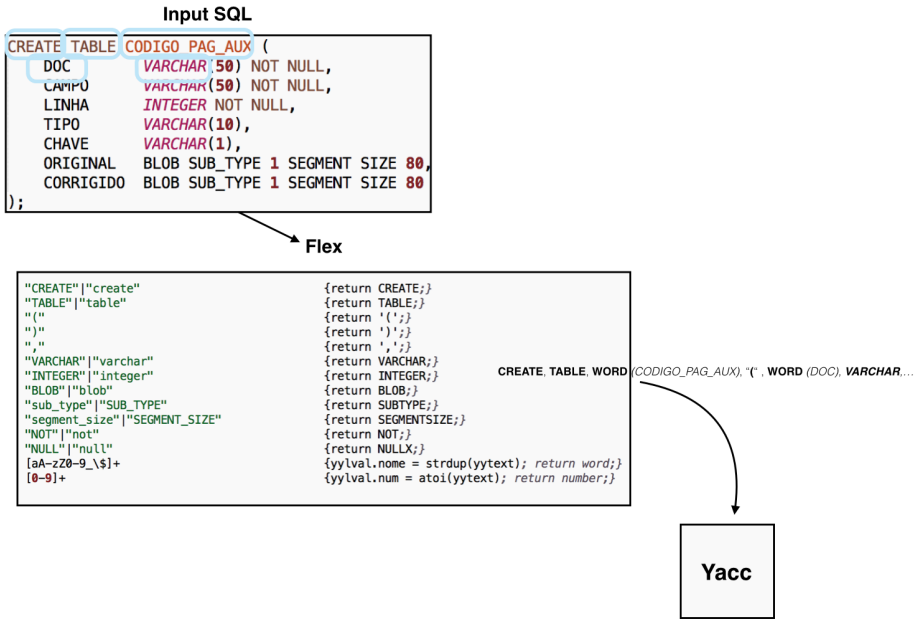


Figura 35: Utilização do *Flex* com o *Yacc*.

Através da leitura do ficheiro *Flex* presente nesta figura, pode-se observar que este contém a declaração de palavras reservadas do *PL/SQL* - *Create*, *Table*, *VARCHAR*, *Integer*, *Blob*. As restantes declarações são expressões regulares que permitem obter o resto da informação importante. Agora, de modo a exemplificar o processo do *Flex* e a sua comunicação com o *Yacc*, é pretendido que o código do *create* table, representado por *Input SQL*, seja convertido. Em primeiro lugar, invoca-se o analisador léxico, *Flex*, para processar o documento. Como se pode constatar, este decompõe o texto de acordo com as expressões declaradas. O *Flex* começa por encontar a chave *Create*, *Table* e, de seguida, uma palavra que, em princípio, é o nome da *Table*. É importante retificar que, mais tarde, esta palavra é verificada e contextualizada no *Yacc*. Assim sendo, conforme vai decompondo, este envia as chaves encontradas para o *Yacc* e o processo não se altera até ao fim do ficheiro.

5.3 YACC

Voltando a usar a linguagem humana como motee, facilmente se consegue perceber onde se encaixa o *Yacc* no espectro do tradutor.

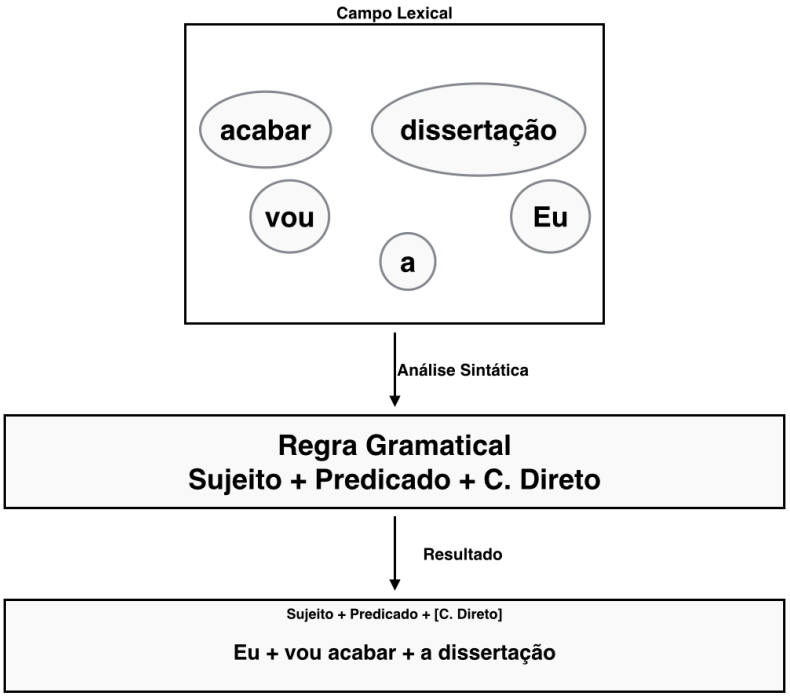


Figura 36: Exemplo de análise sintática.

Uma frase é um conjunto de palavras organizadas com um sentido lógico e completo e, isso só é possível, através de regras gramáticais, regras de construção frásica. Aliás, como se pode observar pela imagem, qualquer outra combinação de palavras tornaria fútil a leitura da frase; e se a leitura é impossível, por consequência será também impossível compreender e analisar semântica a frase. Sem análise semântica é impossível converter fielmente uma frase, uma instrução, bloco de código, já que as palavras têm contextos diferentes, conotações escondidas que precisam de ser tidas em conta no momento da tradução. É imperativo, antes de se proceder à conversão, a criação e o desenvolvimento de um analisador sintático.

Yacc - Yet Another Compiler-Compiler - é uma ferramenta que possibilita o desenvolvimento do analisador sintático - a validação de frases, de conjunto de componentes léxicos [45]. Um programa alicerçado no *Flex*. Há medida que o analisador léxico vai encontrando palavras, este envia-as para o analisador sintático a fim de serem validadados. Por outras palavras, o *Yacc* valida a sequência ordenada de *tokens*, enviados pelo *Flex*, de acordo com as regras gramaticais definidas.[46]

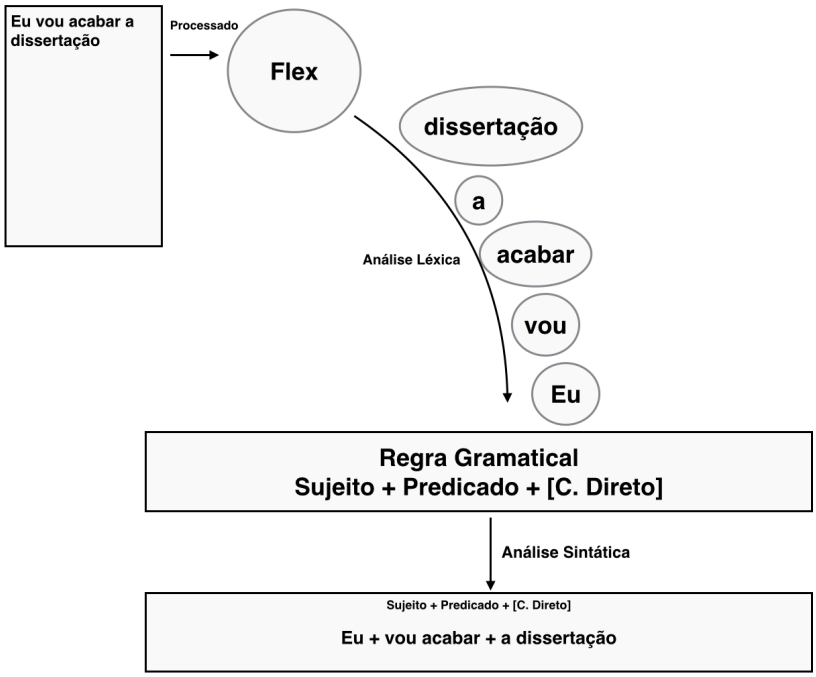


Figura 37: Sequência ordenada de *tokens* do *Flex* para o *Yacc*.

Genericamente, o Yacc consiste no:

Figura 38: Visualização genérica de um ficheiro *Yacc*.

A estrutura de um documento *Yacc* não difere muito de um documento *Flex*. Do mesmo modo que no *Flex* se precisava de declarar as chaves e a sua respetiva acção, no *Yacc* também se segue os mesmos princípios. Uma Regra equivale a uma acção e o que é uma regra? . Esta é um conjunto de:

- Símbolos Terminais;
- Símbolos Não Terminais;

Os Símbolos Terminais são caracteres literais que estão presentes nas regras ou *tokens* recebidos do *Flex* [47]. Por sua vez, os simbolos não terminais, podem ser classificados, leigamente, como sendo outras regras de gramática. Uma regra gramatical pode ser o conjunto de subregras.

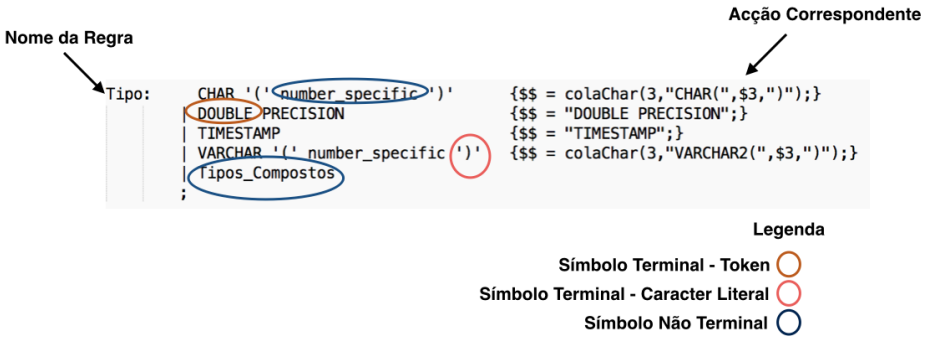


Figura 39: Exemplificação de uma regra.

Por fim, é importante ressalvar que as acções só são executadas quando a regra é concluída. Estas serão analisadas na secção seguinte.

5.4 ALGORITMOS C

O *Flex* é um analisador léxico - permite selecionar as palavras que se coadunam no universo léxico especificado. O *Yacc* é um analisador sintático em que valida os componentes léxicos recebidos pelo *Flex* de acordo com a gramática. Porém, ainda falta um analisador que permita tirar ilações da informação que recebe, possibilitando assim uma tradução fiel e contextualizada do que se pretende. Isto é conferido pelo desenvolvimento e utilização de algoritmos de análise semântica em *C*. Esta era a única linguagem que se podia conciliar com estas ferramentas. No entanto, os algortimos utilizados no tradutor não se resumem ao processamento de informação, análise semântica. É necessário criar estruturas, algoritmos, métodos que permitem e facultem a conversão da informação. Esses algoritmos auxiliares podem-se agrupar em:

- Gestão de Memória;
- Correção de *Strings*;
- Gestão de Conhecimento;
- Análise Semântica;

5.4.1 *Gestão de Memória*

A programação em *C*, por si só, e em projetos de grandes dimensões, torna-se um desafio aliciante. Ainda mais aliciante se torna quando 90% do código resume-se ao processamento de quantidades enormes ficheiros, sobretudo à manipulação de milhões de *arrays* de caracteres. Mais dificilmente se consegue garantir que não haja acessos indevidos de memória, problemas de *overflow* ou até mesmo *segmentation fault*.

A solução para este obstáculo incide-se na prevenção. Sempre que um conjunto de caracteres - uma *string* - é reconhecida pelo *Flex* e enviada para o *Yacc*, o primeiro passo consiste no alocamento dessa *string* em memória. O seu respetivo apontador fica guardado numa estrutura, e este passo repete-se até ao fim da leitura do bloco de código em questão. Quando termina a leitura, o tradutor, de seguida, começa a guardar a informação relevante para a tradução ou extração de conhecimento. No fim, este percorre a estrutura que armazena os apontadores utilizados, limpando um a um, o respetivo espaço ocupado, e por consequência, aumentando mais uma vez, o espaço livre utilizável.

Os métodos responsáveis pelo alocamento do espaço para cada palavra, *string* e *array* de caracteres também estão assentes na prevenção. Além do mais, é importante ressalvar que estes métodos são dinâmicos. Através de bibliotecas específicas, conseguiu-se desenvolver métodos que não tivessem um número fixo de parâmetros. Isto permite um maior nível de manipulação e simplificação da arquitetura do tradutor, visto que o número de parâmetros a serem processados nunca é constante.

Voltando à alocação, esta é realizada de modo a reduzir as probabilidades de acesso indevidos a zonas de memórias. Sempre que um conjunto de caracteres é passado por parâmetro para um dos métodos de alocação, este verifica o tamanho em *bytes* do conjunto e faz o *malloc* consoante esse resultado. Esse resultado acaba também por ser guardado junto com o apontador. Assim ao limpar volta-se também a verificar se o tamanho prestes a ser limpo, coincide com o espaço ocupado pelo respetivo *array* de caracteres. Este tipo de comportamento reduziu imensamente os casos de problemas relativos a memória.

A respeito dos métodos de alocamento, estes resumem-se a dois:

- *ColaString*;
- *ColaChar*;

Os dois têm o mesmo objetivo e função - alocação dos caracteres, prevenção de falhas de memória, bem como o reencaminhamento do conjunto de caracteres alocados. Como já foi referido anteriormente, a informação relevante é guardada para posterior processamento e extração de conhecimento e, de modo a conseguir isso, é necessário a criação de várias estruturas diferentes, mas relacionadas entre si. Os métodos *ColaString* e *ColaChar* são os responsáveis pelo envio da *string* para a estrutura correspondente. Por exemplo, a informação relativa a *tables*, *columns*, *primary keys*, *et cetera* são remetidas para a estrutura referente ao armazenamento do *schema* da base de dados. Porém, de

modo a alcançar esta meta, tal cenário só é possível através da junção destes métodos com as regras gramaticais declaradas no fichero *Yacc*. Cada regra gramatical pode ter uma acção associada.

A diferenciação entre os dois métodos resume-se num simples caracter, o espaço. Aquando do desenvolvimento do tradutor, verificou-se a necessidade de inserir ou manipular espaços em certos *array* de caracteres. O *colaString* assegura que isso aconteça. Por sua vez, o *colaChar* não faz qualquer alteração a nível deste caracter. Simplesmente processa os parâmetros que recebe.

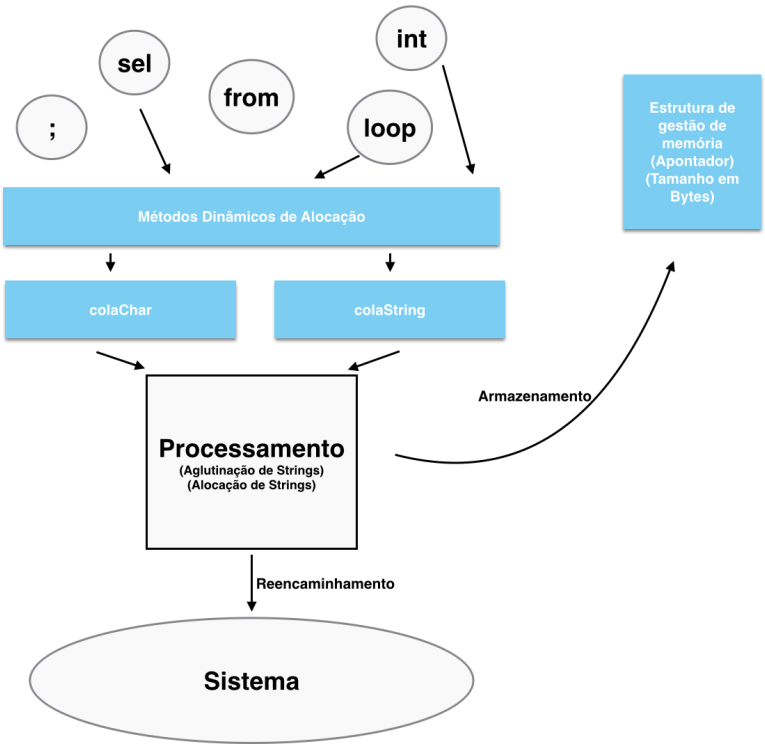


Figura 40: Overview da gestão de memória.

5.4.2 *Gestão de Conhecimento*

A Gestão de Conhecimento, *GC*, é um dos tópicos que potencia a diferenciação entre este tradutor e os que circulam na *internet*. Hoje em dia, a maioria dos tradutores de linguagens de programação limitam-se a traduzir literalmente o que é apresentado. Uma troca de palavras equivalentes entre duas linguagens de programação. A restante informação não é relevante para o auxílio da tradução e no fim é este desleixo que leva o utilizador do tradutor a gastar horas de *debug*. A fim de combater esse problema e de aumentar a persistência e consistência da informação apresentada, o tradutor vai guardando, na unidade de Gestão de Conhecimento, meta-informação de tudo o que é relevante:

- Declaração de *domains*;

- Estrutura de *tables*;
- Estrutura de *procedures*;
- Estrutura de *functions*;
- *Indexes*, *triggers*, entre outros.

Esta informação torna-se crucial aquando de uma representação fiel da arquitetura do *Firebird* e o contexto envolvente. Graças à unidade de Gestão de Conhecimento, é possível:

- Validação de *SQL* - Existência de *tables*, *columns*, *et cetera*;
- Diferenciação ao nível de *procedures* e *functions*;
- Utilização correta dos *domains* de *Firebird*;
- Manipulação de *SQL*.

Na hora de tradução o contexto é importante. Uma instrução *SQL* não é isolada, depende de uma panóplia de conceitos, tais como *tables*, *columns*, entre outros; e é neste isolamento que reside o principal problema. A maioria dos tradutores só se focam em traduzir o que lhes é apresentado. Não sabem o que verdadeiramente estão a traduzir, não verificam se o que está a ser traduzido está correto, se faz sentido.

É importante salientar que esta estrutura fica na íntegra guardada em ficheiros de configuração e disponíveis a qualquer altura para consulta através da própria ferramenta.

5.4.3 *Análise Semântica*

A Análise Semântica - *AS* - está inerente ao conhecimento. Sem conhecimento prévio da linguagem, não é possível realizar uma análise semântica correta. Uma pessoa se não conhecer as palavras e as suas respetivas conotações, nunca vai conseguir interpretar o significado de uma frase e responder de acordo com a mesma. O mesmo aplica-se na tradução, é imperativo um conhecimento prévio do universo ao qual a informação pertence. Esse conhecimento necessário à Análise Semântica advém dos algoritmos da Gestão de Conhecimento. A *GC* possibilita à AS a validação de instruções de *SQL*, a distinção entre *procedures* e *functions*, *et cetera*. Permite ao AS reconhecer, compreender e se necessário corrigir o que lhe é apresentado.

Por exemplo, como já foi demonstrado anteriormente as *functions* de sistema no *Firebird* são quase inexistentes. O utilizador precisa de criar *functions* que por norma já existem noutros motores de base de dados, tais como, processamento e conversão de tipos, *output*, operações aritméticas, entre outras. Neste caso, a unidade de AS em junção com a de GC permite ao tradutor substituir as *functions* de utilizador de *Firebird*, propícias ao erro humano, pelas *functions* de sistema do *Oracle*. Além de

possibilitar a validação de instruções *SQL*, os algoritmos de *AS* e de *GC*, são os responsáveis pela optimização de código ao nível das *functions* de processamento e manipulação das variáveis. Os restantes casos de utilização da unidade Análise Semântica concentram-se:

- Processamento de Cadeias *if-then-else*;
- Utilização do *for-Loop*;
- Criação dos mecanismos necessários para conversão de *procedures* em *functions*;
- Extração da informação necessária para conversão de todo o tipo de objeto do *schema*.

Estes serão descrito e estudados no capítulo seguinte - **Apresentação e Discussão dos Resultados**.

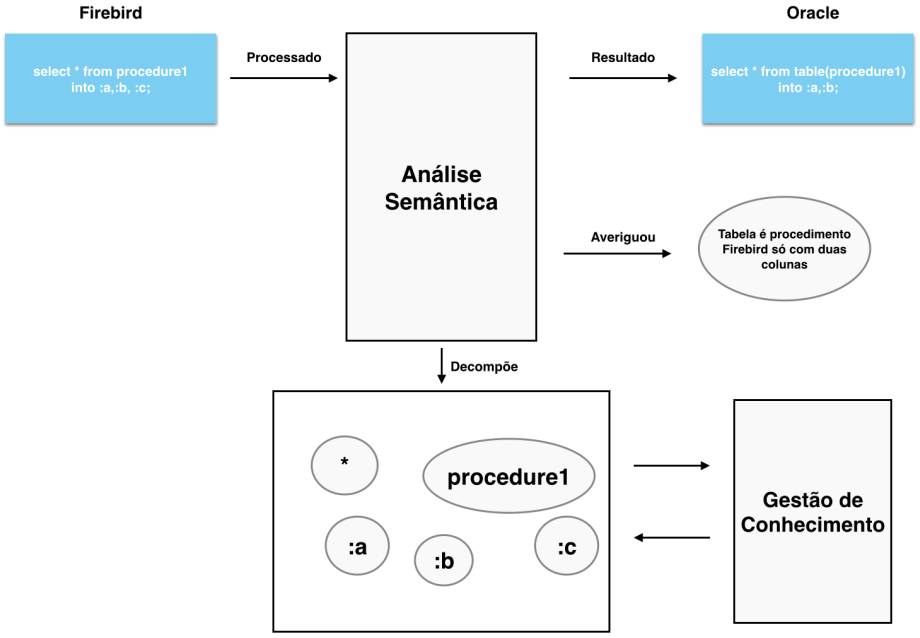


Figura 41: Exemplo simplificado da validação de um *select Firebird*.

Neste exemplo, pode-se verificar, de um modo simplista, como se realiza a validação do *SQL* e a posterior manipulação do *select*. Em primeiro lugar, os algoritmos de *AS*, começam por verificar quais são as *tables* utilizadas na instrução *select*, de seguida, começam a processar cada uma das *columns* de modo a verificar se não há nenhum incoerência. Após a decomposição estar concluída, a unidade de Análise Semântica começa a trocar informação com o núcleo de Gestão de Conhecimento a fim de tirar ilações, através da meta informação guardada,*tables* e respetivas *columns*, a troca e processamento desta informação resultou na descoberta do seguinte:

- A *table* utilizada é um *procedure* de *Firebird*;

- Essa *table* só tem duas *columns*;
- O *into* do *select* tem 3 variáveis declaradas.

Neste caso, esta validação já é suficiente para aferir que a instrução encontra-se inválida. Por defeito, os algoritmos de Análise Semântica verificam se uma das variáveis está declarada, visto que, geralmente, é aí que reside o problema - muito código utilizado no desenvolvimento de *Firebird* resumia-se a *copy & paste*. Porém, se esse não for o caso, o tradutor emite um erro no ficheiro resultante da configuração. No fim, se tudo estiver em conformidade, a Análise Semântica procede às alterações necessárias de modo a corrigir os problemas de validação.

5.4.4 *Tradução*

A tradução só se resume à construção do *output* - a linguagem *PL/SQL* de *Oracle*. E como é que isso é conseguido? Através de estruturas de texto previamente definidas - esqueleto de código Oracle, onde os algoritmos de tradução só preenchem essas estruturas com a informação existente e gerada no *GC* e *AS*.

Esta aproximação permite uma facilidade ao nível da alteração da linguagem de *output*, bem como da correcção de erros que advenha da conversão, já que só basta alterar os próprios esqueletos de texto. Não é necessário estar à procura do método de Análise Semântica ou de Gestão de Conhecimento correspondentes.

Ao nível da leitura e escrita da estrutura *Firebird* para *Oracle*, a tradução é feita com auxílio à leitura e escrita de ficheiros. Do lado do *Firebird*, exporta-se a sua arquitetura, *procedures*, *tables*, todos os objetos do *schema*, para documentos de texto. Aquando da tradução, o utilizador indica tanto o *path* do documento de leitura, bem como o sítio onde deseja escrever o documento que contém a tradução correspondente em *Oracle*.

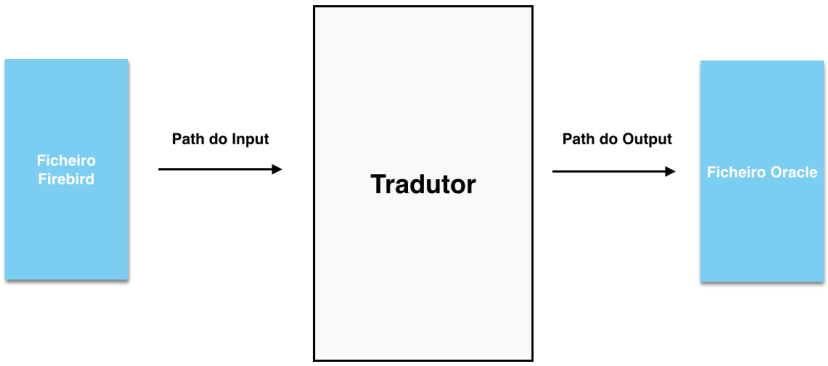


Figura 42: Overview da tradução ao nível dos ficheiros.

6

APRESENTAÇÃO DOS RESULTADOS

A apresentação dos resultados pode ser considerado o epílogo da literatura referente ao tradutor. Pretende-se através de uma apresentação organizada e objetiva de exemplos de conversão - os resultados - a consolidação de todos os conceitos e funcionalidades relativas à ferramenta de tradução. Uma apresentação estruturada no antes e no o após da conversão, não esquecendo a descrição dos procedimentos intrínsecos. O antes corresponde ao código *PL/SQL* de *Firebird*:

- *Domains*;
- *Tables*;
- *Functions*;
- *Procedures*;
- *Triggers*.

Os 'procedimentos intrínsecos' são os métodos utilizados durante a tradução. O após é o resultado da conversão, o equivalente do código *Firebird* em *Oracle*.

Antes de dar ínicio à ilustração dos vários exemplos, é importante aludir à interação do utilizador com a ferramenta de tradução. Esta pode ser feita de duas maneiras:

- Linha de Comandos;
- Menu;

O tempo disponível para o desenvolvimento deste *software* não permitiu um aprimoramento da *interface*. Além disso, o objetivo principal não é a ferramenta de tradução em si, mas sim a conversão, na íntegra, do sistema inteiro de base de dados *Firebird*. A ferramenta é um meio para atingir esse objetivo.

6.1 REQUISITOS

Como já foi referido anteriormente, a Gestão de Conhecimento, ou seja, a contextualização da informação processada, é um dos pontos principais deste tradutor. De modo a aperfeiçoar a conversão,

o tradutor, continuadamente, afere e guarda o contexto da informação que recebe. Ou seja, a relação e contextualização da informação recebida com o Universo do Sistema de Informação baseado em *Firebird*. Por exemplo, à medida que o tradutor processa e converte uma *table* em *Firebird*, este vai criando uma rede de conhecimento para essa mesma *table*:

```
CREATE TABLE CODIGO_PAG_AUX (
    DOC         VARCHAR(50) NOT NULL,
    CAMPO       VARCHAR(50) NOT NULL,
    LINHA       INTEGER NOT NULL,
    TIPO        VARCHAR(10),
    CHAVE       VARCHAR(1),
    ORIGINAL    BLOB SUB_TYPE 1 SEGMENT SIZE 80,
    CORRIGIDO   BLOB SUB_TYPE 1 SEGMENT SIZE 80
);
```

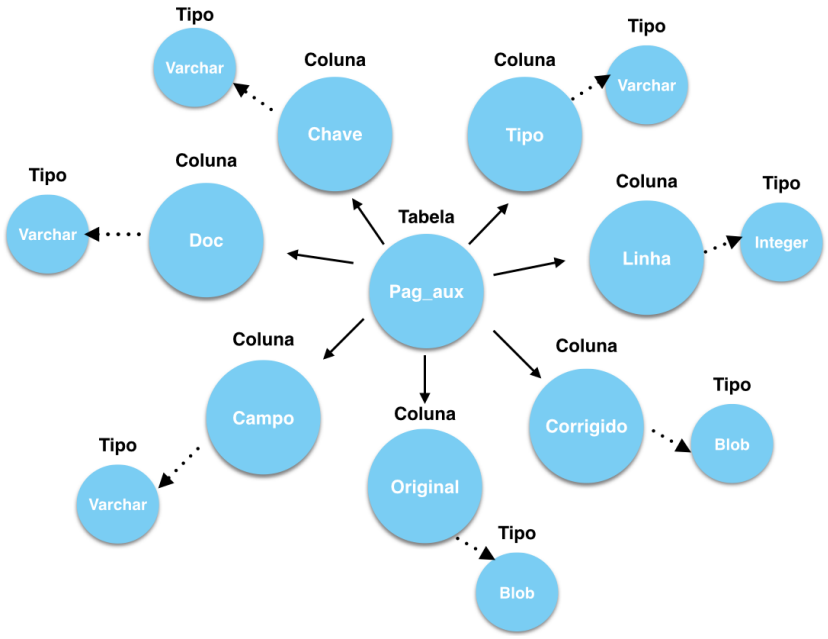


Figura 43: Exemplo de rede de conhecimento.

Esta persistência de meta-informação - uma rede de conhecimento - permite mais tarde que o tradutor possa validar e corrigir instruções de *sql*, dessa *table*, existentes em *procedures*, *functions*, *triggers*, bem como na declaração de *tables*, *views*, *constraints*, *et cetera*.

Devido a esta razão, torna-se imperativo, num primeiro passo, o processamento da estrutura da base de dados, de modo a dotar o compilador do conhecimento principal para uma conversão correta:

- *Tables*;
- *Domains*;
- Outros objetos do *schema*;

6.1.1 *Domains*

Na diferenciação realizada anteriormente, capítulo 4, entre o motor de base de dados *Firebird* e *Oracle*, verificou-se que o conceito de *domain* não existe no *Oracle* e, além disso, a sua utilização no *Firebird* só se resume a um *alias* para um tipo. Assim sendo, o processamento de *domains* só se resume a conferir um maior nível de contexto e conhecimento à unidade de Gestão de Conhecimento.

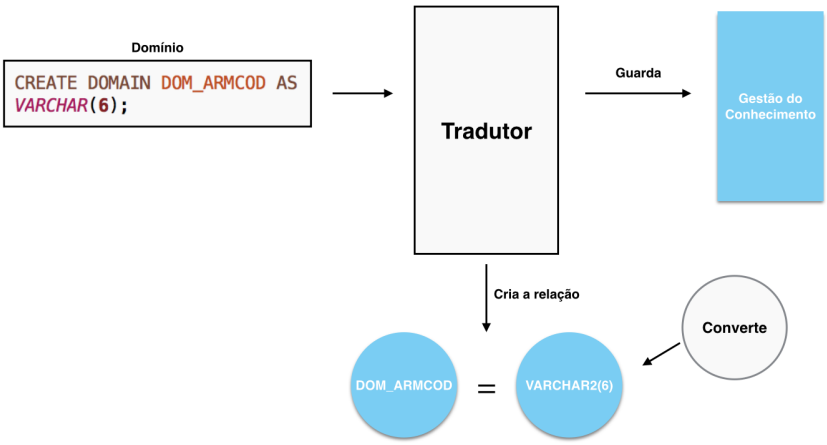


Figura 44: Tradução de *domains*.

A declaração dos *domains* é imutável. Esta figura representa na totalidade os *domains* existentes no *Firebird*, alternando somente o tipo utilizado. O tipo de dados, por sua vez, é convertido para o equivalente em *Oracle* e guardado na Gestão de Conhecimento juntamente com o nome correspondente - *DOM_ARMCOD*. Isto permite ao compilador, substituir sempre que apareça a palavra *DOM_ARMCOD*, pelo seu tipo designado. Na maioria dos compiladores disponíveis no mundo web, não se verifica este tipo de alterações.

6.1.2 *Tables*

Ao contrário dos *domains*, as *tables* já requerem uma tradução de *Firebird* para *Oracle*. Contudo, de modo a se efetuar uma tradução correta e livre de erros, torna-se essencial que o tradutor já tenha um conhecimento básico dos *domains* existentes no Sistema *Firebird*. As *tables* dependem de *domains* - isto já se verificou no capítulo 4 - visto que os *domains* são usados, maioritariamente, na declaração de *tables*.

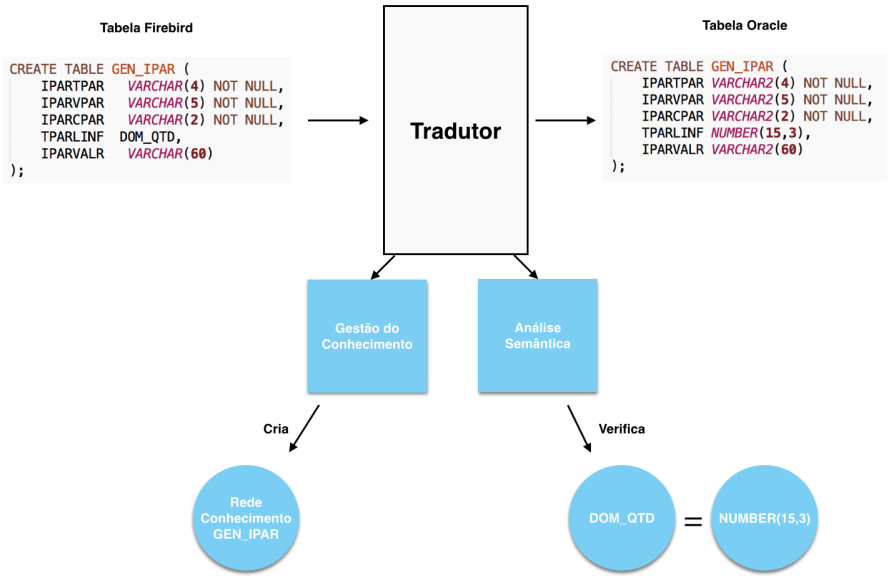


Figura 45: Tradução de *tables*.

A tradução de *tables* de *Firebird* para *Oracle* engloba duas unidades de processamento complementares, a Gestão de Conhecimento e Análise Semântica. Em primeiro lugar, o tradutor começa por analisar a informação recebida, ou seja, por verificar semânticamente se a declaração da *table* em *Firebird* está de acordo com as regras declaradas na gramática, *Yacc*. Durante esse estudo semântico, o compilador descobre que foi utilizada uma palavra atípica - *DOM_QTD*. Prontamente, o tradutor dá início à procura dessa palavra através da Unidade Gestão de Conhecimento e, conclui que esta se trata de um *domain*, procendendo, então, à substituição do *domain* pelo tipo correspondente. Quando o Analisador Semântico constata que já não existe qualquer outro tipo de dúvida ou inconsistência, este dá, então, início à conversão da *table* de *Firebird* para o equivalente em *Oracle*.

Por fim, o tradutor não deixa de efetuar uma representação de meta-informação da *table* - rede do conhecimento - a fim de ser utilizada posteriormente para o aprimoramento do *AS*.

6.1.3 *Outros Objetos do Schema*

Excepto as tabelas, aqui encontra-se as conversões referentes a tudo que esteja relacionado com o esquema da base de dados tais como:

- *Constraints*;
- *Views*;
- *Triggers*;

- *Indexes*;
- *Descriptions*;

Constraints & Indexes

No âmbito das *constraints*, a tradução de *Firebird* para *Oracle* é quase inexistente. A sintaxe usada, como se poderá verificar, é idêntica. No entanto, isto não implica que esta informação não seja processada pelo Tradutor. Ao nível da declaração dos *indexes*, a ferramenta de tradução é utilizada como ferramenta de correcção, ou seja, aquando do processamento dos dados recebidos para conversão, o tradutor verifica:

- A existência de incoerência ao nível da relação entre *columns* e *tables*;
- A nomenclatura utilizada na declaração das *primary* ou *foreign keys*. Por regra, o nome de uma chave está restrito a - *PK_(Nome da table)_KEY*. Caso isto não se verifica o tradutor adiciona ou corrige o nome utilizado.

Por fim, após à análise semântica, o tradutor efetua mais uma vez um mapeamento do conhecimento adquirido na unidade de Gestão de Conhecimento. Já que a informação encontra-se guardada, isto também permite ao utilizador converter sempre que quiser informação de *Firebird* para *Oracle*, sem necessitar de voltar a fornecer a mesma informação.

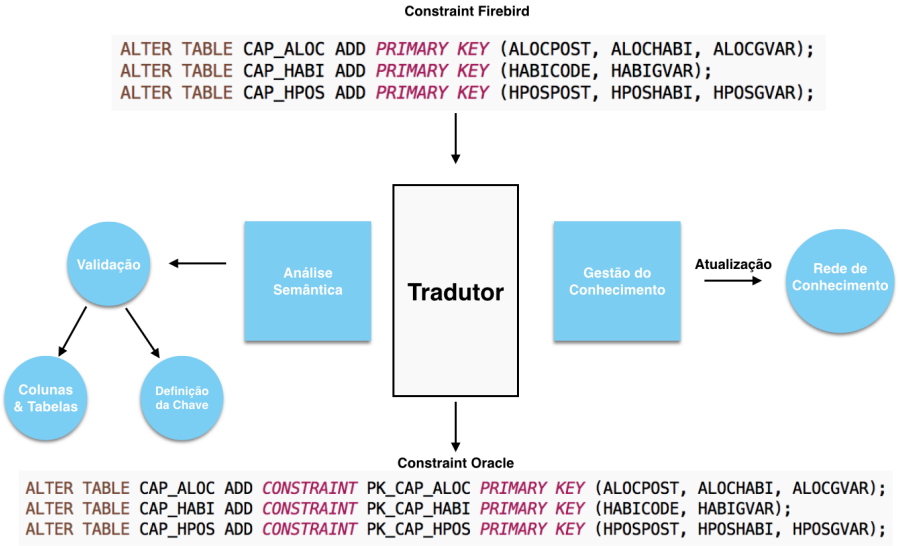


Figura 46: Tradução de *constraints*.

Descriptions

Ao contrário das *constraints* e dos *indexes*, as *descriptions* são transcritas para *Oracle*. No entanto, há uma alteração, embora que pequena, da sintaxe utilizada entre o *Firebird* e *Oracle*. É importante

referir que, mais uma vez, o processamento de nova informação é guardado e mapeado nas redes de conhecimento existentes.

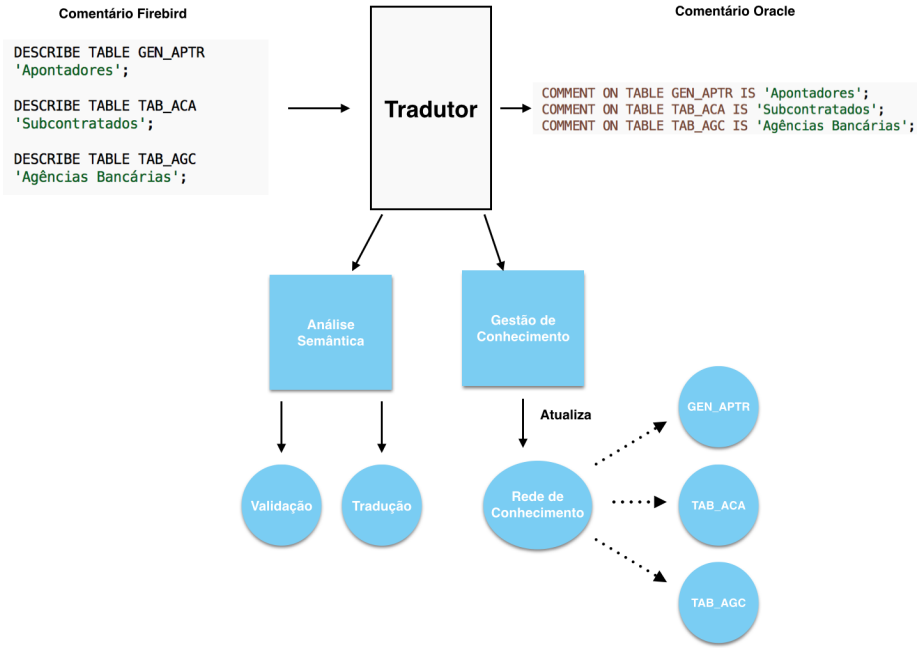


Figura 47: Tradução de *descriptions*.

Views

A conversão de *views* encontra-se no mesmo espectro da declaração de *constraints* e *indexes*. A sintaxe é igual, logo o objetivo do tradutor centra-se na validação da informação recebida:

- Coerência entre *columns*, *tables* e comparação de dados.
- Verificação de erros de sintaxe ao nível das instruções *sql*.

No fim da validação realizada pela unidade de Análise Semântica, a nova informação relativa às *views* é adicionada na Unidade de Gestão de Conhecimento.

Triggers

No caso dos *triggers* já se denota uma disparidade ao nível da sintaxe. Além das diferenças aquando da declaração do *trigger*, as instruções subjacentes ao *trigger* já inclui o *PL/SQL* do *Firebird*. Como já foi referido anteriormente no capítulo 4, o *PL/SQL* utilizado nas dois motores de base de dadores, *Firebird* e *Oracle* são completamente heterogéneos. Isto verifica-se desde uma simples atribuição de um valor a uma variável, até à criação de *loops*.

Em relação ao cabeçalho, a conversão só foi possível após um estudo da arquitetura de um *trigger* em *Firebird* e *Oracle*, de modo, a descobrir ou criar equivalências entre as *flags* utilizadas por ambos.

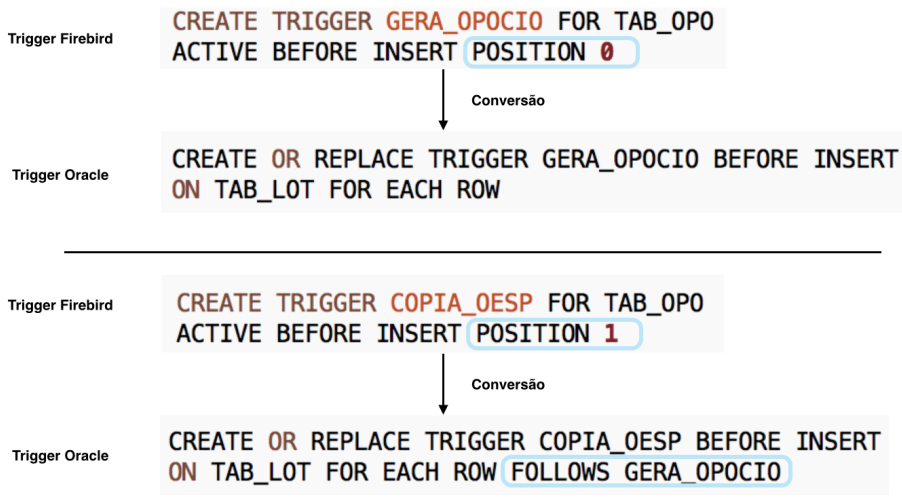


Figura 48: Tradução de cabeçalhos de *triggers*.

Após uma apreciação da figura apresentada, pode-se verificar que a única *flag* que pode dar azo a dúvida é o significado de *position*. As restantes *flags* como *before, active, insert* facilmente se conseguem representar, ou inferir o seu significado num cabeçalho de um *trigger* em *Oracle*, visto que em muitos casos até partilham o mesmo nome. Porém, não existe a *flag position* no *Oracle*, mas graças ao estudo realizado da arquitetura do *Firebird* permitiu-se concluir que esta *flag* corresponde ao nível de prioridade da execução de *triggers*. A ordem pela qual os *triggers* de uma *table* são executados. E como é que o tradutor pode resolver o problema da ordem da execução já que no *Firebird* só tem a indicação da posição?

Como já se sabe, a unidade de Gestão de Conhecimento é a responsável por armazenar e criar uma relação entre os vários conceitos e definições processadas pelo tradutor. Assim sendo, facilmente o tradutor pode, aquando da análise semântica, aferir qual o *trigger* precedente.

6.1. Requisitos

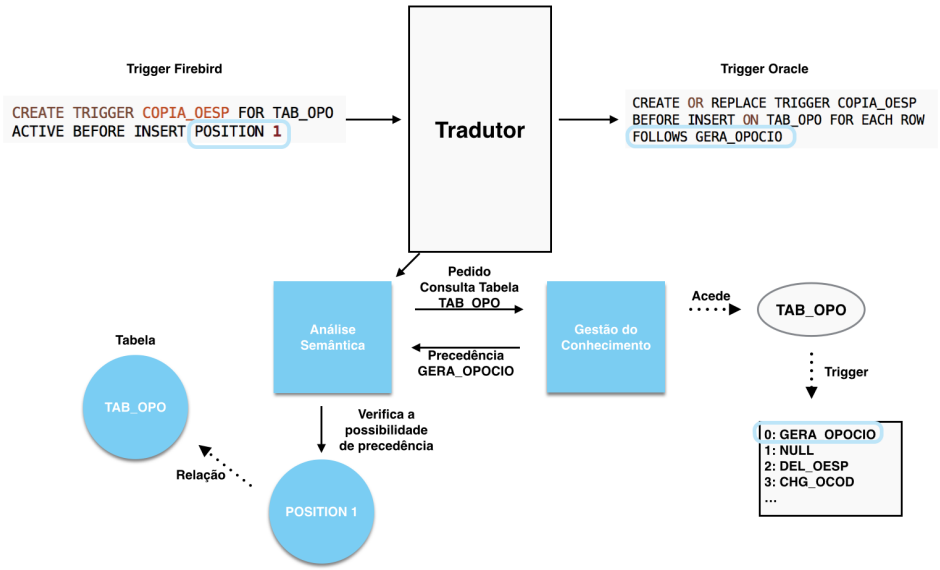


Figura 49: Análise Semântica do cabeçalho de um *trigger*.

Completado a análise semântica do cabeçalho, o tradutor dá início à tradução do restante corpo do *trigger* - o código *PL/SQL*. A figura que se segue tem como objetivo elucidar procedimentos de análise semântica que ainda não foram mencionados.

Trigger Firebird

```
CREATE TRIGGER COPIA_OESP FOR TAB_OPO
ACTIVE BEFORE INSERT POSITION 1
AS
DECLARE VARIABLE ORD INTEGER;
DECLARE VARIABLE PLNO VARCHAR(5);
DECLARE VARIABLE PRG VARCHAR(3);
DECLARE VARIABLE TIPO VARCHAR(1);
DECLARE VARIABLE OESP VARCHAR(25);
BEGIN
    ORD=NEW.OPOORD;

    SELECT ORDPRG FROM TAB_ORD WHERE ORDCOD=:ORD INTO :PRG;
    IF (PRG='SIM') THEN
    BEGIN
        SELECT OESP FROM ATRIBUI_OESP(:ORD, NEW.OPOCOD) INTO NEW.OPOOESP;
    END
END
```

Figura 50: Análise Semântica inicial do corpo de um *trigger*.

Aquando da elaboração de um *trigger*, pode-se aceder em *real-time* a valores antigos ou recentes de *columns*, *OLD.'column'* e *NEW.'column'* respetivamente, da *table* definida no cabeçalho do *trigger*.

Então, como qualquer outra variável declarada, o analisador semântico tenta garantir que não haja incoerências nas relações de *column-table*. Ademais, o analisador semântico também verifica seg as variáveis utilizadas no *trigger* estão previamente declaradas. Por fim, o último tópico prende-se com a tal prenunciada utilização, por parte do *Firebird*, de *procedures* como *tables* - isto verifica-se facilmente na cláusula do *from* da ultima instrução de *select*.

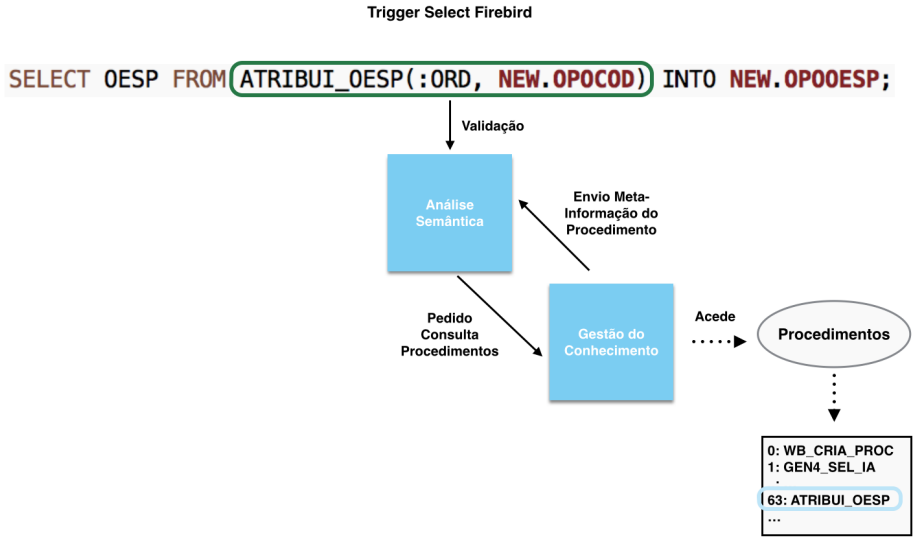


Figura 51: Análise Semântica da invocação de um *procedure* num *trigger*.

Após a recepção da meta-informação referente ao *procedure* invocado, o analisador verifica mais uma vez se não há qualquer tipo disparidade ao nível do número de parâmetros e dos seus respetivos tipos. Quando o analisador dá o trabalho por concluído, a unidade de Gestão de Conhecimento efetua, mais uma vez, uma atualização da rede de conhecimento, adicionando a meta-informação do *trigger* à *table* em questão. No fim de tudo, procede-se à tradução do próprio *trigger*.

```
CREATE OR REPLACE TRIGGER COPIA_OESP
BEFORE INSERT ON TAB_OPO FOR EACH ROW
FOLLOWS GERA_OPOCIO
IS
    ORD NUMBER;
    PLNO VARCHAR2(5);
    PRG VARCHAR2(3);
    TIPO VARCHAR2(1);
    OESP VARCHAR2(25);
begin
    ORD:=NEW.OPOORD;
    select ORDPRG INTO PRG from TAB_ORD WHERE ORDCOD=ORD;
    if ( PRG='SIM' )
    then
        begin
            select OESP INTO NEW.OPOOESP from table(ATRIBUI_OESP(ORD, NEW.OPOCOD));
        end;
    end if;
end COPIA_OESP;
```

Figura 52: Tradução do *trigger*.

6.1.4 *Procedures & Functions*

Procedures e *functions* são relativamente semelhantes. Do ponto de vista estrutural e do trabalho realizado pelo Tradutor não se denota alterações, visto que um *procedure Firebird*, na maioria das vezes, será traduzido para uma *function Oracle*. Assim, pode-se agrupar os *procedures* e *functions* na mesma secção. É importante também ressalvar que a tradução exemplificada só será referente a um *procedure Firebird*. A conversão dum *procedure Firebird* além de englobar os métodos utilizados pela Unidade de Gestão de Conhecimento e Análise Semântica aquando da tradução de uma *function*, também é mais completa e abrange mais tópicos e procedimentos,

Deste jeito, e dando início à exemplificação da tradução de um *procedure*, pode-se começar por dizer que um *procedure* está dividido em duas partes:

- cabeçalho;
- corpo.

Como foi referido anteriomente no capítulo 4, os *procedures* em *Firebird* podem ou não ser utilizados como *tables*. Como é que se sabe isso? Ao visualizar o cabeçalho que se segue, podemos deduzir, de uma forma leiga, que qualquer *procedure* que retorne um ou mais parâmetros será utilizado como *table*. E como é que se insere dados nessa '*table*' em *Firebird*? Através do comando *suspend;*. Este insere na *table* os valores atuais das variáveis a retornar aquando da execução da instrução.

Procedimento Firebird

```
CREATE OR ALTER PROCEDURE GEN4_CAMPOS (
    TIPO VARCHAR(20),
    ENTIDADE VARCHAR(15))
RETURNS (
    RELACODE VARCHAR(3),
    RELADESC VARCHAR(30),
    ENTDCODE VARCHAR(15),
    ATRIBUTO VARCHAR(15),
    ATRBDESC VARCHAR(40))
```

Figura 53: Cabeçalho de *procedure Firebird* a traduzir.

Outro aspeto relevante a reter, é que o *Oracle* não permite a utilização de qualquer tipo de *procedures* como *tables*. Todavia, fruto do estudo do motor de base de dados *Oracle*, foi possível a imitar o comportamento de *procedures Firebird* em Oracle, através da utilização de objetos, tipos e *functions*.

Primeiro, cria-se um objeto que permita englobar os parâmetros de *return* declarados no *Firebird*. Abstratamente, isto pode ser caracterizado por uma *row*, como se pode verificar na figura que se segue. O objeto é equivalente à cláusula *return* do *procedure*.

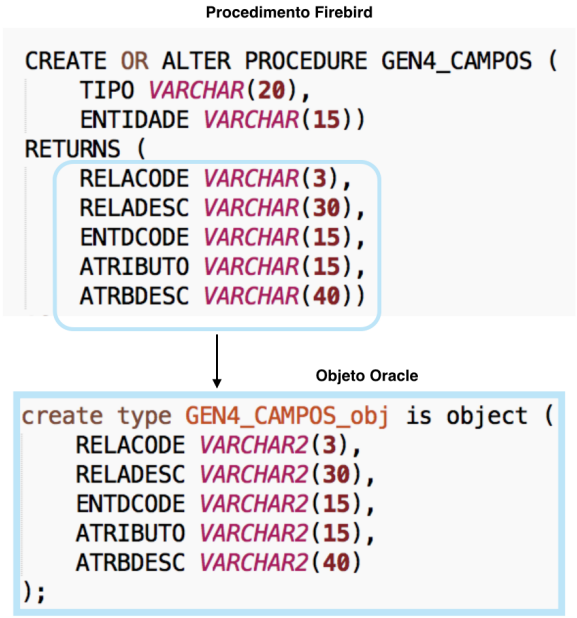


Figura 54: Criação do objeto *row* no *Oracle*.

Em seguida, utiliza-se este objeto recentemente criado, num objeto de tipo *table*. O âmago deste objeto não pode ser considerado verdadeiramente uma *table*, quanto muito, é só uma *column* de objetos. No fim, são os objetos da *column* que conferem a tal *row*, conseguindo assim uma aproximação às *tables* verdadeiras do *Oracle*.

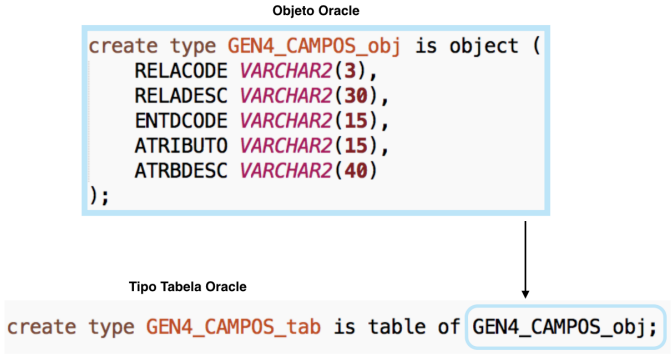


Figura 55: Criação da '*table*' do *procedure Firebird* em *Oracle*.

Só após a criação destes dois tipos de objetos, fundamentais para a imitação do comportamento de um *procedure Firebird* em *Oracle*, é que se dá início à conversão do *procedure Firebird* numa *function Oracle*. E porquê? Ao contrário das *functions Oracle*, os *procedures* não permitem retornar qualquer tipo de variável, não tem cláusula de *return*. O *return* nos *procedures* do *Oracle* é feito através dos dois tipos de variáveis de parâmetros - *IN & OUT* - em que o valor das variáveis com a *flag OUT* é

acessível fora do *scope* desse *procedure*. Porém, isto não permitia replicar o mesmo comportamento verificado no *Firebird*. É aqui que a conversão de *procedure Firebird* para *function Oracle* torna-se viável, visto que a última permite retornar somente uma e uma só variável. O *return* é o objeto do tipo *table*.

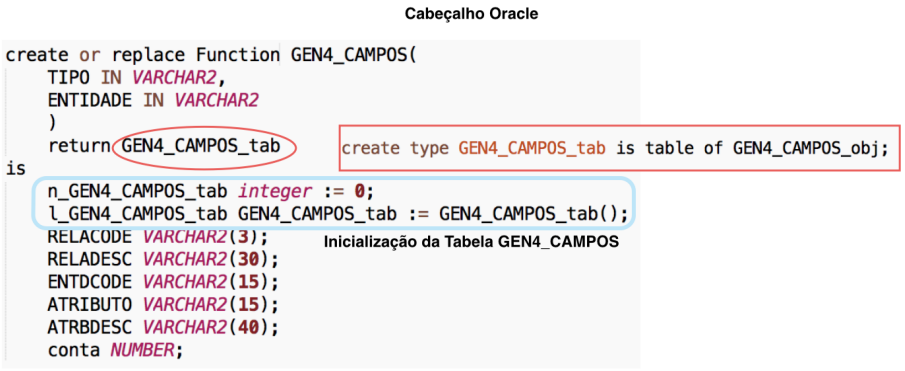


Figura 56: Tradução do cabeçalho do *procedure*.

Esta figura consiste na conversão do cabeçalho do *procedure Firebird* previamente demonstrado. Ainda assim é preciso salientar outro detalhe. O objeto *table* que será retornado é, como já foi referido, no seu cerne, uma *column* de objetos. Não tem as mesmas 'regalias' de uma *table* de sistema. Não tem qualquer tipo de manutenção automática ou acesso a comandos como *insert, update* ou *delete*. Analogamente o objeto *GEN4_campos_TAB* pode ser equiparado a um *array*. Tal como num array em *C*, torna-se necessário a existência de um contador/apontador para ajudar na manutenção da *table*, em casos de inserir, procurar e remover. No âmbito do *Firebird*, a manutenção da *table* do *procedure*, mais concretamente o *insert*, é realizada através do comando *suspend*. A figura que se segue acaba por mostrar o equivalente em *Oracle*, bem como as instruções a realizar a fim de garantir um preenchimento correta do objeto *table*.

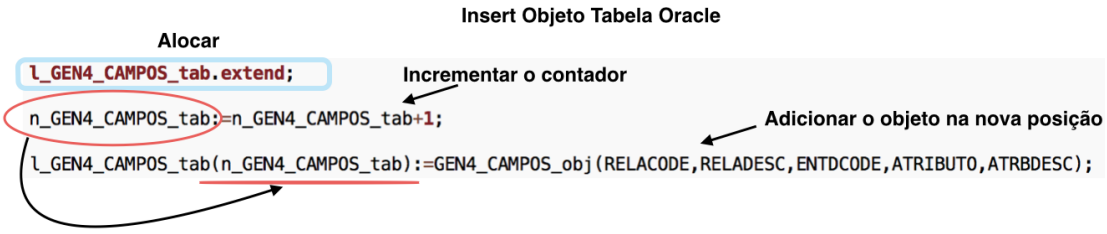


Figura 57: Equivalente em *Oracle* à instrução *suspend* de *Firebird*.

Continuando no espetro da tradução. Após a conversão do cabeçalho, o analisador semântico começa a focar-se na validação e tradução do corpo. Antes de dar início à análise do corpo, a unidade

de Gestão de Conhecimento utiliza a meta-informação recolhida para criar uma rede de conhecimento deste *procedure*. Esta informação torna-se, mais tarde, útil para:

- Distinguir instruções *select* a *tables* de sistema ou *functions Oracle*.
- Distinguir entre os *procedures* de *Firebird* que serão convertidos em *procedures Oracle* ou *functions Oracle*.
- Facultar as transformações necessárias a realizar ao nível da invocação e utilização de *procedures* e *functions Oracle*.

Em relação ao corpo do *procedure Firebird*, a principal preocupação concentra-se na estruturação dos ciclos. Este é outro aspeto diferenciador entre os dois motores de base de dados. O *Firebird* por si próprio, facilmente, consegue criar relação entre os valores da *column* que se quer obter e as respetivas variáveis onde se pretende salvaguardar o valor. O mesmo não se verifica em certos casos no motor *Oracle*, mais concretamente *sql* dinâmico.

```
for select 'T'||gen4_rela.relacode,gen4_rela.reladesc1,gen4_entd.entdcode,gen4_atrb.atrbcode,
gen4_atrb.atrbdesc from gen4_rela
join gen4_entd on gen4_entd.entdcode=gen4_rela.relaent1
join gen4_atrb on gen4_atrb.atrbtabl=gen4_entd.entdtabl and gen4_atrb.atrbsist='N' and gen4_atrb.atrbio='OUTPUT'
where gen4_rela.relaentm=:entidade
order by gen4_rela.relacode
into :relacode,:reladesc,:entdcode,:atributo,:atrbdesc
```

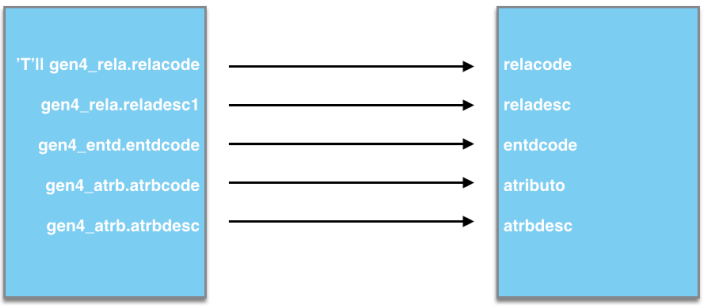


Figura 58: Relação entre *columns* e variáveis *into* - *Loop Firebird*.

Um dos maiores obstáculos, no desenvolvimento do tradutor, prendeu-se com a interpretação do *sql* dinâmico por parte do *Oracle*, principalmente nas estruturas cíclicas. A facilidade com que o *Firebird* infere as relações entre as *columns* dinâmicas e a cláusula de *into*, mesmo com *sql* dinâmico, não se verifica no *Oracle*. No domínio do *Oracle*, estas relações através do *sql* dinâmico, como no caso da primeira *column*, só são possíveis de serem implementadas através de certas alterações na estrutura *for* do *Oracle*.

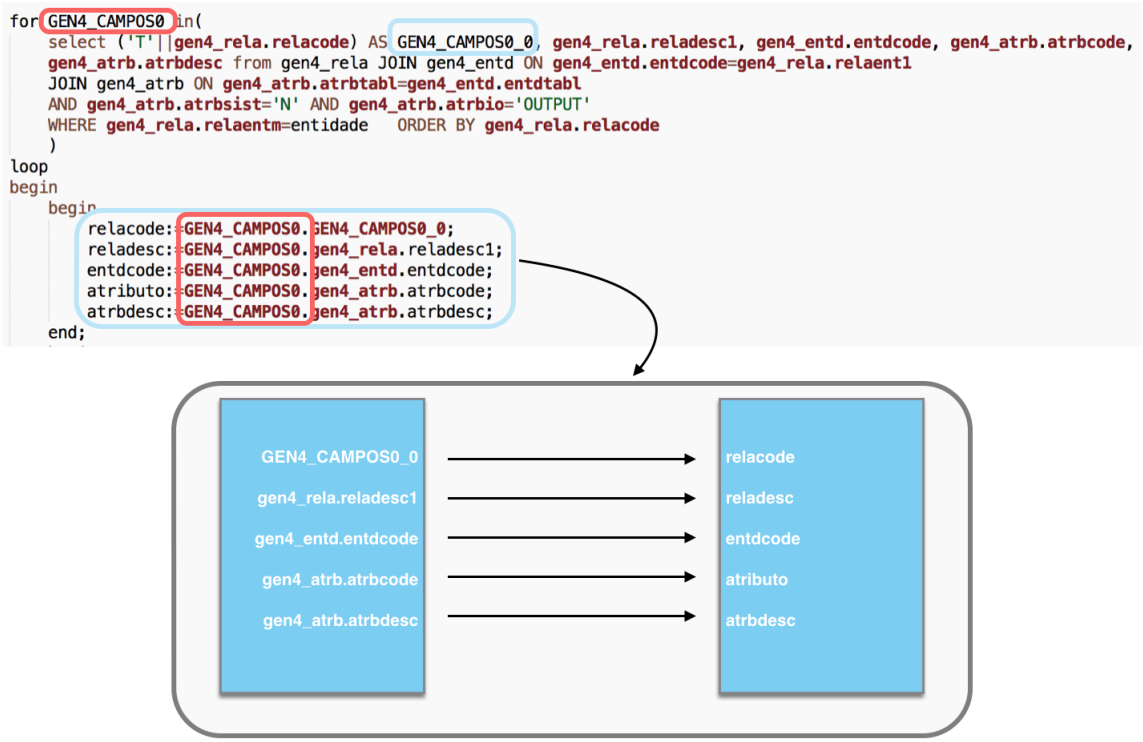


Figura 59: Conversão da estrutura *For* em *Oracle*.

Como se pode verificar na figura 59, a solução passa pela descrição explícita da relação entre a cláusula *into* e as *columns* da instrução *select*. E como é que se descreve uma relação correta entre o *sql* dinâmico e uma variável? De modo a que não haja incoerências ou erros durante a compilação do código *Oracle PL/SQL*, é necessário a criação de um *alias* que englobe a totalidade do sql dinâmico referente à *column*. À medida que a unidade de análise semântica vai processando a instrução *select*, o analisador verifica a existência de sql dinâmico e, caso encontre, o analisador procede à geração de uma *tag* única para a respetiva *column*. Todavia, caso encontre um *alias* já previamente definido, este é guardado para posterior utilização no bloco correspondente à declaração de relações.

Até aqui não parece que venha haver qualquer tipo de incoerência ou conflito entre a relações dos dados num ciclo *for*, visto que cada *column* dinâmica está associada a um id único. Contudo, o caso muda de figura quando se encontra ciclos dentro de ciclos, uma vez que pode haver repetições de instruções *select*, originando, assim, problemas de inconsistência e persistência de dados. A solução traduz-se na criação de uma *tag* pai associada a um único *for*. Assim sendo, a junção destas duas *tags* é mais do que suficiente para proibir o cruzamento, não intencional, de dados entre os vários ciclos. É importante ressalvar que o analisador semântico é o responsável por gerar e garantir que as *tags* associadas a cada *for* ou *sql* dinâmico são únicas.

Função Oracle

```
begin
for GEN4_CAMPOS0 in(select gen4_rela.relacode, gen4_rela.reladesc1, gen4_entd.entdcode, gen4_atrb.atrbcode,
    gen4_atrb.atrbdesc from gen4_rela JOIN gen4_entd ON gen4_entd.entdcode=gen4_rela.relaent1
    JOIN gen4_atrb ON gen4_atrb.atrbtabl=gen4_entd.entdtabl
    AND gen4_atrb.atrbsist='N' AND gen4_atrb.atrbio='OUTPUT'
    WHERE gen4_rela.relaentm=entidade   ORDER BY gen4_rela.relacode)
loop
begin
    begin
        relacode:=GEN4_CAMPOS0.gen4_rela.relacode;
        reladesc:=GEN4_CAMPOS0.gen4_rela.reladesc1;
        entdcode:=GEN4_CAMPOS0.gen4_entd.entdcode;
        atributo:=GEN4_CAMPOS0.gen4_atrb.atrbcode;
        atrbdesc:=GEN4_CAMPOS0.gen4_atrb.atrbdesc;
    end;
    begin
        conta:=0;
        select COUNT(*) INTO conta from gen4_cjoi WHERE gen4_cjoi.cjoientm=entidade AND gen4_cjoi.cjoiatrm=atributo;
        if  ( tipo='DETALHE' AND conta=0 )
        then
            l_GEN4_CAMPOS_tab.extend;
            n_GEN4_CAMPOS_tab:=n_GEN4_CAMPOS_tab+1;
            l_GEN4_CAMPOS_tab(n_GEN4_CAMPOS_tab):=GEN4_CAMPOS_obj(RELACODE,RELADESC,ENTDCODE,ATRIBUTO,ATRBDESC);
        end if;
        conta:=0;
        select COUNT(*) INTO conta from gen4_cjoi WHERE gen4_cjoi.cjoirela=relacode AND gen4_cjoi.cjoientm=entidade
        AND gen4_cjoi.cjoiatr1=atributo;
        if  ( tipo='SEL' AND conta=0 )
        then
            l_GEN4_CAMPOS_tab.extend;
            n_GEN4_CAMPOS_tab:=n_GEN4_CAMPOS_tab+1;
            l_GEN4_CAMPOS_tab(n_GEN4_CAMPOS_tab):=GEN4_CAMPOS_obj(RELACODE,RELADESC,ENTDCODE,ATRIBUTO,ATRBDESC);
        end if;
    end;
end;
end loop;
return l_GEN4_CAMPOS_tab;
end;
```

Figura 60: Resultado da tradução do *procedure Firebird*.

No final da tradução, a unidade de Análise Semântica verifica se o resultado é uma *function* ou *procedure* em Oracle. Caso seja *function*, a última instrução - *return* - é adicionada. No caso do *procedure*, além da instrução *return*, as restantes operações limitadas por quadrados não existiriam, dado que os objetos *tables* só são permitidos no uso de *functions*.

7

CONCLUSÃO

A eficiência, a diversificação, a interoperabilidade, o compromisso e a curiosidade representam, em toda a sua plenitude, a criação e o percurso percorrido ao longo desta dissertação. Antes de mais, a parceria entre a Universidade do Minho e a *Bosch* tem como mote a criação de novas soluções em torno do ciclo de desenvolvimento e produção de soluções de multimédia avançadas para automóveis. Mais concretamente e no âmbito desta Bolsa de Investigação, o objetivo principal concentrava-se no desenvolvimento de uma ferramenta, a *SMART Manufacturing Control*. Uma ferramenta que potencie os níveis de **eficiência** e de controlo de qualidade dos processos nas linhas de inserção automática. Como é que isso é possível?

Através do acoplamento dos vários sistemas de informação da *Bosch* numa só plataforma, num só sistema. Aumentando assim, a organização, a persistência de dados e a possibilidade de extração de conhecimento - daí a plataforma denominar-se de um Sistema de Gestão Inteligente. Porém, na *Bosch* existe um grande **diversificação** de componentes, de *software* e de sistemas de informação, criados e desenvolvidos por entidades diferentes. Tendo como pano de fundo, a sociedade atual e a conjetura tecnológica em que nos encontramos - um mundo de alta competitividade - pode-se facilmente verificar que estes *softwares* não comunicam entre si, estão isolados. A **Interoperabilidade** existente, no Universo *Bosch*, resume-se na maioria dos casos ao processamento, extração e comunicação manual.

A *Bosch* não é a única. Hoje em dia, ou as empresas dependentes da tecnologia criam os seus próprios sistemas informáticos, o que leva a um investimento muito avultado, ou acabam por depender de terceiros, isto é o que geralmente acontece. Daí a **Interoperabilidade** estar cada vez mais a ganhar relevo e importância na nossa sociedade. Todavia a diversificação não é o único problema. De modo a facilitar o processamento de informação por parte da plataforma *SMART Manufacturing Control*, esta tem que guardar a informação toda num só sistema de informação. O acoplar explicitado anteriormente, refere-se ao ato de assimilar todos os dados num só lugar, numa só base de dados, e, é aqui que reside o segundo problema. Esta plataforma já está em desenvolvimento há mais de 5 anos, já está assente numa base de dados *Firebird*. No entanto, verificou-se, numa fase de testes, que o *Firebird* não é suficiente para suportar, nem para processar a quantidade colossal de informação que é gerada durante o funcionamento das linhas de produção automática. Mais tarde, após o levantamento do Sistema Informático da Unidade de Produção da *Bosch*, constatou-se que um motor de base de dados *Oracle* seria suficiente para responder com a eficácia pretendida. Após uma análise e estudo

das opções disponíveis, o **compromisso** necessário para resolver o obstáculo final que se apresentara, passou pela replicação do sistema de informação *Firebird* num sistema *Oracle*. O resultante do compromisso é um compilador de linguagens de alto nível - tradutor - inteligente e modular. Inteligente na medida em que, não se resume a retornar o equivalente noutra linguagem. O Tradutor guarda e processa toda a informação, verifica as relações entre os vários conjuntos de dados, e só após esse processamento é que procede a uma tradução fiel da base de dados *Firebird* em *Oracle*. A modularidade está relacionado com os princípios seguidos aquando do seu desenvolvimento. Como se está a guardar toda a informação no *software*, a conversão para outros tipos de motor de base de dados só se resume à alteração dos métodos de *print*.

Por fim, o que levou à culminação desta dissertação, do desenvolvimento do tradutor foi a pura **curiosidade** de saber mais, de aprender novas tecnologias, processamento de linguagens, uso de expressões regulares e de tentar completar o objetivo, sem bases prévias, de uma forma inteligente, prática e rápida, visto que o tempo não era suficiente para a quantidade de *procedures*, *tables*, *triggers* e restante informação necessitada de conversão. *Stay Hungry, Stay Foolish*.

8

TRABALHO A REALIZAR

Anteriormente à bolsa de investigação, o conhecimento de processamento de linguages era nulo, inexistente. Restringia-se a uma cadeira semestral do 3º ano. Agora, descoberto o universo do processamento de linguagens, compiladores e tradução, esta dissertação só pode ser denotada como o início. O desenvolvimento desta ferramenta serviu de entrada nesse Universo. Porém, devido ao conhecimento prévio inexistente, ao tempo gasto na auto-aprendizagem e ao tempo disponível para desenvolver o tradutor, ainda se pode encontrar muitas áreas de aprimoramento no âmbito do Tradutor. Estas áreas podem ser circunscritas pela:

- Automação da Tradução;
- Utilização de novas ferramentas;
- Conversão de novas linguagens;

A automação da tradução consiste no ato de converter automaticamente, sem qualquer tipo de supervisão humana, uma base de dados *Firebird* para *Oracle*. Do ponto de vista técnico, já se efectuou um levantamento da temática e verificou-se que é possível. O único entrave são as bibliotecas de acesso a base de dados *Firebird* para *C*. No âmbito do *Oracle* não se aponta qualquer tipo de problema.

A utilização de novas ferramentas vai de encontro à auto-aprendizagem e ao conhecimento prévio. Este último só permitiu a utilização de ferramentas inseridas no meio educativo do que no profissional. Como o tempo disponível também não era suficiente para aprender de raíz novos paradigmas e ferramentas de processamento de linguagens, optou-se pela utilização do *Yacc* e *Flex*.

Graças à facilidade com que se pode alterar o modo como o *output* é escrito - sintaxe da linguagem. Um dos focos vai passar pela disponibilização de mais possibilidades de escolha da linguagem traduzida. Numa primeira fase, será o *MySQL*. Uma análise breve permite concluir que tem muitas similaridades com a arquitetura do *Oracle*. Logo o desenvolvimento dos novos esqueletos de texto, algorimtos de tradução não devem impor muitas dificuldades.

BIBLIOGRAFIA

[1] J. A. O'Brien, "Introduction to inform systems-essential for the e-business enterprise," *America: McGraw Hill*, 2003.

[2] F. Marins, L. Cardoso, F. Portela, M. F. Santos, A. Abelha, and J. Machado, "Improving high availability and reliability of health interoperability systems," in *New Perspectives in Information Systems and Technologies, Volume 2*, pp. 207–216, Springer, 2014.

[3] O. Iroju, A. Soriyan, and I. Gambo, "Ontology matching: An ultimate solution for semantic interoperability in healthcare," *International Journal of Computer Applications*, vol. 51, no. 21, pp. 7–14, 2012.

[4] H. Kubicek, R. Cimander, and H. J. Scholl, *Organizational interoperability in e-government: lessons from 77 European good-practice cases*. Springer Science & Business Media, 2011.

[5] W. Stallings, *Handbook of computer-communications standards; Vol. 1: the open systems interconnection (OSI) model and OSI-related standards*. Macmillan Publishing Co., Inc., 1987.

[6] M. Miranda, J. Machado, A. Abelha, and J. Neves, "Healthcare interoperability through a jade based multi-agent platform," in *Intelligent Distributed Computing VI*, pp. 83–88, Springer, 2013.

[7] L. Cardoso, F. Marins, F. Portela, M. Santos, A. Abelha, and J. Machado, "A multi-agent platform for hospital interoperability," in *Ambient Intelligence-Software and Applications*, pp. 127–134, Springer, 2014.

[8] M. Miranda, M. Salazar, F. Portela, M. Santos, A. Abelha, J. Neves, and J. Machado, "Multi-agent systems for hl7 interoperability services," *Procedia Technology*, vol. 5, pp. 725–733, 2012.

[9] Å. Dragland, "Big data–for better or worse," *SINTEF, retrieved on July*, vol. 22, 2013.

[10] H. Peixoto, M. Santos, A. Abelha, and J. Machado, "Intelligence in interoperability with aida," in *Foundations of Intelligent Systems*, pp. 264–273, Springer, 2012.

[11] P. M. T. Tudorache, A. B. C. Welty, C. K. D. Vrandecic, P. G. N. Noy, and K. J. C. Goble, *The Semantic Web – ISWC 2014*. Springer, 2014. ISBN 978-3-319-11964-9.

[12] F. Alves, A. Abelha, J. M. Machado, J. Neves, and J. Neves, "Interoperability performance in a healthcare environment," 2010.

[13] H. Peixoto, J. M. Machado, and A. Abelha, "Interoperabilidade e o processo clínico semântico," 2010.

[14] J. T. Pollock and R. Hodgson, *Adaptive information: Improving business through semantic interoperability, grid computing, and enterprise integration*, vol. 50. John Wiley & Sons, 2004.

[15] J. Walker, E. Pan, D. Johnston, J. Adler-Milstein, D. W. Bates, and B. Middleton, "The value of health care information exchange and interoperability," *HEALTH AFFAIRS-MILLWOOD VA THEN BETHESDA MA-*, vol. 24, p. W5, 2005.

[16] T. Berners-Lee, J. Hendler, O. Lassila, *et al.*, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[17] W. W. W. Consortium *et al.*, "Rdf 1.1 concepts and abstract syntax," 2014.

[18] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," 2009.

[19] J. A. Marques, A. J. G. Correia, L. Cerqueira, J. M. Machado, and J. Neves, "Archetype-based semantic interoperability in healthcare," 2010.

[20] W. W. W. Consortium *et al.*, "Rdf 1.1 semantics," 2014.

[21] R. Angles and C. Gutierrez, "Querying rdf data from a graph database perspective," in *The Semantic Web: Research and Applications*, pp. 346–360, Springer, 2005.

[22] D. Brickley and R. V. Guha, "Resource description framework (rdf) schema specification 1.0: W3c candidate recommendation 27 march 2000," 2000.

[23] D. Brickley and R. V. Guha, "Rdf schema 1.1," *W3C Recommendation*, 2004.

[24] W. W. W. Consortium *et al.*, "Owl 2 web ontology language document overview," 2012.

[25] L. Ding, L. Zhou, T. Finin, and A. Joshi, "How the semantic web is being used: An analysis of foaf documents," in *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pp. 113c–113c, IEEE, 2005.

[26] A. Powell, M. Nilsson, A. Naeve, P. Johnston, T. Baker, D. C. M. Initiative, *et al.*, "Dcmi abstract model," 2007.

[27] D. Brickley, "Using rdfa 1.1 lite with schema. org," *schema. org*, vol. 11, 2011.

[28] D. Allemang and J. Hendler, *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

[29] A. Miles, B. Matthews, M. Wilson, and D. Brickley, "Skos core: simple knowledge organisation for the web," in *International Conference on Dublin Core and Metadata Applications*, pp. pp–3, 2005.

[30] W. W. W. Consortium *et al.*, "Json-ld 1.0: a json-based serialization for linked data," 2014.

[31] B. Adida and M. Birbeck, "Rdfa primer 1.0: Embedding rdf in xhtml," *W3C working draft*, vol. 12, 2007.

[32] D. Beckett and B. McBride, "Rdf/xml syntax specification (revised)," *W3C recommendation*, vol. 10, 2004.

[33] W. S. W. Group *et al.*, "Sparql 1.1 overview," *World Wide Web Consortium, Recommendation REC-sparql11-overview-20130321*, 2013.

[34] J. M. Machado, M. Miranda, P. Gonçalves, A. Abelha, J. Neves, and J. A. Marques, "Aidatrace: interoperation platform for active monitoring in healthcare environments," 2010.

[35] M. Miranda, G. Pontes, A. Abelha, J. Neves, and J. Machado, "Agent based interoperability in hospital information systems," in *Biomedical Engineering and Informatics (BMEI), 2012 5th International Conference on*, pp. 949–953, IEEE, 2012.

[36] M. Miranda, J. Machado, A. Abelha, J. Neves, and J. Neves, "Evolutionary intelligence in agent modeling and interoperability," in *Ambient Intelligence-Software and Applications*, pp. 253–257, Springer, 2011.

[37] L. Cardoso, F. Marins, F. Portela, M. Santos, A. Abelha, and J. Machado, "The next generation of interoperability agents in healthcare," *International Journal of Environmental Research and Public Health*, vol. 11, no. 5, p. 5349, 2014.

[38] L. Cardoso, F. Marins, F. Portela, A. Abelha, and J. Machado, "Healthcare interoperability through intelligent agent technology," *Procedia Technology*, vol. 16, pp. 1334–1341, 2014.

[39] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques*. Addison wesley, 1986.

[40] S. Kumar, "Coms w4115 programming languages and translators fall 2006 prof. stephen edwards," 2006.

[41] K. Kennedy and J. R. Allen, "Optimizing compilers for modern architectures: a dependence-based approach," 2001.

[42] P. Beynon-Davies, *Database systems*. Macmillan, 2000.

[43] V. Paxson *et al.*, "Flex-fast lexical analyzer generator," *Lawrence Berkeley Laboratory*, 1995.

[44] J. Levine, *Flex & Bison: Text Processing Tools*. "O'Reilly Media, Inc.", 2009.

[45] C. Donnelly and R. Stallman, "Bison. the yacc-compatible parser generator," 2004.

[46] J. R. Levine, T. Mason, and D. Brown, *Lex & yacc*. "O'Reilly Media, Inc.", 1992.

[47] S. C. Johnson, *Yacc: Yet another compiler-compiler*, vol. 32. Bell Laboratories Murray Hill, NJ, 1975.

Apêndices

A

TOPOLOGIA DO SISTEMA INFORMÁTICO DA BOSCH

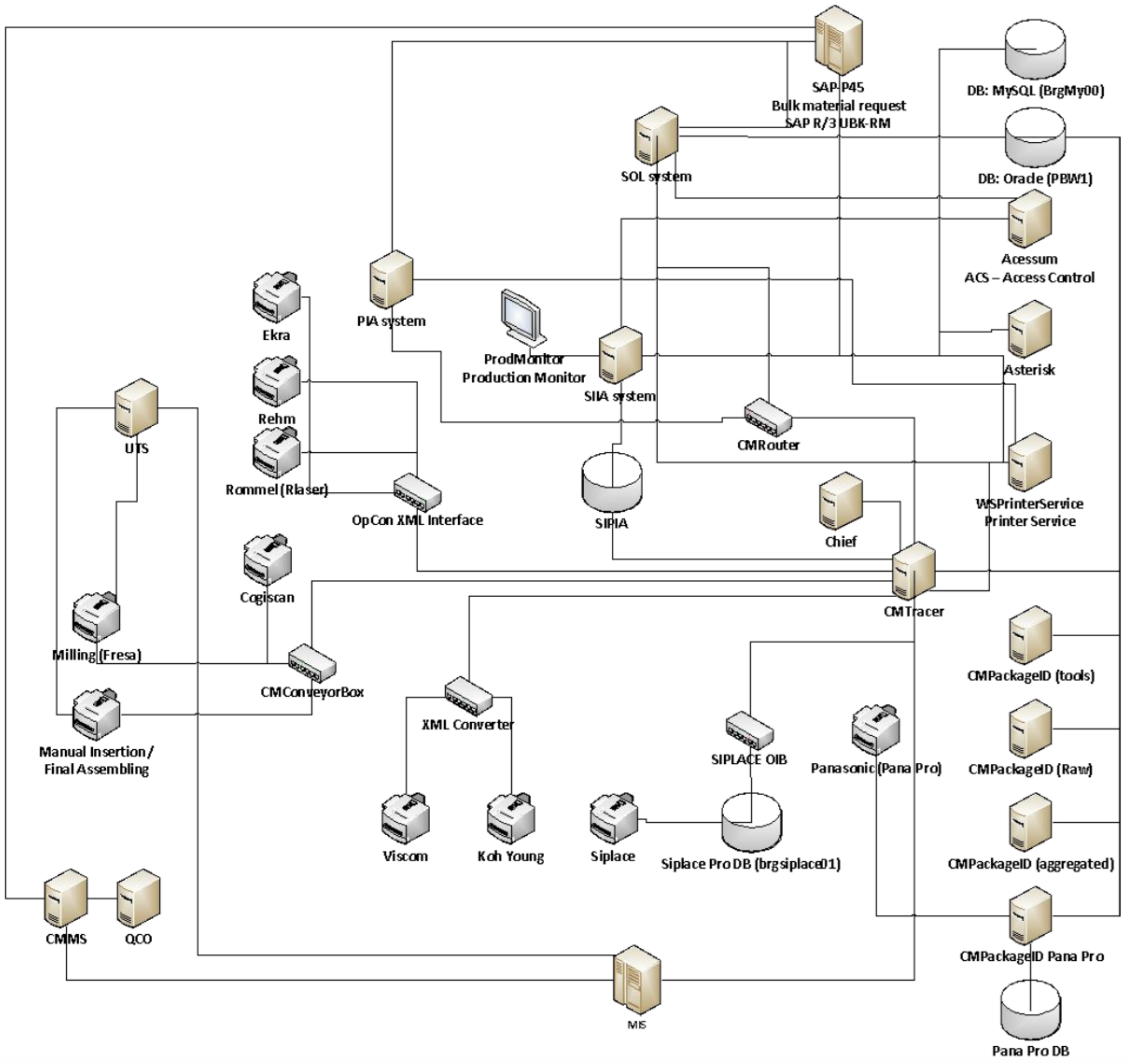


Figura 61: Overview do Sistema Informático da Bosch.

B

COMPARAÇÃO ENTRE OS VÁRIOS MOTORES DE BASE DE DADOS

Material utilizado aquando do estudo dos motores de base de dados *Firebird* e *Oracle*.

Ref: http://www.sql-workbench.net/dbms_comparison.html

Feature	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Queries										
Window functions	Yes	Yes(*)	Yes(*)	No	Yes	No(*)	No	No	No	No
Common Table Expressions	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes(*)
Recursive Queries	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes(*)
Row constructor(*)	No	Yes	Yes(*)	No	Yes	No	No	Yes	No	No
Filtered aggregates(*)	No	Yes(*)	No	No	No	No	No	Yes	No	No
PIVOT Support	Yes	No(*)	Yes	No	No	No	No	No	No	No
GROUP BY .. ROLLUP	Yes	No(*)	Yes	Yes	Yes	No	No	No	Yes	No
Temporal queries(*)	Yes	No	No	No	Yes	No	No	No	No	No
SELECT without a FROM clause	No	Yes	Yes	No(*)	No	No	Yes	No	No	Yes
Parallel queries(*)	Yes	No(*)	Yes	No	Yes	No	No	No	No	No
Aggregates for strings	Yes(*)	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes
Tuple comparison	Yes	Yes	No	Yes	Yes	No	(Yes)(*)	Yes	No	No
Tuple updates	Yes	No(*)	No	No	Yes	No	Yes	Yes	No	No
UPDATE with a join	No	Yes	Yes	Yes	No	No	No	No	No	No
ANSI date literals(*)	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No
Query variables(*)	No	No	Yes	Yes	No	No	Yes	No	No	No
Regular Expressions	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Comparison based on RegEx(*)	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No	No
Substring(*)	Yes	Yes	No	No	No	No	No	Yes	No	No
Replace(*)	Yes	Yes	No	No(*)	No	No	Yes	No	No	No
Constraints	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Deferred constraints(*)	Yes	Yes	No	No	No	No	No	No	Yes(*)	Yes
Check constraints	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Check constraints with sub-query	No	No	No	No	No	Yes	No	No	No	No
Check constraints using custom functions(*)	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No
Exclusion constraints(*)	No	Yes	No	No	No	No	No	No	No	No
Statement based constraint evaluation	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes
ON DELETE CASCADE(*)	Yes	Yes	(Yes)(*)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ON UPDATE CASCADE(*)	No	Yes	(Yes)(*)	Yes	No	Yes	Yes	Yes	No	Yes
Indexing	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Partial index(*)	Yes(*)	Yes	Yes	No	No	No	No	No	No	Yes
Descending Index(*)	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes
Index on expression(*)	Yes	Yes	(No)(*)	No	Yes	Yes	No	No	No	No

Index using a custom function (*)	Yes	Yes	No	No	Yes	No	No	No	No	No
Index include columns (*)	No	No	Yes	No	Yes	No	No	No	No	No
Clustered index (*)	Yes (*)	No	Yes	Yes	Yes	No	No	No	No	No
Duplicate NULL values in unique index (*)	No (*)	Yes	No	Yes (*)	No	No	Yes	Yes	No	Yes
DML	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Writeable CTEs (*)	No	Yes (*)	Yes (*)	No	No	No	No	No	No	No
Multi-row INSERTs (*)	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
TRUNCATE table with FK (*)	Yes (*)	Yes	No	No	No	No	No	No	No	No
Read consistency during DML operations (*)	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
MERGE support (*)	Yes	(No) (*)	Yes	Yes (*)	Yes	Yes	Yes	Yes	Yes (*)	No
SELECT .. FOR UPDATE NOWAIT (*)	Yes	Yes	Yes	No	No	No	No	No	No	No
RETURNING clause	No (*)	Yes	Yes	No	No	Yes	No	No	No	No
Parallel DML	Yes	No	No	No	No	No	No	No	No	No
Data Types (*)	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
User defined datatypes for columns (*)	No	Yes	No (*)	No	Yes	No	No	Yes	No	No
Domains (*)	No	Yes	(Yes) (*)	No	No	Yes	Yes	Yes	No	No
Distinct types (*)	No	No	No	No	Yes	No	No	No	No	No
Arrays	No	Yes	No	No	No	Yes	Yes	Yes	No	No
Enums (*)	No	Yes	No	Yes	No	No	No	No	No	No
IP address	No	Yes	No	No	No	No	No	No	No	No
BOOLEAN (*)	No (*)	Yes	No (*)	No (*)	No (*)	No (*)	Yes	Yes	Yes	No
Interval	Yes	Yes	No	No	Yes	No	No	Yes	No	No
TIME (*)	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
DATE (*)	No (*)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
TIMESTAMP (*)	Yes	Yes	Yes (*)	Yes (*)	Yes	Yes	Yes	Yes	Yes	No
Range types (*)	(No) (*)	Yes	No	No	No	No	No	No	No	No
DDL	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Transactional DDL (*)	No	Yes	Yes	No	Yes	Yes	No	No	No	Yes
Computed columns (*)	Yes	No	Yes	No (*)	Yes	Yes	Yes	Yes	No	No
Functions as column default (*)	Yes	Yes	Yes	No	No	No	Yes	No	No	Yes
Sequences	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No
Auto increment columns (*)	Yes (*)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Synonms	Yes	No	Yes	No	Yes	No	No	No	Yes	No
Non-blocking index creation (*)	Yes	Yes	Yes	No	Yes	No	No	No	No	No
Partitioning	Yes	(Yes) (*)	Yes	Yes	Yes	No	No	No	No	No
Cascading DROP (*)	Yes	Yes	No	No (*)	Yes	No	Yes	Yes	No	No
DDL Triggers (*)	Yes	Yes	Yes	No	No	No (*)	No	No	No	No
TRUNCATE Trigger (*)	No	Yes	No	No	No	No	No	No	No	No
Custom name for PK constraint (*)	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
ALTER a table used in a view (*)	Yes	No (*)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Add table column at specific position (*)	No	No	No	Yes	No	Yes	Yes	Yes	No	No

Temporary Tables	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Global temporary tables(*)	Yes	No	No	No	Yes	Yes	No	No	No	No
Use a temporary twice in a single query	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No(*)	Yes
Programming	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Stored procedures(*)	Yes	(Yes)(*)	Yes	Yes	Yes	Yes	No	Yes	No(*)	No
Table functions(*)	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	No
Custom aggregates(*)	Yes	Yes	No(*)	No	No	No	No	No	No	No
Function overloading(*)	Yes(*)	Yes	No	No	Yes	No	No	No	No	No
User defined operators(*)	No(*)	Yes	No	No	No	No	No	No	No	No
Statement level triggers(*)	Yes	Yes	Yes	No	Yes	No	(No)(*)	Yes	Yes	No
Row level triggers(*)	Yes	Yes	No	Yes	Yes	Yes	(No)(*)	Yes	Yes	Yes
Before triggers(*)	Yes	Yes	(No)(*)	Yes	Yes	Yes	(No)(*)	Yes	Yes	Yes
Dynamic SQL in functions(*)	Yes	Yes	No(*)	No	Yes	No	No	No	No	No
Dynamic SQL in triggers(*)	Yes	Yes	No	No	No	No	No	No	No	No
Delete triggers fired by cascading deletes(*)	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No
Built-in scheduler	Yes	No	Yes	Yes	Yes	No	No	No	No	No
Views	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Updateable Views	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No
WITH CHECK OPTION(*)	Yes	Yes(*)	Yes	Yes	Yes	Yes	No	No	No	No
Triggers on views	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes
Views with derived tables(*)	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
JOINs and Operators	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
CROSS JOIN	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
FULL OUTER JOIN	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No
LATERAL JOIN	Yes(*)	Yes(*)	Yes(*)	No	Yes	No	No	Yes	No	No
JOIN ... USING (...)(*)	Yes	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes
INTERSECT	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes
EXCEPT	Yes(*)	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes
ORDER BY ... NULLS LAST	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	No
IS DISTINCT FROM	No	Yes	No	Yes(*)	No	Yes	No	Yes	No	No
BETWEEN SYMMETRIC	No	Yes	No	No	No	No	No	Yes	No	No
OVERLAPS(*)	Yes	Yes	No	No	No	No	No	Yes	No	No(*)
Other	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
Catalogs ("databases")	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes
Schemas	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No
INFORMATION_SCHEMA(*)	No	Yes	Yes	Yes	No	No	Yes	Yes	No	No
NoSQL Features	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
XML Support(*)	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
XPath(*)	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
XQuery	Yes	No	Yes	No	Yes	No	No	No	No	No
JSON(*)	Yes(*)	Yes	No	No(*)	Yes(*)	No	No	No	No	No
Key/Value storage	No	Yes	No	No(*)	No	No	No(*)	No	No	No

Security	Oracle	Postgres	SQL Server	MySQL	IBM DB2	Firebird	H2	HSQLDB	Derby	SQLite
User groups / Roles	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No
Row level security(*)	Yes	No(*)	Yes	No	Yes	No	No	No	No	No
Grant on column level(*)	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No

C

EXCERTO DA GRAMÁTICA TABLE

```
Tabelas : table_specific table_statements ';'                                   {$$ = colaChar(3,$1,$2,";\n/\n");}
        ;

table_specific  : CREATE TABLE word_specific                                    {   table_name = $3;
                                                                                    addTabela($3);
                                                                                    $$ = colaString(2,"CREATE TABLE",$3);
                                                                                }
                | ALTER TABLE word_specific                                     {   table_name = $3;
                                                                                    addTabela($3);
                                                                                    $$ = colaString(2,"ALTER TABLE",$3);
                                                                                }
                ;

table_statements    : table_create_statement                                    {$$ = $1;}
                    | table_alter_statement                                     {addIndexes(table_name,$1);$$ = $1;}
                    ;

table_create_statement  : '(' column_list ')'                                   {$$ = colaChar(3," (\n",$2,"\n)");}
                        ;

table_alter_statement   : ADD CHECK '(' express_conditional ')'                 {$$ = colaChar(3," ADD CHECK (",$4,")");};
                        | ADD table_mprimary_item table_extra_item              {$$ = colaString(3," ADD",$2,$3);}
                        | ADD table_primary_item table_extra_item               {$$ = colaString(3," ADD",$2,$3);}
                        ;

table_mprimary_item : CONSTRAINT word_specific PRIMARY KEY '(' primary_list ')' {$$ = colaChar(5,"CONSTRAINT ",$2," PRIMARY KEY (",$6,")");}
                    ;

table_primary_item  : PRIMARY KEY '(' primary_list ')'                          {$$ = colaChar(5,"CONSTRAINT PK_",table_name," PRIMARY KEY (",$4,")");}
                    ;

table_extra_item    : USING INDEX word_specific                                 {$$ = colaString(2,"USING INDEX ",$3);}
                    |                                                           {$$ = "";}
                    ;

column_specific : word_specific type_specific column_opt            {addColuna(table_name,$1,$2);$$ = colaString(3,$1,$2,$3);}
                | word_specific word_specific column_opt            {   dom_name = procura($2);
                                                                        addColuna(table_name,$1,$2);
                                                                        $$ = colaString(3,$1,$2,$3);
                                                                    }
                ;

column_opt  : NOT NULLX                                             {$$ = colaChar(1,"NOT NULL");}
            | NULLX                                                 {$$ = colaChar(1,"NULL");}
            | DEFAULT string_specific                               {$$ = colaString(2,"DEFAULT",$2);}
            |                                                       {$$ = colaChar(1,"");}
            ;


type_specific   : BLOB SUB_TYPE number SEGMENT SIZE number_specific     {   if($3==1) $$ = colaChar(1,"CLOB");else $$ = colaChar(1,"BLOB");
                                                                        }
                | CHAR '(' number_specific ')'                          {$$ = colaChar(3,"CHAR(",$3,")");}
                | DATE                                                  {$$ = "DATE";}
                | DECIMAL '(' number_specific ',' number_specific ')'   {$$ = colaChar(5,"NUMBER(",$3,",",$5,")");}
                | FLOAT                                                 {$$ = "FLOAT";}
                | DECIMAL                                               {$$ = "DECIMAL";}
                | INTEGER                                               {$$ = "NUMBER";}
                | NUMERIC '(' number_specific ',' number_specific ')'   {$$ = colaChar(5,"NUMBER(",$3,",",$5,")");}
                | SMALLINT                                              {$$ = "SMALLINT";}
                | TIME                                                  {$$ = "TIME";}
                | DOUBLE PRECISION                                      {$$ = "DOUBLE PRECISION";}
                | TIMESTAMP                                             {$$ = "TIMESTAMP";}
                | VARCHAR '(' number_specific ')'                       {$$ = colaChar(3,"VARCHAR2(",$3,")");}
                ;
```