Universidade do Minho
Escola de Engenharia

Rui Daniel Ferreira Peixoto

Pervasive Data Mining Engine

Dissertação de mestrado

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho realizado sobre a orientação de

Professor Carlos Filipe da Silva Portela

e a coorientação de

Professor Manuel Filipe Vieira Torres dos Santos

Outubro 2015

DECLARAÇÃO

Nome Rui Daniel Ferreira Peixoto……

Endereço eletrónico: ruidfpeixoto@gmail.com.....Telefone: _____ / _____

Número do Bilhete de Identidade: 12604762……..

Título dissertação

Pervasive Data Mining Engine……

Orientador(es):

Professor Carlos Filipe da Silva Portela……

Professor Manuel Filipe Vieira Torres dos Santos…….....Ano de conclusão: 2015……

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação……...

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respetiva, deve constar uma das seguintes declarações:

1. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, 30/10/2015

Assinatura: Rui Daniel Ferreira Peixoto

Agradecimentos

Um especial agradecimento ao professor Filipe Portela, pela paciência nas alturas mais difíceis, pelo acompanhamento constante, mas permitindo toda a liberdade possível para desenvolver este trabalho.

Pervasive Data Mining Engine

Resumo

As ferramentas de *Data Mining* atuais necessitam de um conhecimento aprofundado na área para obter resultados otimizados. Esta dissertação apresenta como questão de investigação analisar "A viabilidade da construção de um sistema de *Data Mining* semiautónomo com características *pervasive*, sem perder a sua identidade e funcionalidades" e como objetivo desenvolver um protótipo de um sistema designado *Pervasive Data Mining Engine* que revolucione e facilite o processo de *Data Mining*. A metodologia de investigação utilizada é o *Design Science Research. O Pervasive Computing* oferece a possibilidade de serviços/aplicações estarem disponíveis em qualquer lugar, de forma invisível e não intrusiva ao utilizador. O protótipo desenvolvido apresenta benefícios como a realização automática de processos de *Data Mining*, a construção de modelos de *Data Mining* em paralelo, o registo de todas as instâncias do processo e a possibilidade de comparação entre elas. O *Data Mining Engine* é também capaz de se adaptar às necessidades dos utilizadores, com diversas possibilidades de configuração, resolvendo processos de *Data Mining* em modo automático ou manual, permitindo também um modo misto, em que é possível definir tarefa a tarefa o que é executado automaticamente ou manualmente pelo utilizador. Sendo este um projeto inovador e exploratório, os resultados obtidos durante a investigação são muito motivadores, ficando provado que é possível a construção de um sistema de *Data Mining* semiautónomo com características *pervasive.* O trabalho desenvolvido é também uma excelente base para continuar a investigação, desenvolvimento e aplicação deste conceito.

Pervasive Data Mining Engine

Abstract

The current data mining tools require great knowledge in the area to show optimized results. This dissertation presents as its investigation question “The viability of constructing a data mining system semi-autonomous with pervasive characteristics, without losing its identity and functionalities”, and its objective is to develop a prototype designated Pervasive Data Mining Engine that revolutionizes the Data Mining process. The methodology used in this investigation is Design science Research. Pervasive computing offers the possibilities of services and applications to be available every ware, non-intrusive and invisible to its users. The prototype developed, presents benefits such as automatic process execution, parallel construction of data mining models and a detailed record of all instances of the Data Mining process. This Data mining Engine is also capable of adapting to the needs of its users, with many possible configurations. It executes processes in automatic or manual mode but it also allows a mixed mode where it can be defined task by task if it is automatic or manual. It was proved that it is possible to construct a viable data mining system semi-autonomous with pervasive characteristic. Because this project is innovative and exploratory, the obtain results are very motivating. The developed work is also an excellent basis for future research, development and application of this concept.

Índice

Lista de abreviaturas e siglas

DM – Data Mining

CRISP-DM – Cross Industry Standard Process for Data Mining

DME – Data Mining Engine

DSR – Design Science Research

PDME – Pervasive Data Mining Engine

UbiComp – Ubiquitous Computing

ETL – Extract Transform and Load

ExT – Executores de Tarefas

SSD – Sistema de Suporte à Decisão

JSP – JavaServer Pages

HTML – HyperText Markup Language

Java EE – Java Enterprise Edition

RAE – Relative absolute error

MAE – Mean absolute error

RSE – Relative square error

ACC – Acuidade

TFN – Taxa de falsos negativos

TFP – Taxa de falsos positivos

Índice de Tabelas

Índice de Figuras

1. Introdução

1.1. Enquadramento e motivação

A quantidade de dados existentes hoje em dia escapa à compreensão humana, não só somos incapazes de ter uma perceção da quantidade, como (principalmente) da sua qualidade. *Data Mining (DM)* é mais uma solução para este problema, pois ajuda a navegar por estes dados, sintetizar e ultimamente compreender e retirar utilidade deste esforço. Atualmente existem diversos *Data Mining Engines (DME)* no entanto necessitam de utilizadores com conhecimento aprofundado na área e enumeras configurações para apresentarem resultados otimizados. Esta dissertação pretende estudar e investigar uma nova solução de fácil compreensão e utilização e consequentemente desenvolver um protótipo designado por *Pervasive Data Mining Engine (PDME)* que revolucione e facilite o processo de *DM*, contribuindo assim para uma maior adoção de *Data Minining Engines*, no máximo de áreas e utilizadores. A história ensina-nos que muitas descobertas são realizadas por pessoas fora da sua área de investigação/trabalho. A possibilidade de capacitar todas estas pessoas com um *DME* é uma grande motivação que apresenta inúmeras potencialidades.

1.2. Finalidade e objetivos

Esta dissertação apresenta como questão de investigação "É viável a construção de um sistema de *DM* semiautónomo com características *pervasive*, sem perder a sua identidade e funcionalidades?", sendo o objetivo desta dissertação responder a esta questão.

A finalidade desta dissertação é a construção de um protótipo de *PDME*, capaz de automatizar o processo de *DM* mas garantindo as caraterísticas *pervasive.* Permitindo assim não só uma otimização do processo, mas também a possibilidade de qualquer pessoa (com ou sem experiência) obter resultados com valor. Esta simplificação do processo é feita através da automatização de todo o processo de carregamento de dados, transformação, modelação, validação e apresentação de resultados, mas permitindo uma adaptação do processo ao utilizador. Dependendo do conhecimento do utilizador, o *PDME* permite diferentes níveis de otimização.

Para a realização do objetivo principal foram definidos 4 objetivos específicos:

- Análise de *DME* existentes;
- Definição da arquitetura de *PDME*;
- Desenvolvimento de um protótipo *PDME*;
- Teste do protótipo utilizando dados reais.

1.3. Metodologia de investigação

O *Design Science Research (DSR)* é geralmente definido como uma atividade que contribui para a compreensão de um fenómeno. Normalmente o objeto em estudo é um fenómeno natural, como por exemplo as emissões solares ou ciclo de vida de uma planta, no caso de *DSR* o objeto de estudo é criado (completo ou parcialmente), por exemplo um *software* que simule o ciclo de vida de uma planta. O processo de *DSR* tem um foco muito grande na contribuição de conhecimento, esta contribuição está representada na Figura 1. O *DSR* envolve a criação de um novo artefacto que não existe, a análise do seu uso/performance, e a reflexão do mesmo, aumentando a compreensão do fenómeno estudado (Vaishnavi & Kuechler Jr, 2007).

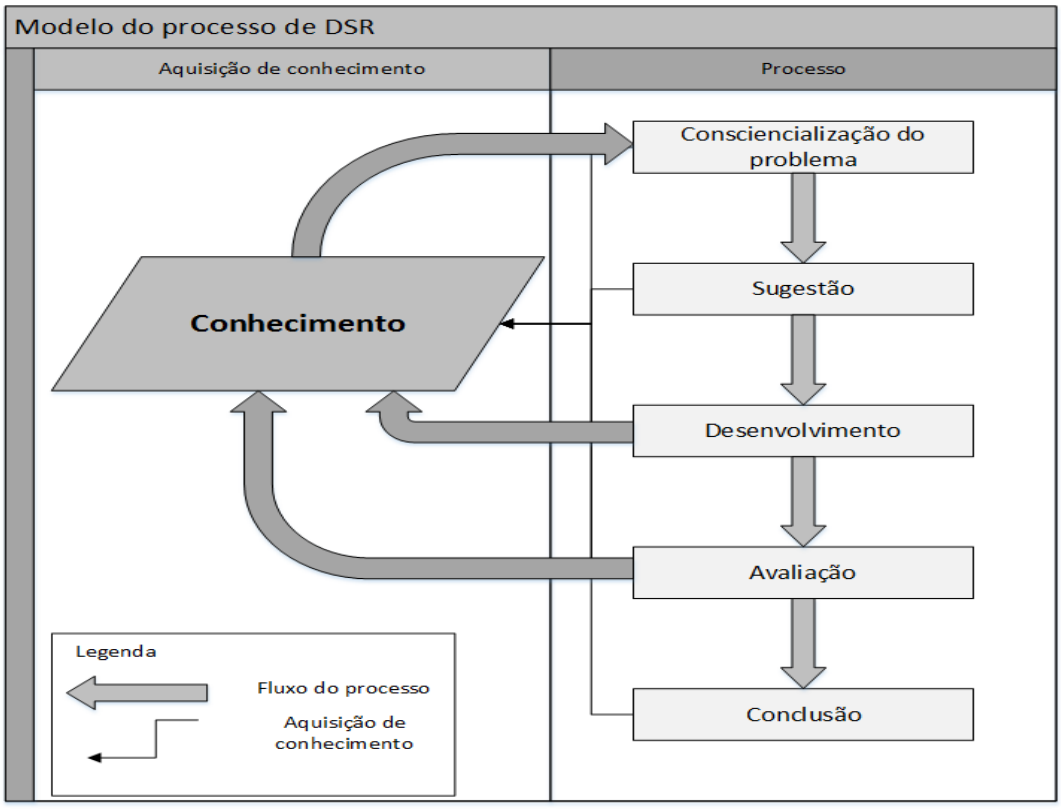

Figura 1 – Modelo de processo de *DSR* (adaptado de (Vaishnavi & Kuechler Jr, 2007))

1.3.1. *Design Science Research*

O processo desta metodologia é composto por 5 fases.

1.3.1.1. Cinco fases

As fases de *DSR* são sempre sequenciais, apesar disso permite retornar para fases anteriores, sendo normal a existência de iterações no processo *DSR*.

a. Consciencialização do problema

Na primeira fase é realizado a consciencialização de um problema existente, assim pretende-se realizar o levantamento da literatura, e formalizar o problema.

Resultados – Criação de um documento denominado pré-teste. Este documento contém a revisão de literatura.

b. Sugestão

Segundo Vaishnavi & Kuechler Jr (2007) a sugestão é essencialmente o passo criativo de *DSR*, elaborando uma tentativa de solução.

Resultados – Realização da análise das ferramentas existentes, sendo o resultado principal deste passo os requisitos e a arquitetura de uma possível solução.

c. Desenvolvimento

No terceiro passo é desenvolvida e implementada uma possível solução.

Resultados – Desenvolvimento do protótipo de um *Pervasive Data Mining Engine*

d. Avaliação

Após a construção do artefacto, este é avaliado segundo os critérios definidos utilizando dados reais. Qualquer desvio das espectativas tem que ser anotado e explicado (Vaishnavi & Kuechler Jr, 2007).

Raramente a tentativa inicial define completamente o comportamento do fenómeno, normalmente a informação recolhida na fase de desenvolvimento e avaliação contribuiu para a consciencialização do problema e o processo é recomeçado numa nova fase de sugestão, seguida novamente da fase 3 e 4 (Vaishnavi & Kuechler Jr, 2007), como é demonstrado na Figura 1.

Resultados – O resultado desta fase são os comportamentos observados do artefacto.

e. Conclusão

Fase terminal de *DSR*: é nesta fase que é consolidado e escrito o conhecimento adquirido durante o processo *DSR*.

Resultados – Registo do conhecimento adquirido, bem como as contribuições para o conhecimento do fenómeno e hipóteses sobre trabalho futuro. Escrita de documentos académicos e científicos contendo os resultados atingidos.

1.3.2. Resultado final

March e Smith (1995) defendem quatro resultados gerais para *DSR*.

- Construtores: são o vocabulário conceptual de um problema/solução;
- Modelos: são proposições ou declarações que expressão relações entre construtores;
- Métodos: são um conjunto de passos usados para executar uma tarefa;
- Instanciações: são operacionalizações de construções, modelos e métodos, e segundo March e Smith (1995) o resultado final de uma *DSR*. É a realização de um artefacto num ambiente.

O resultado final do processo *DSR* é o artefacto (protótipo).

1.4. Estrutura do documento

Este documento está dividido nas seguintes capítulos.

Introdução – Composto pelo enquadramento e motivação para a dissertação, bem como a sua finalidade e os objetivos. Apresenta também a descrição da metodologia de investigação utilizada nesta dissertação

Enquadramento conceptual – O enquadramento conceptual está divido em dois temas centrais, *pervasive computing/data* e *Data Mining*. No primeiro tema é apresentada a introdução ao tema, as suas vantagens e objetivos. É apresentado também um estudo mais aprofundado dos desafios, pois englobam 3 áreas diferentes: Sistemas distribuídos, computação móvel e *pervasive.* Por último é apresentado a visão de *pervasive data.* O segundo tema é iniciado pela introdução e a sua relação com *Knowledge Discovery Process,* Sendo depois abordado o modelo de processo de *DM* que será utilizado no desenvolvimento do *PDME*, *CRISP-DM*. Neste tema são também descritas as temáticas de modelação, avaliação *scoring*, algoritmos e *Data*

Mining Engine. É também apresentado uma análise das ferramentas de *data mining* existentes.

Conceito – Neste capítulo é apresentado o conceito do *Pervasive Data Mining Engine*. Iniciado com o impacto nas características das duas áreas de conhecimento envolvidas, e termina com as conclusões desses impactos, definindo o conceito do *Pervasive Data Mining Engine*.

Arquitetura – Neste capítulo são apresentados os requisitos e funcionalidades da nova ferramenta *Pervasive Data Mining Engine*, os benefícios desta ferramenta e a descrição da arquitetura. A descrição da arquitetura é constituída por duas subseções, a independência de componentes e a definição da arquitetura.

Protótipo – Neste capítulo são apresentadas as tecnologias utilizadas no projeto. A implementação e funcionamento dos módulos do protótipo.

Processo de investigação – Neste capítulo são apresentadas as iterações utilizadas durante o desenvolvimento desta dissertação segundo a metodologia de investigação definida.

Caso de estudo – Neste capítulo é apresentado a execução e os resultados de processo de *DM*.

Discussão – Neste capítulo são discutidos os resultados obtidos durante a investigação.

Conclusão – Neste capítulo são apresentadas as conclusões de todo o projeto.

Trabalho Futuro – Neste capítulo são apresentadas as principais ideias para trabalho futuro.

2. Enquadramento conceptual

Neste capítulo é apresentado na primeira secção a temática de *pervasive computing* e na segunda secção a temática de *Data Mining*. Estas duas temáticas são apresentadas independentemente pois nunca foram interligadas, sendo o objetivo desta dissertação fazer esta ligação. É também apresentado neste enquadramento as definições das duas áreas, e as necessidades para o desenvolvimento desta dissertação.

2.1. *Pervasive computing*

Devido à enorme utilização dos termos *Pervasive computing* e *ubiquitous computing (UbiComp)* como sinónimos, neste documento foi decidido manter uma similaridade entre ambos ao nível da definição.

2.1.1. Introdução

"As tecnologias mais profundas são as que desaparecem. Elas entrelaçam-se no quotidiano do dia-a-dia até ficarem indistinguíveis do ambiente."(Weiser, 1991).

Pervasive computing advém do termo *UbiComp* introduzido à comunidade por Weiser. Nesta secção é apresentada a visão de *pervasive computing*, os seus dois princípios (ubiquidade e invisibilidade), as quais podem ser consideradas como a base da definição de *pervasive computing.*

Desaparecimento para o *background* – Um dos princípios de *UbiComp* é o "desaparecimento para o *background"* que segundo Araya (1995) pode ser subdivido em duas partes interligadas. Primeiro, a integração de computadores em objetivos físicos do mundo real, como por exemplo relógios e post-its substituídos por computadores que continuam a cumprir a sua função original, mas estendendo-a, são capazes de realizar operações autónomas, tais como, regular automaticamente a temperatura das salas considerando o número de pessoas e as suas posições reais numa casa, ou enviar uma notificação ao utilizador que o leite no frigorifico expirou a sua validade. Objetos do mundo real a cumprir a função que sempre cumpriram mas retirando esforço ao dia-a-dia das pessoas. A segunda parte refere-se à invisibilidade, à "*utilização inconsciente*" da tecnologia, mas a invisibilidade não é apenas uma consequência da tecnologia mas também da psicologia humana, apenas quando as pessoas têm conhecimento suficiente sobre algo, deixam de estar cientes dela, sendo considerada como garantida nas nossas vidas (Araya, 1995) até que falhe. Em termos

tecnológicos isto significa que os computadores têm que ser capazes de identificar e se adaptar ao ambiente sendo intuitivos ao utilizador (Weiser, 1993), sem necessidade de inteligência artificial (Weiser, 1991) .O conceito atual é a criação de modelos de decisão para cada situação que permitem aos dispositivos que se adaptem automaticamente (modelos estáticos) a cenários definidos, mas Lyytinen & Yoo (2002) teoriza que no futuro os dispositivos serão capazes de criar modelos incrementais dinâmicos que se adaptam a qualquer situação (modelos dinâmicos). Em conclusão a utilização da tecnologia computacional é expectável que seja tão inconsciente como é atualmente a utilização de uma televisão.

Ubiquidade – O segundo princípio de *UbiComp* advém direto da definição de "*ubiquitous*" que significa "em todo o lado", a disponibilidade da computação em todo o lado em qualquer altura (Weiser, 1991), este contexto não significa apenas que poderemos levar um dispositivo para qualquer local e continuar a utilizar plenamente as suas aplicações (Weiser, 1991), mas também que a aplicação/serviço que pretendemos utilizar está disponível em qualquer lugar, em qualquer dispositivo. Os dispositivos não devem ser vistos como "computadores pessoais" onde instalamos aplicações que pretendemos usar, mas sim como portais de acesso a aplicações/serviços. As aplicações/serviços são desenhadas para desempenhar uma função, que pode ser executada a partir de qualquer dispositivo em qualquer lugar (Banavar et al., 2000).

2.1.2. Vantagens

As vantagens de *pervasive computing* são de nível global, pois permitirá uma disseminação das tecnologias de computação a toda a população, removendo a necessidade de adaptação e resistência à mudança, a utilização inconsciente implica diretamente estas vantagens. Em situações mais específicas, *pervasive computing* irá permitir um maior conforto de vida com espaços inteligentes (Mark, 1999), uma maior produtividade com acesso a informação e computação em qualquer local (Weiser, 1991), bem como a autoconfiguração de dispositivos (Banavar et al., 2000). Concluindo o *Pervasive computing* promete uma nova era de computação, focada nas pessoas com a tecnologia em segundo plano, promovendo acesso direto, simples e inteligente a informação ou serviços em qualquer local (Ye, Dobson, & Nixon, 2008).

2.1.3. Objetivos

Pervasive computing tem quatro objetivos principais: disponibilização de serviços/aplicações em qualquer local, qualquer dispositivo, de forma não intrusiva e contextual (Weiser, 1991; Ye et al., 2008). Cada objetivo apresenta vários desafios, sendo alguns comuns entre si.

Os dois objetivos de disponibilização de serviços/aplicações apresentam desafios maioritariamente comuns com outras áreas de investigação, nomeadamente sistemas distribuídos e computação móvel. Os outros dois objetivos apresentam novos desafios associados maioritariamente à área *pervasive*.

2.1.4. Desafios

Satyanarayanan (2001) visiona estas três áreas como um processo evolutivo. Sistemas distribuídos como a primeira evolução iniciada com a introdução de computadores pessoais e redes locais. Portáteis e redes sem fios originaram a área de computação móvel, enquanto os *smartphones* e sensores, ditaram a evolução de *pervasive computing* (Figura 2).

Figura 2 – Impulsionadores na evolução áreas relacionadas com *pervasive computing* (adaptado de (Satyanarayanan, 2001))

Apesar de já existir uma solução para maior parte dos desafios em sistemas distribuídos e computação móvel, nem todas as soluções aplicam-se diretamente em *pervasive computing*. Por esta razão são apresentados aqui os desafios das três áreas, para que sejam considerados no desenvolvimento desta dissertação.

2.1.4.1. Sistemas distribuídos

Um sistema distribuído é constituído por componentes localizados em computadores conectados em rede, que comunicam e coordenam as ações apenas através de

mensagens. A internet é um exemplo de um sistema distribuído (Coulouris, Dollimore, & Kindberg, 2005).

A principal motivação para a criação de sistemas distribuídos é a partilha de recursos. Mas a complexidade destes sistemas apresentam os seguintes desafios, descritos por Coulouris et al.(2005):

Heterogeneidade

Um sistema distribuído tem que considerar a variedade e diferença das redes, *hardware*, sistemas operativos, linguagens de programação e implementações por diferentes programadores. A comunicação com a internet através de uma rede local tem uma implementação, enquanto que uma rede diferente necessita de uma outra implementação dos protocolos da internet para funcionar. Dados primitivos podem ter representações diferentes em *hardware's* diferentes. Apesar de os sistemas operativos implementarem os protocolos da internet, os interfaces de programação podem não ser os mesmos entre sistemas operativos. Diferentes linguagens de programação podem não representar os dados da mesma forma, se programas em diferentes linguagens têm que comunicar entre si, este problema tem que ser solucionado. Programas desenvolvidos por diferentes programadores podem não utilizar protocolos *standard*, impedindo assim a sua comunicação.

Uma solução para este problema é a utilização de um *middleware*, que fornece uma abstração sobre a rede, *hardware*, sistemas operativos e linguagens de programação, mas geralmente força a utilização de uma linguagem de programação única, *Java* por exemplo.

Abertura

Abertura é determinada pela capacidade de o sistema ser estendido ou reimplementado. A Abertura não pode ser conseguida sem a especificação e documentação dos interfaces chaves do sistema. O desenvolvimento por várias pessoas, a falta ou lenta publicação de *standards* são alguns dos desafios possíveis.

Segurança

Segurança é definida pelo equilíbrio (Figura 3) entre os três pilares, confidencialidade (garantir acesso aos dados apenas a utilizadores autorizados), integridade (impedir

alteração ou corrupção dos dados) e disponibilidade (impedir interferência no acesso aos dados).

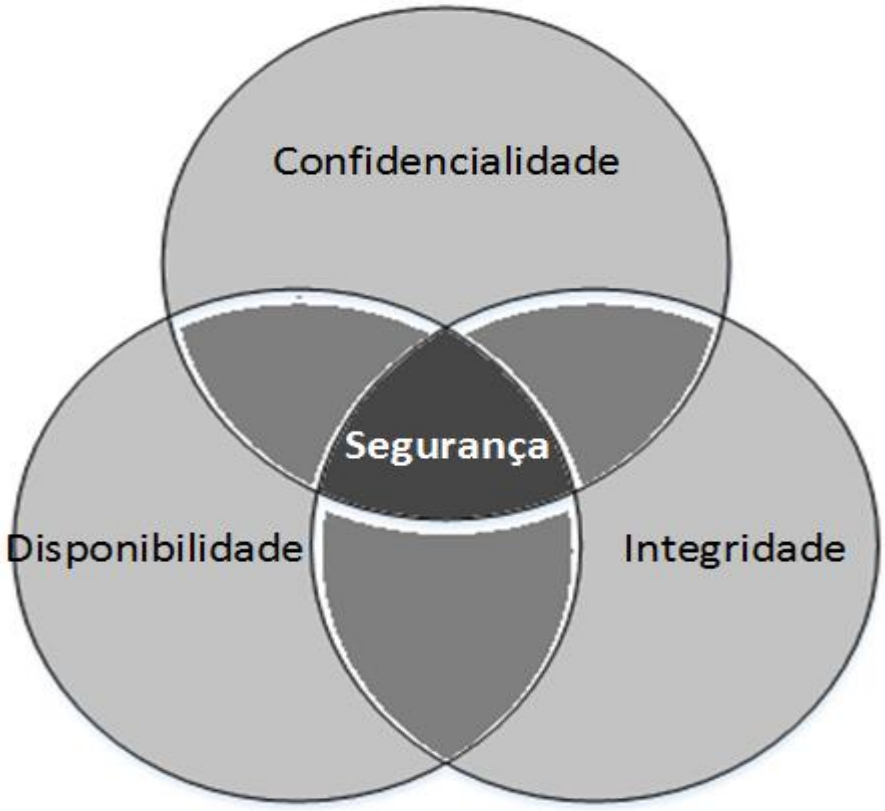

Figura 3 – Três Pilares Segurança (adaptado de (Pfleeger & Pfleeger, 2006))

Para garantir a confidencialidade é normalmente utilizado algum modelo de acesso ao sistema. Definindo quem acede a que recurso, ou conjunto de recursos, um exemplo desses modelos é o *lattice* (Denning, 1976). A Integridade dependo da necessidade e importância dos dados pode implicar muitas necessidades, pode implicar a necessidade de uma *firewall* para garantir que nenhum atacante, modifique ou apague informação do sistema, ou a redundância nos dados através de replicação para garantir que a corrupção de um disco não afete a integridade dos dados (Pfleeger & Pfleeger, 2006). Existem muitos mais exemplos, mas como em todas as questões relacionas com segurança, deve ser equacionado as ameaças, os ataques, e as vulnerabilidades do sistema, e atuar consoante o risco, definindo os controlos de segurança apropriados para o sistema. A disponibilidade em sistemas principalmente ligados à internet é difícil de atingir, hoje ainda não é possível garantir que os ataques de negação de serviço impeçam o acesso aos dados, mas a disponibilidade refere-se não só ao acesso, como também ao tempo necessário para os aceder. Considerando técnicas de

encriptação, normalmente utilizados para impedir o acesso não autorizado ou interceção de dados, no caso de a chave de encriptação ser perdida, o acesso a esses dados deixa de ser possível, ou por exemplo, devido ao elevado número de utilizadores do sistema e ao elevado processamento necessário para realizar o processo de cifra, o tempo de acesso ao sistema pode não ser comportável.

Escalabilidade

Um sistema é considerado escalável se continuar eficaz após um aumento significativo de recursos e utilizadores. Segundo Coulouris et al.(2005) a escalabilidade apresenta quatro desafios:

- **Controlo de custos dos recursos físicos**
  A adição de novos servidores tem que ser economicamente viável. A adição de novos servidores deverá ser proporcional aos novos utilizadores no sistema, por exemplo, o servidor inicial suporta 50 utilizadores, o segundo deverá suportar outros 50 utilizadores, ou o mais próximo possível e assim sucessivamente.
- **Controlo da perda de performance**
  No momento da definição de uma estrutura de dados, o seu tempo de acesso deve ser avaliado e não deve exceder uma escalabilidade logarítmica.
- **Prevenir a exaustão de um recurso de *software***
  A criação de um identificador de utilizador com um máximo de 6 carateres pode levar a exaustão desse mesmo recurso. Embora em alguns casos a previsão do número de utilizadores seja difícil, este problema deve ser sempre tomado em consideração.
- **Evitar performance *bottlenecks***
  A utilização de um recurso ou função centralizada deverá ser evitada pois pode gerar um *bottleneck* no sistema. Quando não é possível evitar a sua utilização, técnicas como *cashing*, replicação de dados, ou utilização de múltiplos servidores, são soluções de sucesso comprovadas.

Tolerância a falhas

Falhas de *hardware* e *software* podem produzir resultados errados ou impedir o funcionamento do sistema. O tratamento destas falhas é imprescindível, e pode ser realizado através das seguintes técnicas.

- Deteção de falhas – Algumas falhas podem ser detetadas, através de mecanismos como por exemplo o *cheacksum.*
- Ocultar falhas – As falhas detetadas podem ser ocultadas ao utilizador e/ou minimizar a sua importância, por exemplo, quando uma mensagem enviada não recebe uma resposta, a sua falha é ocultada ao utilizador e é reenviada uma nova mensagem. Claro que o servidor pode estar *offline*, e nunca receber uma resposta, mas esse caso é resolvido na técnica seguinte.
- Recuperação de falhas – Quando a falha é crítica e o sistema não consegue continuar o seu funcionamento, é necessário o restauro do sistema para a ultima versão dos seus objetos (Coulouris et al., 2005). Para atingir este objetivo, o sistema tem que guardar todas as versões dos objetos e restaurar para a ultima versão, ou para a versão anterior caso exista corrupção na última versão.
- Tolerância a falhas – Muitos dos serviços na internet exibem falhas (Coulouris et al., 2005). Quando a falha não pode ser ocultada e o seu funcionamento restaurado, o utilizador deve ser notificado com a informação indicada sobre o problema.
- Redundância – Deverá existir sempre (pelo menos) uma réplica de um serviço ou objeto, de forma a colmatar a falha desse serviço ou corrupção do objeto. Apesar de logico, a implementação de redundância sem perda de performance e aumento de custos é extramente difícil.

Concorrência

Por questões de performance, a enorme quantidade de pedidos a um recurso partilhado obriga a utilização de concorrência. A utilização de *threads* é muito comum atualmente, mas o problema principal desde o seu início contínua o mesmo, recursos partilhados. Em alguns casos é possível limitar que apenas uma *thread* aceda ao

recurso em simultâneo, mas nem sempre esta solução é aplicável, nestes casos um compromisso entre consistência e disponibilidade tem que ser considerado.

Transparência

Transparência é definido como a ofuscação ao utilizador e programador do sistema da separação dos componentes de um sistema distribuído. Isto significa definir que o nível de abstração vai ser disponibilizado ao utilizador ou programador do sistema. Por exemplo a transparência de localização: permite que os recursos sejam acedidos sem a necessidade de saber a sua localização.

2.1.4.2. Computação móvel

Forman e Zahorjan (1994) utilizam o termo *mobile computer* como um computador móvel capaz de comunicação sem fios. Estes dispositivos permitem níveis de comunicação que não eram possíveis através de um simples telemóvel, sendo distinguidos pela capacidade de comunicação constante com serviços e recursos em redes com fios.

Comunicação sem fios

Redes sem fios são instáveis. A distância, os obstáculos, o número de utilizadores e as taxas de transferência baixas, resultam numa qualidade de comunicação inferior às redes com fios (Forman & Zahorjan, 1994).

Mobilidade

A capacidade de mudar de localização altera a relevância dos dados. Técnicas como o *cashing* podem não ser aplicáveis, e até pode ser viável/necessário a computação dos dados em cada utilização, sem nunca os guardar no dispositivo.

Portabilidade

Computadores pessoais (secretária) estão desenhados para ser estacionários, tiram o máximo proveito do espaço, têm alimentação constante, apresentam uma elevada performance, etc. Os computadores moveis têm outras considerações, como o peso, a durabilidade e o tempo de bateria. Estas diferenças têm implicações a vários níveis sendo entre eles, a dimensão do interface gráfico do dispositivo, o número de

operações (computação), o espaço em disco, etc. Estes fatores não são triviais e têm que ser considerados.

2.1.4.3. Novos desafios *Pervasive*

Nesta secção são apresentados os diversos desafios da temática *pervasive computing.*

Escalabilidade localizada

A proliferação de dispositivos, utilizadores e aplicações em *pervasive computing* apresentam um novo desafio para além do descrito em sistemas distribuídos. As aplicações tradicionalmente necessitam de um desenvolvimento para cada tipo de dispositivo, em *pervasive computing* é impraticável devido à crescente diversidade de dispositivos e as suas arquiteturas.

Heterogeneidade de espaços físicos

A heterogeneidade a nível do ambiente tem que ser considerada, ambientes diferentes têm diferentes sensores, podem ter implementações diferentes e níveis de inteligência diferentes. Todos estes aspetos têm que ser transparentes ao utilizador.

Integração

Considere que uma aplicação *pervasive* utiliza a iluminação da sua casa para diversos propósitos, agora multiplique essa aplicação por cada dispositivo, e volte a multiplicar por cada outra aplicação que utiliza a iluminação da sua casa. A coordenação entre a mesma aplicação em dispositivos diferentes é óbvia, mas como definir quem tem prioridade entre diferentes aplicações, ou exclusividade ao acesso à iluminação da casa. Este é o grande desafio de integração.

Invisibilidade

Para um ambiente ser realmente invisível é espectável que os utilizadores não necessitem de o configurar. Apesar de os utilizadores poderem fazer alguma configuração, quando o ambiente não corresponde às espectativas, é esperado que o ambiente aprenda por si e se auto reajuste. As configurações dos utilizadores devem ser mínimas ou nenhumas e componentes no processo de aprendizagem do sistema.

Perceção de contexto

A contextualização do ambiente é um dos maiores desafios em *pervasive computing*, seja pela insuficiente ou falta de sensores, interação não compatível com os modelos existentes ou informação contraditória dos sensores, a contextualização incorreta pode provocar o insucesso do sistema, não pela facto de o sistema não ser funcional, mas sim por não ser aceite pelo utilizador.

Gestão de contexto

Após a contextualização do ambiente é necessário a sua eficaz utilização, é necessário uma perceção completa do espaço físico e a implementação de mecanismos para a sua atuação, por exemplo é necessário acesso ao controlo da iluminação ou encontrar alternativas, como por exemplo, a notificação ao utilizador.

2.1.5. *Pervasive Data*

O conceito de *pervasive computing* está intrínseco à computação e apesar de já existir projetos com sucesso em pequena escala de *pervasive computing* (Conti et al., 2012), o foco está em disponibilizar serviços de computação. O que não se tem abordado é o tema *pervasive data*, neste caso não é só os serviços de computação que devem estar disponíveis em qualquer momento e altura, como também os seus dados (sejam dados brutos ou tratados). Os dados devem ter características *pervasive e* se o acesso aos dados não pode ser executado da mesma forma que em computação, o "desaparecimento para o background" será apenas parcial. Assim os dados de uma forma indireta adquirirem as características de *pervasive computing.*

As características acima mencionadas foram todas consideradas no desenvolvimento deste protótipo, fazendo com que não só o processo de *DM* seja invisível ao utilizador, mas também o acesso aos dados resultantes de todo processo de *DM*. Por exemplo, no caso do processo de *DM*, não só as fases e as tarefas devem ser invisível ao utilizador, como também todas as instâncias devem ser registadas.

2.2. *Data mining*

2.2.1. Introdução

Atualmente o ritmo de geração de dados é superior à nossa capacidade de compreensão (Witten, Frank, & Mark, 2011). Uma solução possível para a compreensão destes dados é a utilização de *Data Mining* (*DM*).

DM é definido como a aplicação de algoritmos para a descoberta de padrões nos dados (U. M. Fayyad, Piatetsky-Shapiro, Smyth, & others, 1996; Witten et al., 2011). Mas U. M. Fayyad et al (1996) considera *DM* apenas uma das fases de *Knowledge Discovery Process (KDD)*. *KDD* é definido como um processo não trivial de identificação valida e potencialmente útil na procura de padrões nos dados (U. M. Fayyad et al., 1996).

Atualmente *Cross Industry Standard Process for Data Mining* (*CRISP-DM*) segundo o *website KDnuggets*[1] (*KDnuggets*, 2014a) é a metodologia mais utilizada no desenvolvimento de projetos de *DM*. Azevedo (2008) considera que *CRISP-DM* pode ser visto como implementação do processo *KDD* descrito por U. M. Fayyad et al. (1996), como tal será utilizado *CRISP-DM* no desenvolvimento deste projeto.

2.2.2. *Cross Industry Standard Process for Data Mining*

A metodologia desenvolvida pelo *Consortium Cross-Industry Standard Process for Data Mining* teve como objetivo criar um modelo de processo *standard*, não proprietário e livre para o desenvolvimento de *Data Mining* (Chapman et al., 2000).

2.2.2.1. Hierarquia

CRISP-DM utiliza um processo com uma hierarquia com quatro níveis de abstração: fases, tarefas genéricas, tarefas especializadas, e instâncias.

O nível de topo é composto por 6 fases, como pode ser visualizado na Figura 4. Cada fase é subdividida em tarefas genéricas (dois níveis) capazes (teoricamente) de cobrir todas as possibilidades do processo de *DM* e as características de qualquer ferramenta utilizada. Estes dois níveis representam o modelo de processos *CRISP-DM* sendo as suas tarefas comuns a todos os projetos(Chapman et al., 2000).

[1] **www.KDnuggets.com**

No terceiro nível são descritas como devem ser executas as tarefas genéricas em situações específicas. Por exemplo, considerando uma tarefa de segundo nível denominada limpeza dos dados, no terceiro nível é descrito como esta foi executada, que decisões e operações foram realizadas (Chapman et al., 2000). Neste nível as tarefas são executadas consoante a especialidade, cada caso é um caso.

No quarto nível é feito o registo das ações, decisões e resultados, representando o processo específico que foi executado. Os dois últimos níveis representam o processo em si, sendo diferente para cada projeto, dependendo das especificidades.

2.2.2.2. Modelo de processo

CRISP-DM tem um ciclo de vida composto por 6 fases: Compreensão do negócio, compreensão dos dados, preparação dos dados, modelação, avaliação e implementação.

Compreensão do negócio – Esta fase inicial foca-se em compreender os objetivos e requisitos do negócio, convertendo este conhecimento num problema de *DM*.

Compreensão dos dados – Esta fase foca-se em identificar problemas e qualidades nos dados e elaborar análises permitindo assim uma familiarização com os dados.

Preparação dos dados – Esta fase engloba todas as atividades necessárias para a construção do *dataset* ou *datasets*, que serão utilizados na fase de modelação.

Modelação – Nesta fase é selecionado e aplicado um ou mais métodos de modelação e são calibrados os seus parâmetros de forma a otimizar os resultados.

Avaliação – Após a construção de um ou mais modelos que aparentemente apresentam qualidade, estes têm que ser avaliados para garantir que cumprem os objetivos de negócio definidos previamente.

Implementação – A criação do modelo não é geralmente o fim do projeto. Normalmente o conhecimento adquirido tem que ser organizado. A forma de execução desta fase vária de projeto para projeto, mas o objetivo comum é a apresentação do conhecimento de forma compreensível.

A sequência destas fases não é rígida e retornar a uma fase anterior é por vezes necessário e bastante comum (Chapman et al., 2000). Na Figura 4 podemos visualizar

as mais importantes e mais frequentes dependências entre as fases, neste caso deve ser sempre considerado que não existe uma sequência fixa entre as fases. O círculo externo na Figura 4 simboliza o ciclo natural de *DM*. O processo não termina quando a solução é implementada, pois normalmente o conhecimento adquirido desperta novas questões.

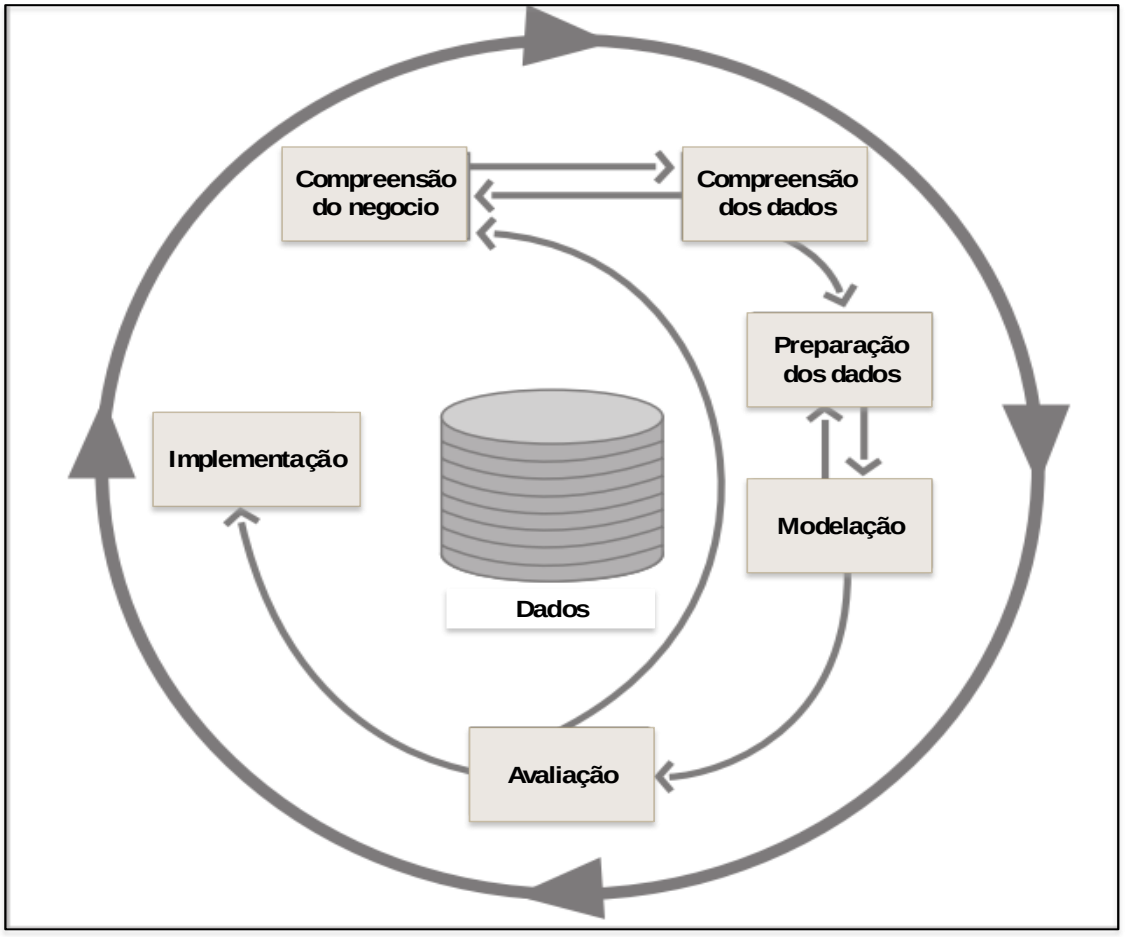

Figura 4 – Ciclo de vida *CRISP-DM* (adaptado de (Chapman et al., 2000))

Todo o processo de *CRISP-DM* está bastante detalhado e é demasiado extenso para ser descrito ao pormenor neste documento. Considerando isto é apresentado apenas na Figura 5 uma visão geral das tarefas de nível 2 do *CRISP-DM*.

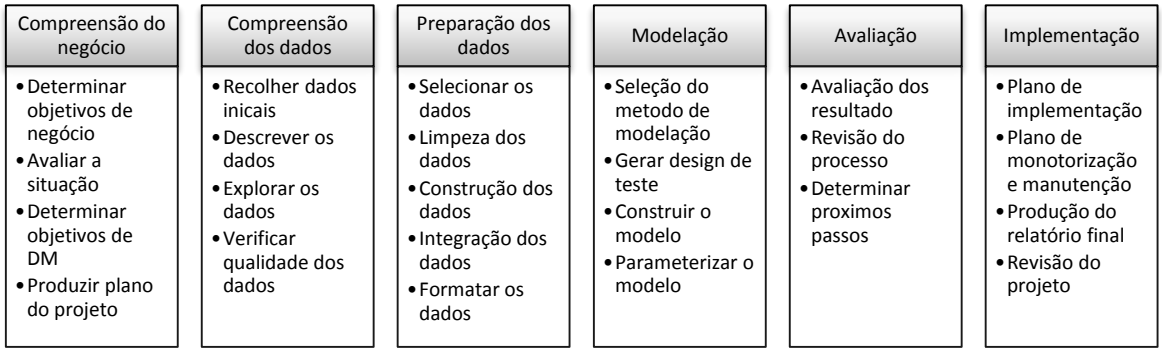

Figura 5 – Tarefas genéricas de nível 2 (adaptado de (Chapman et al., 2000))

Apesar de U. M. Fayyad et al. (1996) considerar *DM* como parte do processo de *KDD* e o paralelismo entre as fases de *KDD* e *CRISP-DM* possa ser feito (Azevedo, 2008), neste documento e para a realização do protótipo o conceito de *DM* foi alargado e teve em consideração todas as fases do *CRISP-DM*.

2.2.3. Modelação

O *DM* consiste na geração de modelos de padrões a partir dos dados. Na modelação existem dois objetivos ou categorias principais em *DM*, previsão ou descrição (Kantardzic, 2011).

2.2.3.1. Modelação preditiva

Modelação preditiva produz um modelo de um sistema baseado nos dados iniciais, e tem como objetivo prever um específico atributo (Y), baseado nos outros atributos (X) dos dados. Se o atributo (Y) a ser previsto for numérico ou continuo, então é utilizado o método de regressão, se for discreto então o método a ser utilizado é classificação (Bradley, Fayyad, & Mangasarian, 1999).

Classificação – A Classificação é uma função que mapeia cada registo para cada uma de várias classes predefinidas.

Regressão – A Regressão consiste em atribuir um valor numérico a cada registo através de uma função. Em regressão é gerada uma função que determina um valor numérico para todos os registos, esta função pode utilizar todos ou apenas alguns atributos dos dados (Hand, Mannila, & Smyth, 2001).

2.2.3.2. Modelação descritiva

Modelação descritiva produz padrões que descrevem os dados e tem como objetivo permitir a interpretação desses padrões.

***Clustering* –** Uma das técnicas descritivas mais comuns, também considerado como segmentação. O objetivo é segmentar os dados por *clusters* (classes) de acordo com a sua semelhança. A diferença entre *clustering* e modelação preditiva, é o conhecimento sobre as classes. Enquanto que na modelação preditiva cada classe tem um significado prévio associado, em *clustering*, o objetivo é encontrar esse significado (Bradley et al., 1999). Apesar de *clustering* ser inicialmente uma técnica descritiva também pode ser utilizado para realizar previsões, podendo assim também ser considerado modelação preditiva.

Sumarização – Uma técnica descritiva que tem como objetivo compactar um conjunto ou subconjunto de dados (Kantardzic, 2011), ou encontrar semelhanças entre atributos (Bradley et al., 1999).

Modelação de dependência – Uma técnica descritiva que tem como objetivo encontrar um modelo que descreve as dependências significativas entre atributos.(U. Fayyad, Piatetsky-Shapiro, & Smyth, 1996).

Deteção de mudança e desvio – Normalmente são utilizados em series temporais ou sequências. Foca-se em descobrir as mudanças mais significativas entre a última sequência e a medição ou normalização dos dados anteriores (U. Fayyad et al., 1996).

2.2.4. Avaliação

Para determinar a utilidade ou qualidade de um modelo é necessário avalia-lo. O processo de avaliação de um modelo requer a utilização de uma estratégia de divisão dos dados, um *dataset* para treino (dados que serão utilizados para gerar o modelo) e um *dataset* para teste (dados onde serão aplicados os modelos para realizar a avaliação), esta divisão é necessária para garantir que os mesmos dados não sejam utilizados no treino e na avaliação corrompendo as métricas. Existem várias estratégias de divisão de dados, a mais comum é utilizar dois terços dos dados para treino e um terço para avaliação (Witten et al., 2011). Esta estratégia é normalmente utilizada para *datasets* de maior dimensão. Mas esta divisão pode não ser ótima para *datasets* de menor dimensão, pois um terço dos dados não são utilizados no treino. Para esta situação existe uma técnica denominada *cross-validation*, em que os dados são divididos em X partes iguais. Normalmente são divididos em dez partes, em que nove partes são utilizadas para treino e uma para validação, repetindo este processo

dez vezes com uma parte de validação diferente, todas as vezes. Assim é garantido que todos os dados são treinados e testados. Existem outras estratégias de divisão, no entanto a solução de momento apenas apresenta estas duas possibilidades. Estes métodos e outros foram previamente detalhados por Witten et al. (2011).

Para quantificar a qualidade dos modelos é necessário utilizar métricas. Existem várias métricas específicas para cada tipo de algoritmo. Neste documento serão descritas apenas as que estão disponíveis no protótipo:

- Regressão
    - *MAE – Mean absolute error*

      Métrica calculada pela média da magnitude dos erros individuais sem considerar o seu sinal. Esta métrica tem a vantagem de tratar todos os erros por igual independentemente da magnitude (Witten et al., 2011).
    - *RSE – Relative square error*

      O erro é calculado relativamente à média dos valores utilizados no treino. Estas métricas (*relative*) são utilizadas quando a diferença relativamente à média é mais importante que o valor em si (Witten et al., 2011).
    - *RAE – Relative absolute error*

      Calculada da mesma forma que o MAE, mas transformada em relativa da mesma forma que o RSE (Witten et al., 2011).
- Classificação
    - *Precisão*

      Rácio entre as instâncias classificadas corretamente como positivas, e as classificadas como positivas.
    - *Acuidade*

      Percentagem de instâncias classificadas corretamente.
    - *TFN - Rácio de Falsos Negativos*

      Percentagem de instâncias corretamente classificadas como negativas. Esta métrica é derivada da matriz de confusão.
    - *TFP - Rácio de Falsos Positivos*

      Percentagem de instâncias corretamente classificadas como positivas. Esta métrica é derivada da matriz de confusão.

A matriz de confusão (apenas para classificação) é utilizada principalmente quando é importante verificar a qualidade para classes específicas (Cichosz, 2014). Na Tabela 1 é possível verificar as métricas para cada classe. Verdadeiros negativos (VN), significa que foi previsto a classe A e aconteceu A, Falsos negativos (FN), significa que foi previsto a classe B mas aconteceu A. Falsos positivos (FP) significa que foi previsto A, mas aconteceu B e Verdadeiros positivos (VP) significa que foi previsto B e aconteceu B.

Tabela 1 – Matriz de confusão

		Previsões	
		A	B
Classes	A	VN	FN
	B	FP	VP

Na Tabela 2 podemos verificar um exemplo, com um total de 20 previsões, onde o modelo previu a classe A, 7 vezes com sucesso e 3 com insucesso, e previu a classe B 5 vezes com sucesso e 5 com insucesso. Podemos então verificar que o modelo obtêm melhores resultados a prever a classe A, do que a classe B.

Tabela 2 – Exemplo de uma matriz de confusão

		Previsões	
		A	B
Classes	A	7	5
	B	3	5

Realçar que não existe uma métrica melhor que outra, a métrica mais adequada depende sempre do contexto do problema.

2.2.5. *Scoring*

Funções de *scoring* quantificam a qualidade de adequação do modelo criado aos dados, a sua utilidade é comparar a adequação entre modelos(Hand et al., 2001). Sem a utilização de funções de *scoring* é impossível determinar qual é o melhor modelo ou mesmo fazer otimizações nos parâmetros do modelo.

2.2.6. Algoritmos

Os algoritmos são implementações de modelos genéricos (classificação, regressão por exemplo). Existem diferentes técnicas de implementação (árvores de decisão, redes neuronais são alguns), algumas são específicas ao tipo de modelo, outras abrangem vários tipos de modelos, como podemos verificar na Tabela 3. Na Tabela 3 são apresentados alguns dos algoritmos mais comuns. Neste trabalho apenas serão aplicados modelos de classificação e regressão, como tal apenas os modelos desse tipo estão apresentados na Tabela 3.

Tabela 3 – Cruzamento de técnicas de algoritmos com tipos de modelos (adaptado de (Giraud-Carrier & Povel, 2003))

	Modelação preditiva	
	Classificação	Regressão
Árvores de decisão	X	(X)[2]
Regras de decisão	X	
Redes neuronais	X	X
Regressão linear/logística		X
Suport Vector Machine	X	X

2.2.7. *Data Mining Engine*

O Data mining engine (DME) é o mecanismo que oferece um conjunto de serviços de *DM* aos seus clientes. Um *DME* pode ser visto como uma caixa negra, onde tudo o que se realiza lá dentro é invisível (funciona sem a intervenção ou conhecimento) do seu utilizador, apenas podemos utilizar os serviços que nos são disponibilizados. A maior parte dos *DME* oferecem serviços de nível 2 do *CRISP-DM* (ex. seleção do modelo, seleção dos dados) enquanto outros, considerados por Mikut & Reischl (2011) como *Specialties* apenas implementam uma técnica de modelação (nível 3 *CRISP-DM*). O objetivo deste *DME* não é só disponibilizar os serviços nível 1, 2 e 3, mas também permitir que todo o processo *CRISP-DM* (desde a compreensão de dados à implementação) seja realizado sem a interação do utilizador.

A capacidade de disponibilizar serviços ao nível de processo vai de encontro às características *pervasive*, permitindo que qualquer utilizador retire valor do processo

[2] **(X) significa que nem todos os algoritmos suportam**

de *DM*. Para além disso para que o *DME* seja realmente um *PDME*, tem que não só estar disponível em qualquer plataforma mas também respeita todas as características *pervasive* definidas na secção 2.1

2.2.8. Análise ferramentas existentes

Existem atualmente enumeras ferramentas de *data mining* e segundo Huang, Liu, & Chang (2012) os dois fatores mais importantes na adoção de uma ferramenta de *data mining* são a perceção da facilidade de utilização e a perceção de proveito retirado da ferramenta. As ferramentas de *data mining* são também consideradas como mais difíceis de utilizar que outras ferramentas de Tecnologias de Informação (TI).

Segundo um questionário realizado no conhecido website KDnuggets[3] (KDnuggets, 2014b) as 10 ferramentas/software mais utilizadas são *RapidMiner*(44.2%), R(38.5%), Excel(25.8%), SQL(25.3%), *Python*(19.5%), *Weka*(17.0%), *KNIME*(15.0%), *Haddop*(12.7%), SAS base(10.9%) e Microsoft SQL Server(10.5%). Deve ser considerado que nesta lista estão presentes ferramentas/*software* que apenas realizam análises, ou são linguagens de programação. Retirando esses casos, verificamos que as ferramentas de *Data Mining* mais utilizadas são, *RapidMiner*, R, *Weka*, KNIME, *Hadoop*, *Microsoft SQL Server*.

[3] **www.KDnuggets.com**

3. Conceito

Neste capítulo será apresentado o conceito explorado nesta dissertação. Este conceito é derivado de dois conceitos já existentes, *pervasive computing* e *data mining*.

O cruzamento destes dois conceitos é uma inovação, as ferramentas atuais focam-se apenas no *DM*, não tendo em consideração o utilizador. *Pervasive computing* é exatamente o oposto. O seu foco é o utilizador, onde as ferramentas têm que ser invisíveis ao utilizador.

O foco no utilizador não é uma novidade, no entanto o *pervasive computing* não se concentra em *designs* intuitivos ou atrativos ao utilizador, estes são importantes, mas o seu real objetivo é abstrair o utilizador da complexidade de utilização da ferramenta e permitir a completa concentração do utilizador no objetivo final. Concluindo, o *pervasive computing* pretende que o utilizador despenda o seu tempo nas preocupações de negócio e nos seus objetivos.

3.1. Características *data mining*

A junção destes conceitos tem diversas implicações, nesta secção é apresentado o impacto que as características *pervasive* têm nas ferramentas e no processo de *DM*.

Um dos principais impactos é a hierarquia definida pelo *CRISP-DM*. Por exemplo, um *DME* pode ser caracterizado, respondendo às seguintes questões:

- Que fases executa?
- Considerando as fases que executa. Que tarefas genéricas executa?
- Considerando as tarefas genéricas. Que tarefas especializadas executa?
- Considerando todas as operações que executa. Quais são as instâncias utilizadas?

Existem naturalmente mais características, como por exemplo: se é um *DME* com arquitetura *descktop*, ou cliente/servidor, multiutilizador ou não, multitarefa ou não, se as operações são automáticas ou não. Estas características aplicam-se aos tradicionais *DMEs* e também se aplicam a este novo *DME*. A diferença neste *DME* é o impacto que as características *pervasive* têm nas características de um *DME* tradicional.

3.2. Características *pervasive*

Nesta secção são apresentadas as características *pervasive* e o seu impacto nas características de *DM*. O protótipo deve considerar as características *pervasive*:

- Disponibilidade em qualquer lugar e dispositivo (ubiquidade);
- Abstrair o utilizador da complexidade da ferramenta.

Apesar da primeira característica ser de fácil compreensão, a segunda é mais complexa do que inicialmente se pensa. Abstrair o utilizador da complexidade da ferramenta implica:

- Automatismo;
- Invisibilidade.

Enquanto que o automatismo pode ser atingido individualmente, o mesmo não se verifica para a invisibilidade, porque é necessário considerar o fator mais importante no conceito de *pervasive computing*, o utilizador. Apesar de ser possível atingir invisibilidade sem automatismo para um tipo de utilizador (experiente por exemplo), o mesmo não se verifica para um utilizador ligeiramente menos experiente. Para atingir esta invisibilidade é necessário identificar os utilizadores automaticamente e ajustar o comportamento do *DME* automaticamente.

Concluindo, é necessário uma especial atenção ao automatismo e invisibilidade, quando se quantifica o impacto destas características no processo natural de *DM*.

3.2.1. Invisibilidade

Esta é de facto a característica mais importante, mais complexa e mais difícil de atingir. Invisibilidade é a utilização inconsciente do utilizador, para atingir este objetivo o *DME* tem que completamente corresponder às expectativas do utilizador. Ou seja, o utilizador não tem de pensar, para utilizar o *DME* de forma eficaz e eficientemente.

Considerar as necessidades de invisibilidade na primeira questão:

- Que fases executa?

  Como diferentes tipos de utilizadores têm diferentes necessidades e todas têm que ser realizadas, por consequência o *DME* tem que suportar todas as fases,

com ou sem automatismo. Este automatismo ao nível das fases é uma das principais diferenças para outros *DMEs*, pois mais nenhum permite esta funcionalidade.

As implicações da invisibilidade nas outras questões são exatamente as mesmas que na primeira questão, uma especial atenção apenas às instâncias, pois novamente a maior parte dos *DMEs* não permitem a comparação entre diferentes configurações de processos de *DM*, não registam todas as operações realizadas, ou porque eliminam-nas na altura da criação de um novo processo, ou apenas o fazem parcialmente.

3.2.2. Automatismo

O automatismo significa que o *DME* tem que ser capaz de realizar operações sem intervenção humana. Para a tomada de decisão é necessário regras (modelo) definidas manualmente ou automaticamente.

A definição automática de regras, para a tomada de decisão automática, parece um contrassenso, mas na verdade não é, apenas utiliza bases de conhecimento diferentes. Enquanto no manual a base de conhecimento são as regras criadas pelo utilizador, resultante da experiencia do utilizador. No automático é utilizado as instâncias dos processos de *DM* anteriores como base de conhecimento, e as regras são criadas por um algoritmo de *DM*. Esta visão não é nova e é definida por alguns autores como a geração de modelos dinâmicos para tomada de decisão.

Nesta dissertação a criação de modelos dinâmicos faz todo o sentido, porque como se trata de uma ferramenta de *DM*, é a ferramenta ideal para a geração desses modelos. Mas, como qualquer pessoa, esta terá que ser ensinada e quanto mais utilizadores explorarem a ferramenta, melhor será a resposta. Especial atenção que com a introdução de modelos dinâmicos gerados internamente pelo *DME*, o *DME* é utilizador dele próprio.

3.2.3. Disponibilidade em qualquer dispositivo e lugar

Obrigatoriamente e por definição terá que ser um sistema distribuído e multiutilizador, e para servir eficientemente estes utilizadores, terá que suportar processamento paralelo. De notar que neste contexto processamento paralelo não implica que todas as operações a nível de implementação terão que implementar processamento paralelo, mas sim o necessário para servir eficientemente os utilizadores. Terá também

obrigatoriamente de estar disponível *online*, sem ser necessário a instalação de um software no dispositivo a utilizar.

3.3. Conclusão

Concluindo, este *DME* fornece os serviços de *DM* tradicionais, mas faz-lho a partir de qualquer lugar, em qualquer dispositivo, com percetividade de utilização igual para todos os tipos de utilizadores. É uma ferramenta para utilizadores de todos os níveis de conhecimento.

O conceito apresentado vai além do previsto no início da dissertação. Foi definido desde início que o *DME* iria apenas implementar as fases 2 a 6 do *CRISP-DM*. Durante o desenvolvimento do protótipo, foi possível verificar que a base de conhecimento poderia suportar a geração de modelos dinâmicos explicada na revisão de literatura e o *DME* poderia ser utilizador de si próprio. O conceito e a arquitetura já incluem estas funcionalidades, ficando apenas a implementação dos modelos dinâmicos por realizar. Esta funcionalidade inicia a implementação da fase 1 do *CRISP-DM*, mas está fora do âmbito desta dissertação.

4. Arquitetura

Neste capítulo é apresentada a arquitetura utilizada para solucionar o problema proposto no conceito. Este capítulo está dividido em 4 secções: Requisitos, funcionalidades, benefícios e definição da arquitetura.

4.1. Requisitos

Nesta secção são apresentados os requisitos do *pervasive data mining engine*. Este *DME* tem cinco requisitos:

- Escalabilidade – O sistema tem que ser capaz de escalar de forma a servir múltiplos utilizadores;
    - Sistema distribuído;
    - Processamento paralelo.
- Disponibilidade – O sistema tem que estar disponível em qualquer dispositivo e sistema operativo (conexão à internet presumida);
- Processos alto nível – Capaz de resolver automaticamente e manualmente várias fases do *CRISP-DM*, bem como as tarefas dessas fases;
- Tecnologias – O sistema tem que ser composto por tecnologias não proprietárias. Tecnologias não proprietárias permitem analisar, alterar, partilhar e executar as tecnologias livremente por qualquer pessoa.

4.2. Funcionalidades

Nesta secção são apresentadas as funcionalidades do *DME*:

- Sistema *pervasive* – Sistema disponível em qualquer dispositivo em qualquer lugar;
- Sistema distribuído – Sistema distribuído que opera em diversas máquinas;
- Persistência – Todo o sistema e fluxo de processos é guardado na base de dados, incluindo as suas decisões;
- Configuração automática – O sistema requer nenhuma ou mínima configuração para funcionar;
- Interface – O sistema tem interfaces diferentes consoante o tipo de utilizador;
- Multiutilizador – O sistema suporta múltiplos utilizadores em simultâneo;

- Adaptabilidade – O sistema está desenhado para a fácil implementação de novos modelos e outro *DMEs*, permitindo a customização do *DME* para áreas específicas de *DM*.

4.3. Benefícios

O maior benefício da utilização do *PDME* é a possibilidade de executar processos de *DM* eficientemente sem a necessidade de se focar no processo de *DM*. A sensibilidade para o negócio passa a ser o principal fator no processo de *DM*, podendo ser executado por qualquer utilizador independentemente da sua experiência em *DM*.

A um nível mais técnico o *DME* apresenta também imensos benefícios:

- Capacidade de execução de previsões em paralelo;
- Capacidade de incorporação de ferramentas de *DM* específicas no processo global;
- Capacidade de gestão automática de recursos a todos os níveis. Independência dos componentes (secção 4.4);
- Tecnologias implementadas são não proprietárias, e as suas implementações não foram adaptadas, permitindo um melhor suporte para desenvolvimento futuro;
- Base de conhecimento completo das instâncias do processo de *DM*;
- Possibilidade de implementação de sistemas de *business intelligence*.

Ao nível de utilizador:

- Permite utilização eficaz e eficiente de *DM* independentemente do conhecimento do utilizador na área;
- Possibilidade de utilizadores mais experientes diversificarem as suas técnicas revendo as decisões realizadas automaticamente pelo *DME*;
- Possibilidade de execução a partir de qualquer local e dispositivo.

4.4. Descrição

Nesta secção será detalhada a arquitetura do *DME*. As decisões realizadas durante a definição da arquitetura e a definição final.

No próximo ponto será detalhado uma decisão arquitetural extremamente importante para todo o projeto. A independência dos componentes e as suas implicações influenciam todo o projeto até à implementação.

4.4.1. Independência de componentes

Este *DME* tem como objetivo a disponibilização de serviços de *DM* para um número elevado de utilizadores e de todos os tipos de utilizadores. Destes dois objetivos surgem dois problemas:

- Escalabilidade

  Por exemplo, na Tabela 4 são apresentadas quatro tarefas com percentagens de processamento e o seu tempo de execução (valores totalmente arbitrários). Considerando que estas tarefas são executadas no mesmo componente e de forma sequencial para garantir que os tempos de execução não mudam, os recursos não serão usados na totalidade durante 7s, porque as tarefas 1, 2 e 3 não utilizam na totalidade o processamento.

Tabela 4 – Necessidades de processamento

Tarefa específica	Tempo execução	Processamento
Tarefa 1	1s	5%
Tarefa 2	2s	10%
Tarefa 3	4s	20%
Tarefa 4	30s	100%

  Com a adição de um segundo utilizador (execução de tarefas em paralelo), podemos verificar na Tabela 5 que apesar de as tarefas 1 a 3 não serem afetadas no tempo de execução, o tempo de execução para a tarefa 4 duplicou. Estes valores foram calculados considerando o melhor dos cenários. Portanto a introdução de processamento paralelo, melhora a utilização de recursos para algumas tarefas, mas piora o tempo de resposta para outras. Isto cria um problema pois o tempo de resposta é extremamente importante para garantir uma experiência constante aos utilizadores.

Tabela 5 – Necessidades de processamento de dois utilizadores

Tarefa específica	Tempo execução	Processamento
Tarefa 1	1s	10%
Tarefa 2	2s	20%
Tarefa 3	4s	40%
Tarefa 4	60s	100%

A solução é identificar estas tarefas na altura da definição da arquitetura e criar os componentes necessários para garantir uma utilização eficiente de recursos e uma experiência consistente aos utilizadores.

Considerar o mesmo problema para 10 utilizadores, mas agora com uma divisão das tarefas por vários componentes e vários servidores. Por exemplo, considerando que cada tarefa tem o seu próprio componente, e corre no seu servidor, a tarefa 1 era capaz de suportar os 10 utilizadores sem perdas de performance.

Na Tabela 6 podemos verificar agora a adição das colunas com o número de servidores e o componente associado à tarefa. Esta divisão por componentes permite que a adição de mais utilizadores mantenha o mesmo tempo de execução e por consequência a mesma experiência para o utilizador.

Tabela 6 – Necessidades processamento para 10 utilizadores

Tarefa específica	Tempo execução	Processamento	Servidores	Componente
Tarefa 1	1s	100%	1	A
Tarefa 2	2s	100%	2	B
Tarefa 3	4s	100%	4	C
Tarefa 4	30s	100%	10	B

Na realidade a utilização completa dos recursos nunca é possível, e a cedência em tempos de execução é sempre necessária quando fatores monetários são incluídos, no entanto, esta arquitetura permite definir o compromisso entre custo e desempenho.

Outra consideração, apesar de a tarefa 1 apenas demorar 1s e ser um bom candidato para cedência de performance em números de utilizadores muito

elevados, pode ser uma tarefa com extrema sensibilidade para o utilizador, enquanto a tarefa 3, pode já não se verificar este caso. Esta arquitetura permite tomar este tipo de decisões, determinar onde ceder, e não simplesmente aceitar a perda de performance como um todo.

Em conclusão, esta decisão obriga à criação de novos componentes anteriormente não previstos e têm implicações a nível de implementação, explicados na secção do protótipo.

- Necessidades de serviços

  Para além da questão de escalabilidade é necessário considerar os serviços que se pretende fornecer aos utilizadores. Por exemplo a existência de ferramentas do tipo *Specialties* (secção 2.2.7) garantem que uma única ferramenta de *DM* nunca irá resolver todas as necessidades de todos os tipos de utilizadores. Portanto a arquitetura tem que suportar várias ferramentas de *DM*.

As necessidades mencionadas anteriormente explicam maioritariamente o problema, mas não totalmente a questão da independência. Como foi mencionado anteriormente, a importância, o tempo de execução e as necessidades de processamento, não são complemente possíveis de prever antes da sua implementação ou execução, bem como a quantidade ou especificidade de ferramentas de *DM*. Considerando este problema, a seguinte questão apresenta-se. Como se pode definir uma arquitetura sem ter toda a informação necessária?

Independência

Como em todos os problemas complexos, a solução é partir o problema em problemas mais pequenos. Todas as tarefas têm que ser independentes umas das outras (até ao nível implementacional), esta independência permite que tarefas possam ser rapidamente transferidas entre componentes, refeitas completamente e que o processo seja realizado por várias ferramentas. Por exemplo, a tarefa de carregamento dos dados pode ser realizada no R, a tarefa de modelação no Weka, e o scoring no rapidmine. Isto é possível porque desde que o processo cumpra as regras do *CRISP-DM* e as tarefas não estejam a nível implementacional dependentes umas das outras, os pontos de início e fim de cada tarefa são sempre iguais independetemente da ferramenta que os execute.

A obrigatoriadade de existir esta independência garante que cada componente é responsavel pela execução eficaz das suas tarefas, permitindo também que cada componente implemente a sua própria estratégia de escalabilidade, com pouco ou nenhum impacto nos restantes componentes.

4.4.2. Definição da arquitetura

A arquitetura está dividida em 4 componentes principais: Base de dados, Controlo, Processamento e interface.

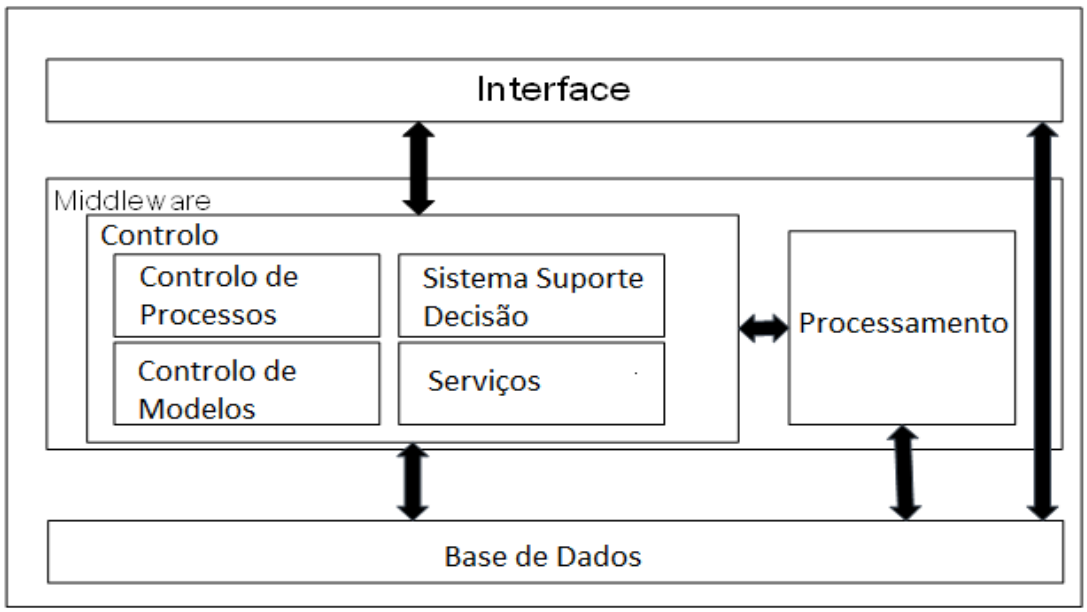

Figura 6 – Arquitetura

A divisão dos componentes segue uma estrutura normal de interface, *middleware* e base de dados, sendo o *middleware* composto por dois componentes. Esta divisão é necessária pelos fatores descritos na subsecção independência de componentes. As decisões arquiteturais focam-se maioritariamente nos requisitos *pervasive*. Neste caso, as questões de escalabilidade e invisibilidade são extremamente importantes.

- Base de dados – Todo o processo é guardado na base de dados, incluindo tarefas intermédias e decisões tomadas durante o processo. A base de dados será o suporte de execução para os outros três componentes, e o sistema é incapaz de funcionar caso este falhe.

  Como qualquer outro componente a base de dados tem a sua estratégia de escalabilidade, mas está fora do âmbito desta dissertação, todos os esforços

estão no componente de *middleware*, pois a estratégia de escalabilidade da base de dados vária caso a caso.

- Processamento – Responsável pela execução das tarefas de *DM* com maior necessidade de processamento. Adota uma arquitetura de múltiplos servidores, com cada servidor a executar apenas uma única tarefa, garantindo que essa tarefa é executada a aproximadamente 100% pois não tem outros processos a serem executados (processamento do sistema operativo ignorado).

  Por exemplo, considerando que todos modelos demoram o mesmo tempo e todos os servidores têm a mesma capacidade de processamento, a escalabilidade é linear, como podemos observar na Tabela 7.

Tabela 7 – Escalabilidade de processamento

Modelos	Tempo médio	Servidores	Terminado
10	1h	1	10h
100	1h	10	10h
1000	1h	100	10h

  Para além das questões de escalabilidade como foi referido na subsecção de independência de componentes, a capacidade de fornecer os mais diversos serviços é também uma necessidade. Esta arquitetura permite que várias ferramentas de *DM* sejam utilizadas para realizar a execução de algoritmos. Desta forma caso exista a necessidade de aplicação de um algoritmo específico só disponível numa ferramenta, a implementação dessa ferramenta é possível, e escalável consoante a necessidade.

  Esta independência permite também que o processamento continue a ser executado sem parar, mesmo que o resto do sistema falhe ou esteja em manutenção (excetuando a base de dados). A manutenção no resto do sistema não afeta os processamentos já iniciados.

- Controlo

  O Controlo é responsável pelo processo de *DM*, mas como qualquer outro componente, tem também a sua própria estratégia de escalabilidade. Apesar de atualmente este componente estar dividido em quatro subcomponentes, arquitecturalmente é o componente mais difícil de definir e mais dependente da implementação. Neste caso verifica-se que é a implementação a ditar as

divisões dos componentes para garantir a escalabilidade. Por exemplo, caso o controlo de modelos apresente necessidades de processamento superiores, no futuro pode ser necessário mais subdivisões. O importante é garantir que cada tarefa é independente e pode ser movida para outro subcomponente, com relativa facilidade. Este facto verificou-se durante o desenvolvimento do protótipo, onde várias tarefas foram movidas entre componentes, incluindo tarefas de suporte, e de comunicação.

As descrições apresentadas representam a função desempenhada atualmente.

- Controlo de Modelos – Responsável pela execução dos modelos de *DM* e comunicação com o componente processamento. É responsável também pela configuração de modelos e o processo de *scoring*.
- Controlo de processos – Responsável pelo controlo dos utilizadores, criação e configuração de novos processos. É também neste módulo que é controlado o carregamento de novos ficheiros para a base de dados.
- Sistema de suporte à decisão – Responsável pelos processos internos de tomadas de decisões.
- Serviços – Fornece serviços globais a todos os outros módulos, permitindo que qualquer módulo execute estes serviços garantindo um comportamento consistente independentemente do módulo que pretende executar o serviço.

- Interface – O objetivo do componente é disponibilizar os serviços do *DME* ao utilizador. Através de uma plataforma *web* o utilizador é capaz de interagir com o *DME* em qualquer local e em qualquer altura. Permitindo que utilize os serviços de *DME* de acordo com as necessidades do utilizador e o seu nível de conhecimento.

  Novamente, a escalabilidade deste componente não está no âmbito desta dissertação.

Na Figura 6 pode-se observar a comunicação existente entre os componentes principais do *DME*. De realçar que o componente Interface nunca comunica diretamente com o processamento, todos os pedidos de processamento são realizados no controlo, isto garante que todas as tarefas são realizadas da mesma forma,

independentemente de quem as inicia, seja no interface Web ou automaticamente pelo *DME*.

A disponibilização dos resultados é possível através do acesso direto à base de dados. É importante voltar a referir que se trata de um sistema distribuído, por isso realizar requisições de informação ao controlo, pode ter tempos de resposta muito voláteis, colocando esforço desnecessário ao controlo. Excetuando estes dois pormenores o processo segue uma estrutura de comunicação normal de interface, *middleware* e base de dados.

5. Protótipo

Neste capítulo são descritas as tecnologias utilizadas no projeto e os módulos desenvolvidos do protótipo.

5.1. Tecnologias

Para o desenvolvimento do protótipo foram utilizadas imensas tecnologias de forma a garantir os requisitos do projeto. Especial nota para o último requisito, em que todas as tecnologias não sejam proprietárias, ou seja, são *open-source* (utilização livre).

Em termos tecnológicos o primeiro componente a ser definido foi a base de dados. Como o objetivo do protótipo não é determinar a escalabilidade da base de dados mas sim do *DME* como um todo, foi definido que a base de dados é um componente independente e responsável pela sua própria escalabilidade. Desta forma qualquer base de dados pode ser utilizada com este *DME* (desde que seja SQL). Para este protótipo foi utilizado uma base de dados *mysql*, que cumpre a necessidades para um protótipo a esta escala. Em termos conceptuais para casos de larga escala, como já foi referido anteriormente qualquer outra base de dados pode ser usada para garantir a escalabilidade.

O segundo componente a ser definido foi o processamento. Depois de analisadas as ferramentas de *DM* existentes, entre as mais utilizadas, o R fornecia a maior compatibilidade com as necessidades do protótipo. Pelo facto de funcionar maioritariamente com scripts, permitia uma implementação mais fácil das otimizações, garantindo que todas as opções estariam sempre disponíveis por linha de comandos. Atualmente o protótipo executa três modelos, *rpart*, *naiveBayes* e *svm*.

O terceiro componente a ser definido foi o controlo. Pelo facto de ser um sistema distribuído seria necessário uma linguagem que facilitasse este processo e garantisse a estabilidade do sistema. Para este componente foi selecionada a linguagem *Java*, posteriormente *Java Enterprise Edition (Java EE)* aquando da seleção da tecnologia Web. A facilidade de desenvolvimento, documentação e garantia de qualidade faziam desta tecnologia a única opção para este componente.

O último componente a ser definido foi o interface, que será puramente *web*. De forma a facilitar a comunicação e como já seria utilizado java para o componente de controlo, a tecnologia *JavaServer Pages (JSP)* foi selecionada, sendo depois todo o

desenvolvimento convertido para *Java EE*. Tecnologias como *Java Script* e *HyperText Markup Language (HTML)* foram utilizadas também no desenvolvimento do componente *web*.

Como uma aplicação *Java EE* é executada num servidor, para este protótipo foi utilizado um servidor *GlassFish*. Desta forma a escalabilidade do sistema a nível de controlo e Web é garantida pela tecnologia, pois permite também a criação de *clusters*. Deve ser sempre considerado que a implementação da aplicação tem sempre impacto na escalabilidade, mas a escalabilidade física dos servidores é garantidas pela tecnologia, como no caso da base de dados.

Resumindo, neste trabalho foram utilizadas as seguintes tecnologias / linguagens:

- R;
- *Java EE;*
- *HTML/Java Script;*
- Servidores;
    - *mysql*;
    - *GlassFish*.

5.2. Módulos

5.2.1. Base de dados

Como referido anteriormente a base de dados é responsável pela sua escalabilidade. E como a escalabilidade da base de dados está fora do âmbito desta dissertação, apenas existiu a preocupação de fornecer uma base de dados capaz de suportar as necessidades deste protótipo.

Este componente é a base de todo o sistema. O sistema não funciona caso a Base de dados não esteja a operacional. Tudo o que é feito pelo *DME* é registado na base de dados, incluindo os *datasets*, as configurações, as decisões, os resultados, etc. O estado do processo é determinado pela base de dados. Todo o sistema pode ser reiniciado, que retomará no mesmo ponto aquando do seu término. A interação entre os outros componentes e a base de dados é total.

Na Figura 7 pode ser visualizado o Diagrama Entidade-Relacionamento do *DME*. Apesar de não existir uma divisão real na base de dados, esta será feita para que melhor se possa entender a sua implementação.

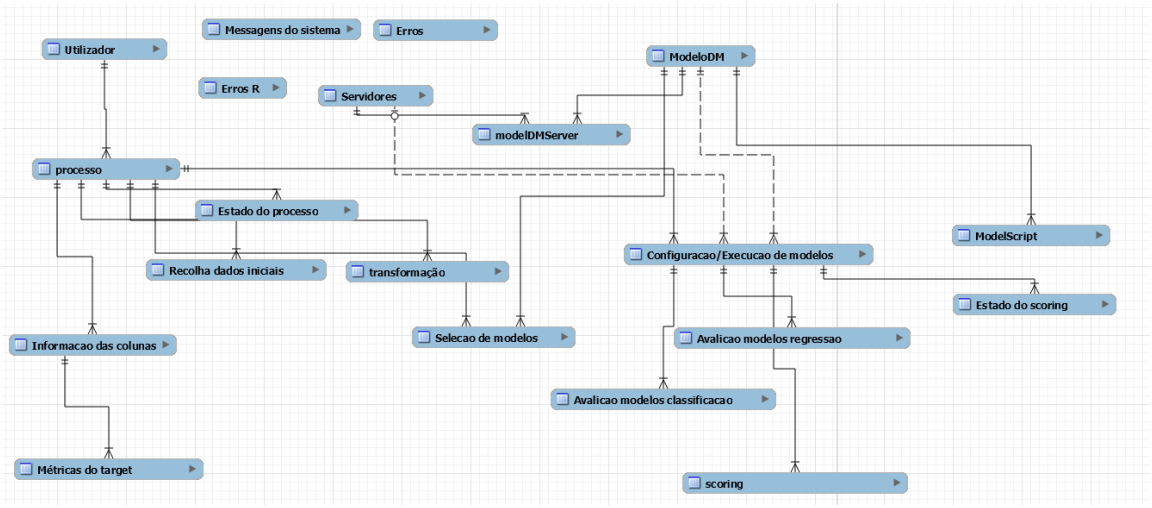

Figura 7 – Diagrama Entidade-Relacionamento (DER)

Dados do processo:

- Utilizador

  Tabela normal em todos os sistemas, guarda as informações relevantes do utilizador, como nome de utilizador, *password* e email.

- Processo

  Tabela com informação sobre os processos de cada utilizador, guarda o nome do processo e o nome da tabela gerada automaticamente para guardar os dados iniciais. É nesta tabela que se encontra a localização dos dados do processo, pois não existe uma tabela prévia para os *datasets*, cada *dataset* tem a usa tabela única. A tabela com os dados de *scoring* por regra, apenas acrescenta a palavra "*scoring*" como sufixo ao nome da tabela com os dados iniciais.

- Informação das colunas

  Duas tabelas, uma com informação geral das colunas dos dados iniciais, iniciais como o tipo de dados e se é o target do processo. A segunda tabela relacionada com a primeira, guarda informação dos níveis para as colunas com dados descritivos. Estas tabelas não guardam muita informação pois a descrição dos dados não foi explorada.

- Dados de treino e *scoring*

  Tabelas criadas automaticamente na tarefa de carregamento de dados iniciais. Esta são ajustadas ao *dataset* selecionado de forma a minimizar o espaço utilizado. O nome destas tabelas é gerado com o nome de utilizador e ficheiro de forma a garantir que é único no sistema. Estas tabelas podem guardar todo o tipo de dados.

Dados dos modelos e servidores

- Servidores

  Tabela com a informação dos servidores registados no sistema, bem como o seu estado (livre, ocupado, desligado), o seu *ip*, a porta do seu conector e tipo de servidor.
- Modelo *DM*

  Tabela com informação dos modelos inseridos no sistema, o seu tipo, a ferramenta associada, o nome, a biblioteca necessária, a linha de execução do modelo e os tipos de dados que suporta como treino e como target. Estes tipos de dados são necessários para evitar que os modelos que não suportam os dados numéricos por exemplo recebam esses dados e causem erros de execução. Mais informação na secção de trabalho futuro.
    - Configurações dos modelos

      Como será explicado na secção de controlo, o *DME* suporta todas as implementações de modelos R, para isso são utilizadas duas tabelas, uma guarda as configurações dos atributos numéricos de cada modelo, como o seu nome, o valor de defeito, mínimo, máximo e o seu incremento e a outra guarda os atributos descritivos, os seus valores e os valores de defeito.

Estados do processo e tarefas.

- Estado do processo

  Tabela de controlo central do todo o processo. Regista o estado de cada tarefa, as requisições de tarefas e os registos de término.

  Todas as seguintes tabelas nesta secção têm a *data* de registo, início e fim da tarefa.

- Recolha de dados iniciais

  Tabela de registo das tarefas de recolha de dados iniciais, composta pelos dados do ficheiro e o seu estado serve de suporte ao processo de recolha de dados. O seu estado vai sendo atualizado ao longo da sua execução.
- Transformação

  Tabela de registo das tarefas de transformação. Novamente uma tarefa não explorada por questões de tempo, sendo explicado com mais detalhe na discussão.
- Seleção de modelos

  Tabela de registo das tarefas de seleção de modelos. Guarda os modelos associados a cada processo para futura configuração. O seu estado vai sendo atualizado ao longo da execução da tarefa.
- Configuração/Execução de modelos

  Tabela de dupla função. São registadas as configurações dos modelos, sendo o controlo da execução realizado através dos estados. Estas tarefas utilizam a mesma tabela para limitar o processamento na base de dados. A informação que ambas as tarefas utilizam é a mesma, considerar a criação de uma tabela extra com apenas acréscimo de uma coluna com o estado, seria ineficiente e apenas aumentaria o processamento na base de dados. Esta tabela guarda as informações do processo, a configuração do modelo, o servidor e a ferramenta onde foi executado.
- Estado do *scoring*

  Tabela de registo das tarefas de *scoring*. É registado o modelo, o *fold* requisitado, e é controlado o seu estado. O registo dos *tuplos* de *scoring* é realizado noutra tabela.

Avaliação

- Avaliação dos modelos de regressão

  Tabela que guarda os resultados da avaliação dos modelos de regressão. É composto pelas colunas de informação do processo e por colunas referentes às métricas do modelo de regressão: RAE, MAE, RSE.
- Avaliação dos modelos de classificação

Para a avaliação de modelos de classificação são necessárias duas tabelas, uma para guardar as métricas gerais dos modelos (associadas a nenhuma classe) como por exemplo a ACC. Outra tabela é necessária para guardar as métricas associadas a cada classe do modelo, Precisão, TFP e TFN.

- *Scoring*

  Tabela que guarda os resultados do *scoring*, a previsão em valor numérico e a classe associada, em caso de classificação. É possível que esta tabela cresça para um número elevado de *tuplos* e necessite de uma estratégia específica para manter a performance no futuro, seja pela criação de uma tabela para cada utilizador, ou pela base de dados. Esta situação é explicada no capítulo de trabalho futuro.

- Métricas do target

  Tabela que guarda as métricas para o target do processo. São registadas nesta tabela e posteriormente acedidas pelo R para avaliar o modelo construído.

Mensagens

As tabelas de suporte ao *DME* contém o registo de operações e erros que ocorrem no sistema.

- Erros sistema

  Tabela para registos dos erros ocorridos no controlo, *web* ou conector R, composto pelas colunas, módulo, classe, mensagem de error, código do erro e *data* completa.

- Erros do R

  Tabela para registo dos erros ocorridos no R, composto pelas colunas mensagem erro, código do erro e *data* completa.

- Mensagens do sistema

  Tabela para registo de mensagens importantes ocorrentes no sistema. Tabela para uso generalizado pelo administrador ou programador. Composto por mensagem e a *data* completa.

5.2.2. Processamento

Este componente executa tarefas referentes a duas fases do *CRISP-DM*. Compreensão dos dados, realizando o carregamento e descrição dos dados. E na fase de modelação,

a execução e avaliação dos modelos. Considerar que em relação à fase de compreensão dos dados a responsabilidade por estas tarefas é partilhada com o controlo de processos, descrito na subsecção seguinte.

Este componente é composto por dois subcomponentes, a ferramenta R e um conector desenvolvido em Java. O R comunica diretamente apenas com o conector e a base de dados, o conector comunica com o controlo, a base de dados e o R.

a. Conector

O conector inicializa vários servidores (Lógicos), cada servidor físico necessita de um conector. Para o sistema global funcionar é necessário no mínimo um servidor lógico R de cada um dos seguintes tipos. Estes servidores lógicos não necessitam de estar no mesmo servidor físico, mas é necessário que pelo menos exista um de cada:

- *R-Data* – Servidor que apenas executas tarefas de carregamento e transformação de dados.
- *R-Model* – Servidor que apenas executa tarefas de modelação.
- *R-Scoring* – Servidor que executa apenas tarefas de *scoring.*

Estes tipos foram criados para garantir que o sistema tem sempre recursos mínimos necessários para executar estas operações, e também permite uma alocação de recursos de acordo com as necessidades naquele momento. Por exemplo se um grande número de novos utilizadores são registados é natural que as necessidades de servidor tipo *R-Data* aumentem, desta forma novos servidores deste tipo podem ser inicializados, ou outros tipos em funcionamento que estejam livres podem ser alocados para este serviço. Esta divisão permite também no caso de inclusão de novas ferramentas, estas sejam realizadas por tipo de servidor. Por exemplo, com a futura inclusão da ferramenta *Weka*, pode-se utilizar da mesma forma o *R-Data* para fazer o carregamento dos dados, e depois realizar a modelação em *Weka*, ou vice-versa.

Depois de inicializados, são registados na base dados sinalizando que estão novos servidores disponíveis para processar. Por defeito o conector inicializa um servidor de cada tipo, mais X número de servidores do tipo "*R-Model*", garantindo sempre que o número global de servidores lógicos não ultrapassa o número de *cores* lógicos do processador. O conector inicia também o serviço de comunicação com o Controlo.

b. Funcionamento

Quando o controlo define uma tarefa a executar é enviado um script para o conector, o conector regista a receção da tarefa na base de dados e envia a tarefa para o servidor R que foi requisitado pelo script a executar. Durante este tempo o conector aceita mais tarefas se tiver servidores lógicos disponíveis, caso contrário rejeita a tarefa.

Ao executar o script, o R acede diretamente à base de dados, executando as tarefas definidas no controlo, eliminando assim a necessidade do controlo enviar dados diretamente para o conector, diminuindo substancialmente o volume das comunicações. Para garantir que as tarefas são executadas o controlador monitoriza as respostas do R linha a linha e caso um erro seja detetado, o mesmo é registado na base de dados, o servidor logico é reinicializado e o controlo é notificado. Mais uma vez o objetivo é garantir a maior independência entre componentes quanto possível. A ferramenta R apenas executa as tarefas que lhe são delegadas, não resolve erros, a não ser que estas também lhe sejam delegados. Desta forma a introdução de diferentes ferramentas não quebra a lógica estabelecida pelo controlador.

c. Conclusão

Concluindo esta secção, o conector inicializa os servidores R, recebe as tarefas atribuiu-as aos servidores, o servidor executa as tarefas e guarda os resultados na base de dados. Estas tarefas são monitorizadas pelo conector, que regista na base de dados o sucesso ou insucesso das tarefas, notificando o controlo no fim, para que o processo continue.

5.2.3. Controlo

Responsável pelos processos e tomadas de decisões do *DME*, é preciso ter especial atenção neste componente, pois é necessário que as tarefas de segundo nível do *CRISP-DM* possam ser realizadas independentemente, respeitando o processo de tomada de decisão, de modo a garantir que os requisitos são cumpridos, nomeadamente o processo manual

Funcionamento (Nível 1)

Na Figura 8 é apresentado um exemplo de execução de uma tarefa de *DM* (Nível 1). Engloba os componentes Web e controlo, e executores de tarefas (Nível 2). É visível a separação da tomada de decisão da execução. Desta forma a decisão pode ser tomada

pelo utilizador ou pelo *DME*, garantindo que a execução é realizada da mesma forma independentemente de quem inicia a tarefa. Esta implementação garante que o *DME* funciona em modo automático e manual, mas mais do que isso, permite utilização dos dois em simultâneo.

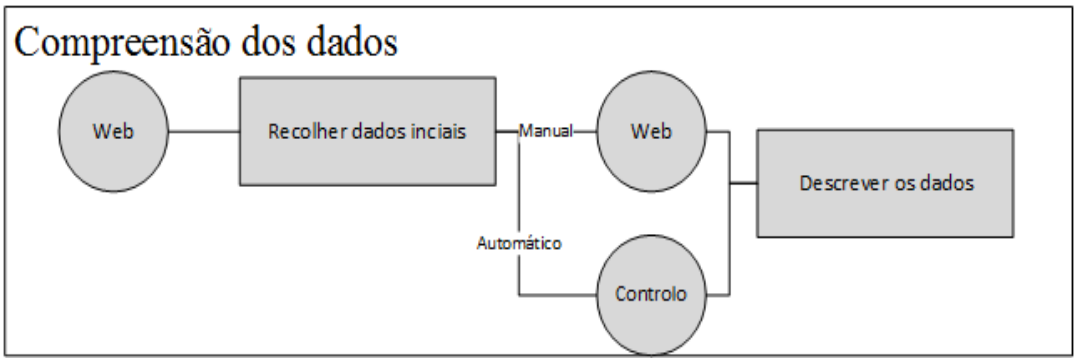

Figura 8 – Divisão de responsabilidades

Funcionamento dos executores de tarefas (Nível 2)

Não esquecer que este *DME* é um sistema distribuído e que realiza operações para múltiplos utilizadores. Para resolver este problema foram desenhados executores de tarefas (ExT) de nível 2.

ExT são implementações únicas no sistema (existe apenas um objeto no sistema) que garantem que as tarefas são realizadas da mesma forma, para todos utilizadores, independentemente de quem inicia a tarefa e utilizando os recursos disponíveis. Estas implementações têm duas variações, uma utiliza processamento sequencial e outra processamento paralelo. Por exemplo, na Figura 9 é apresentado o funcionamento da tarefa de configuração de modelos (à esquerda) e a tarefa de execução de modelos (à direita). Como pode ser visualizado a tarefa levanta todas as requisições (ainda não executadas) na base de dados e inicializa as tarefas uma a uma, esperando que termine para realizar a próxima (à esquerda). Tarefas que requisitam acesso ao componente de processamento (à direita) não executam a tarefa localmente nem esperam pelo seu fim, inicializam a tarefa e delegam para o processamento, posteriormente o registo de finalização é realizado pelo processamento. Apesar de no estado atual de desenvolvimento não ser necessário, caso no futuro seja necessário delegação de tarefas por questões de performance, a arquitetura do executor de tarefas de processamento pode ser aplicada nas tarefas de controlo diretamente.

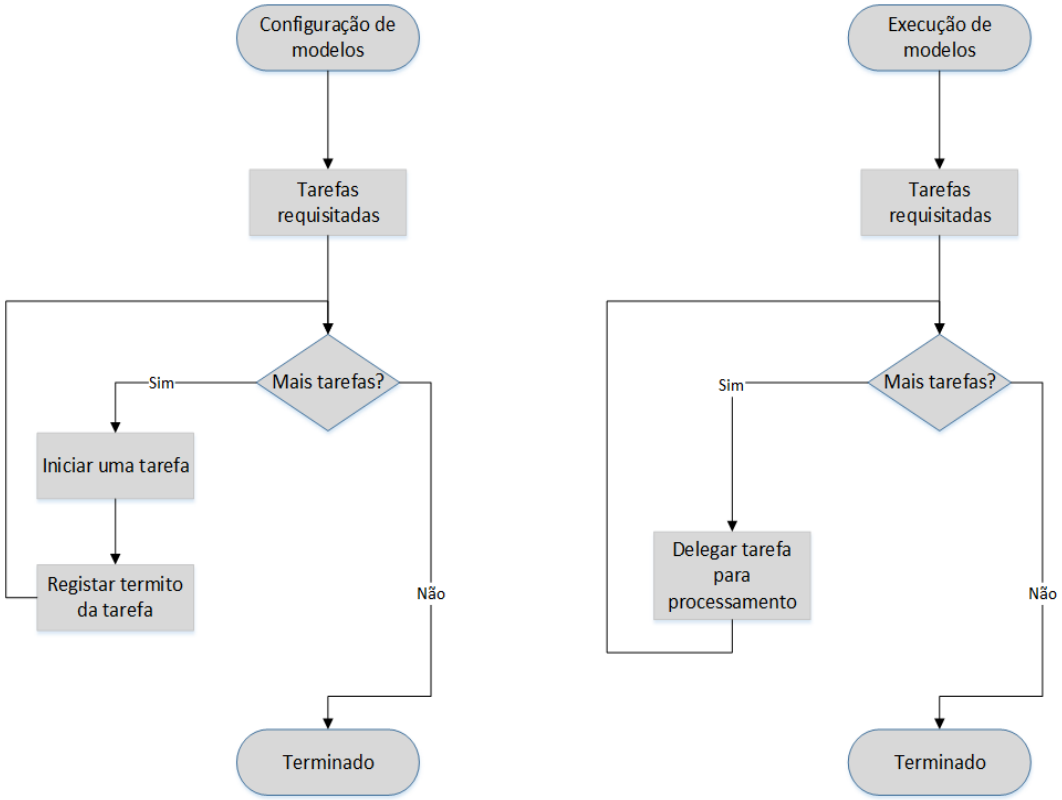

Figura 9 – Exemplo de um executor de tarefas

Para além do comportamento comum na execução das tarefas, a definição das tarefas a executar é também comum em todos os ExT como pode-se verificar na Figura 10. Antes de executar qualquer tarefa o ExT verifica os dois estados (automático e requisitado), o primeiro estado determina se esta tarefa está configurada para ser executada no interface *web* pelo utilizador ou pelo controlo do *DME*, e o segundo estado verifica se esta tarefa está requisitada para ser executada (próxima tarefa na ordem de execução do processo *DM*).

Isto significa que quando o ExT é executado, verifica todos os processos que estão em automático e se estão requisitados. No caso da *web* é apenas verificado para aquele processo específico.

Este comportamento é possível porque todas as tarefas registam da mesma forma o seu estado na base de dados, como pode-se verificar na Figura 11. O registo de início da tarefa é executado para garantir que não são iniciadas duas vezes (execução de modelos por exemplo), após a execução da tarefa e o seu registo na base de dados, é requisitado a execução da próxima tarefa e registado o fim da tarefa atual.

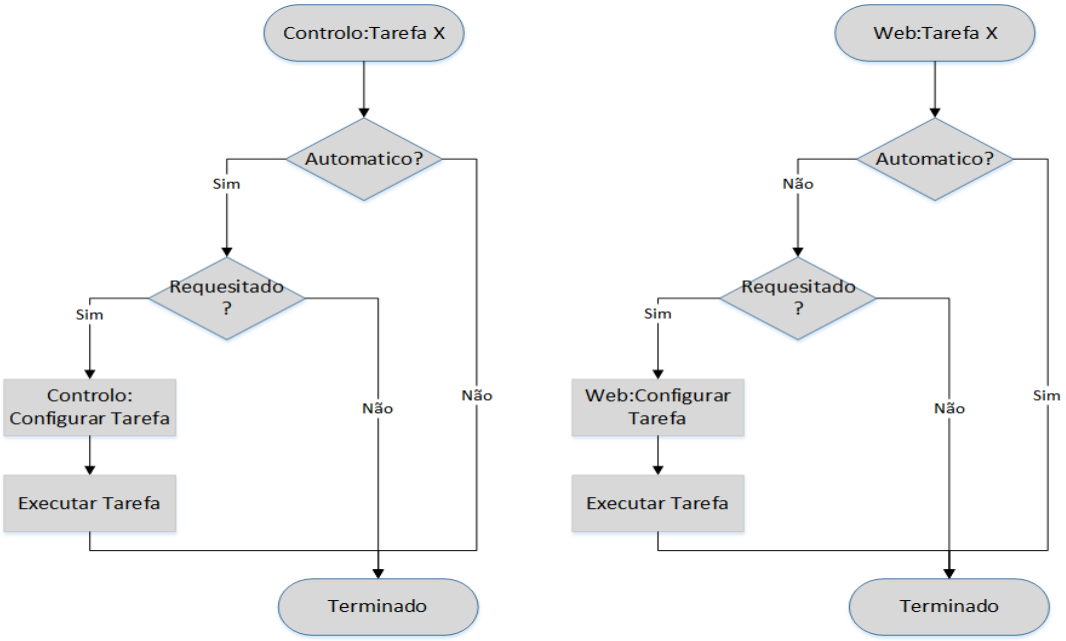

Figura 10 – Controlo de execução de uma tarefa

Um dos benefícios desta implementação é a possibilidade de efetuar manutenção por componentes. Por exemplo, se 50 servidores R estão a processar modelos e uma nova versão do controlo necessita de ser *deployed*, o controlo pode ser terminado e atualizado, que os servidores de processamento continuam a processar e a guardar os seus resultados na base de dados, como consequência apenas os novos processamentos não serão inicializados pois o controlo está desligado. Quando o controlo ficar *online* verifica os estados dos processos e continua como se não tivesse sido desligado. Este benefício não deve ser menosprezado, pois apesar de alguns modelos de *DM* demorarem minutos a ser executados, sem esta arquitetura, o processamento teria que ser abortado, descartando horas de processamento nos modelos mais demorados ou esperar que todos os servidores terminassem o processamento, tendo horas com servidores em espera. No caso da manutenção se realizar nos servidores de processamento, o *DME* continua a responder ao utilizador e executar todas as tarefas que não necessitem de processamento.

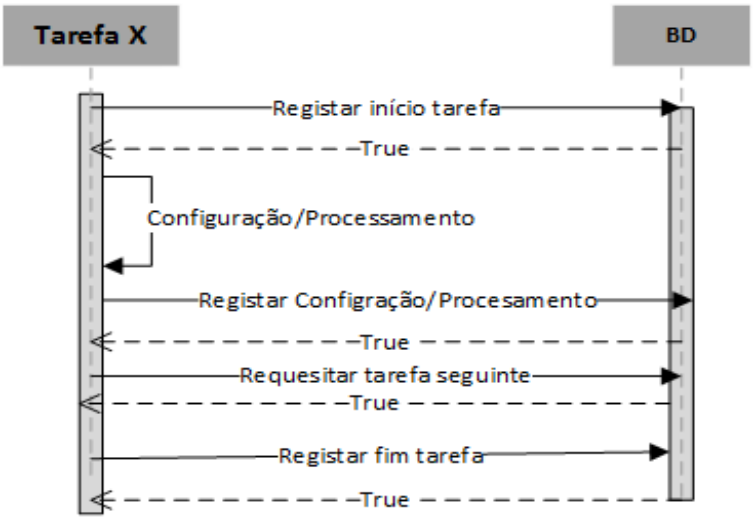

Figura 11 – Registo de estados

Garantia de execução das tarefas

Uma questão ainda não explicada é como o sistema determina quando executar os ExT. Apesar de em alguns casos o processo ser configurado pelo utilizador, maioritariamente ele funciona sem intervenção do utilizador. Para a resolução deste problema um sistema de notificações foi utilizado. Todas as ExT estão disponíveis para ser notificadas (pedido de execução) a partir de qualquer ponto do *DME*. Caso o utilizador tenha iniciado um novo processo pedindo o carregamento inicial dos dados, o pedido é registado na base de dados e o ExT de recolha de dados é notificado pelo componente Web. E sempre que um ExT termina a sua tarefa, notifica o ExT seguinte na ordem do processo. Desta forma é garantido que o processo continua imediatamente ao término da tarefa anterior. Este sistema funciona apenas para situações ótimas onde existem sempre recursos disponíveis e todo o sistema está a funcionar a 100%. Para os outros casos, foi criado uma tarefa que executa os ExT de X em X tempo. Garantindo que as tarefas que não foram possíveis de executar anteriormente (independentemente da razão) sejam executadas.

Concluindo, a arquitetura global para o controlo implementa as seguintes características principais:

- A tomada de decisão é realizada no componente Web pelo utilizador ou no controlo pelo *DME*;
- A execução das tarefas é realizada por um ExT para cada tarefa de segundo nível do *CRISP-DM* para todos os utilizadores em simultâneo;

- Garantia de execução de tarefas e processamento, mesmo com falhas parciais do sistema;
- Esta arquitetura garante novamente a independência dos componentes principais.

a. Controlo de processos

Após a descrição do comportamento global do controlo, nesta secção é descrito de forma detalhada as responsabilidades do Controlo de processos.

Este componente realiza duas tarefas de segundo nível do *CRISP-DM* fase 2. Recolha de dados iniciais, realizando o carregamento dos ficheiros com os dados necessários para o *DM* e a sua descrição através das técnicas disponíveis no R de sumarização, número de linhas dos dados e classes por atributo. Realiza também duas tarefas de segundo nível do *CRISP-DM* fase 3. A seleção dos dados e a sua limpeza, apesar de atualmente apenas permitir que os nulos sejam removidos por defeito, o processo está preparado para no futuro permitir transformações nos dados (por exemplo, criação de classes, conversão de dados textuais em numéricos, entre outros).

Para além da realização de tarefas do *CRISP-DM* este componente é responsável também pela inicialização de novos processos.

Inicialização de um novo processo

Para inicializar um novo processo o controlo de processos, recebe o ficheiro do componente *web*, inicializa o processo, inserindo o correio eletrónico do utilizador, a localização e o nome do ficheiro na base de dados.

Após a criação do novo processo, é inserido na base de dados a requisição de carregamento do ficheiro, com a sua estratégia de divisão (atualmente pré-definida pelo sistema). Esta estratégia define que percentagem é utilizada para treino e validação, bem como se os dados devem manter a ordem ou não.

Após a inserção das configurações desta tarefa, é requisitado no controlo central do processo a sua execução, através da inserção do pedido na base de dados e notificação interna no ExT de recolha de dados iniciais.

Recolha de dados iniciais

Nesta altura o processo inicializado pelo utilizador é requisitado ao sistema e será executado se existir recursos físicos disponíveis, caso não existam, irá ficar em espera até que estejam disponíveis. Na Figura 12 é possível observar este comportamento com maior detalhe. Na figura é demonstrado apenas um cenário em que nenhum problema é encontrado, por questões de espaço e compreensão ao leitor. Nesta figura está representado também o processo de carregamento de dados, desde a sua requisição pelo utilizador, até ao seu término.

Processamento atómico

Como as tarefas de recolha, descrição e transformação dos dados são realizados nos mesmos servidores, estas tarefas são realizadas no mesmo ExT apesar de terem na mesma a sua própria tabela na base de dades e comportarem-se da mesma forma que outro ExT.

Esta questão apresenta um problema adicional à solução apresentada na secção anterior (5.2.2), em que as tarefas de processamento eram divididas em diferentes tipos de servidor. Mas como neste caso as três tarefas serão executadas no mesmo tipo de servidor é necessário considerar questões de atomicidade e tempo de comunicação entre componentes. Por exemplo, considerando que apenas três pedidos podem ser feitos em simultâneo à base de dados, são possíveis as seguintes soluções:

- Atomicidade na base dados

  Solução 1:

  Para garantir que os valores na base de dados são válidos seria necessário alocar os servidores livres imediatamente ao ExT. Esta solução delega a garantia de atomicidade à implementação da base de dados, e como podem ser utilizadas diferentes bases de dados, diferentes comportamentos podiam ser verificados.

  Esta solução necessita também que previamente seja calculado o número de tarefas requisitadas de forma a não reservar mais servidores que o necessário.

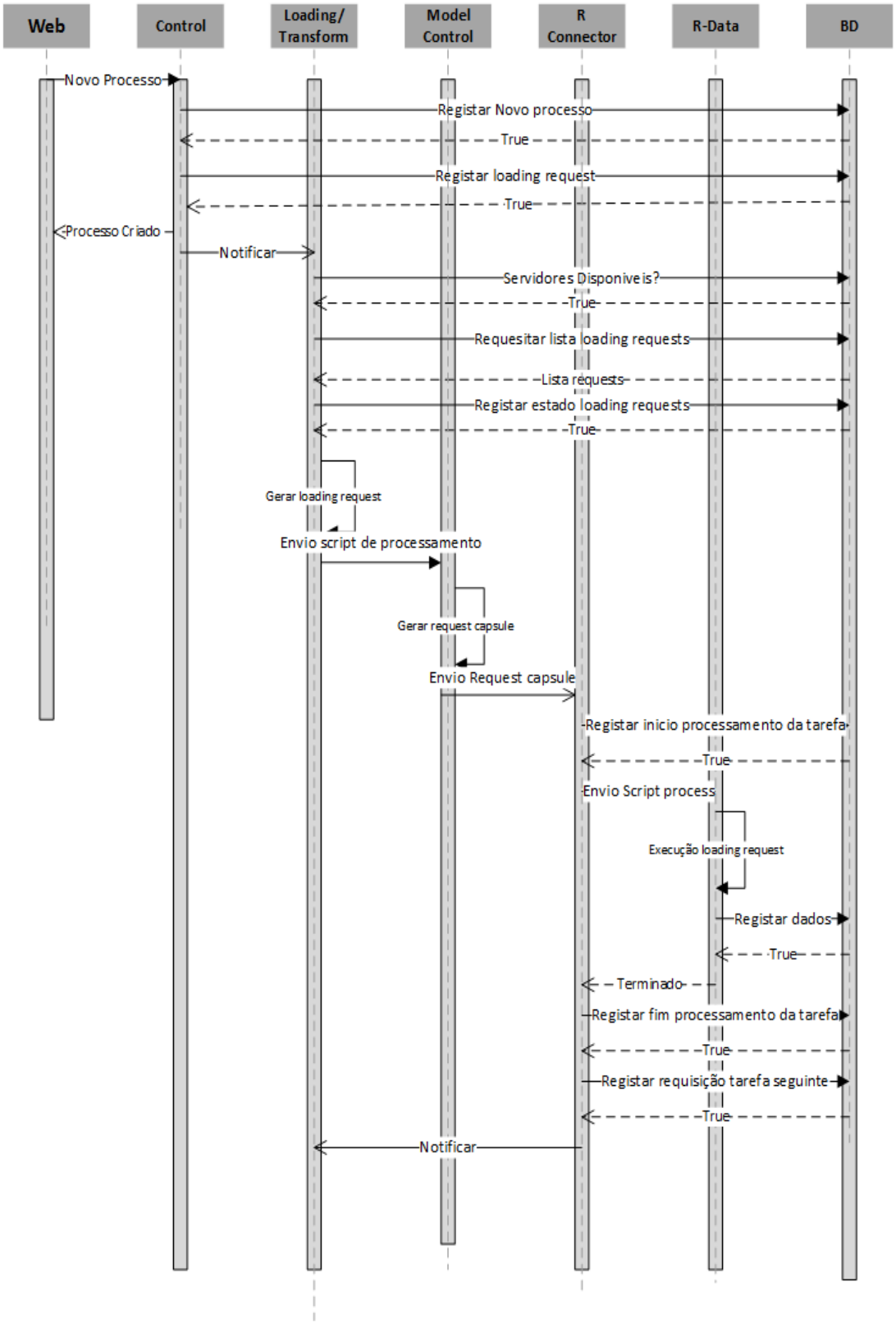

Figura 12 - Execução do carregamento de dados

- Atomicidade no controlo

  Para garantir que duas tarefas não são atribuídas ao mesmo servidor seria necessário a criação de blocos sincronizados que simplificando, garantiam que todo o bloco é executado até ao fim, o que significa que os ExT seriam executados em sequência e não em paralelo.

  Solução 2:

  Este bloco seria aplicado apenas à determinação dos servidores disponíveis. O tempo de resposta da base de dados seria o tempo de espera de cada ExT. Neste caso seria novamente necessário calcular os servidores necessários e alocar no momento da requisição ao ExT que os pediu. Esta solução permite que parte dos três ExT sejam executados em paralelo, no entanto necessita de mais 2 acessos à base de dados que a solução final e tem uma implementação mais complexa.

  Solução 3:

  Junção dos três ExT num só e execução em sequência. Desta forma são recolhidos da base de dados todos os servidores disponíveis, sem necessidade de reservar. Removendo tempo de comunicação dos dois acessos extra das outras soluções e processamento da base de dados. Os servidores são alocados consoante as tarefas disponíveis. As tarefas seguintes utilizam os servidores não utilizados pelas anteriores, por exemplo caso a recolha de dados inicial utilize todos os servidores disponíveis, as outras tarefas nem chegam a ser executadas, reduzindo novamente os acessos à base de dados. Esta implementação permite também definir a prioridade das tarefas de forma global, primeiro a recolha de dados iniciais, seguindo a descrição dos dados e finalmente a transformação dos dados.

A solução 3 é também muito mais simples de implementar e não necessita de utilização de *locks* na base de dados ou no controlo.

Recolha de dados iniciais – continuação

A tarefa será executada pelo ExT que tenha os recursos disponíveis. O ficheiro será carregado para duas novas tabelas na base de dados. O ExT irá gerar um script R, com as informações do ficheiro e do processo e enviar este script para processamento. Este envio é realizado no subcomponente controlo de modelos.

Script de recolha de dados

O script e recolha de dados irá conectar à base de dados e criar duas novas tabelas, uma para os dados de treino outra para os dados de *scoring*.

O ficheiro será carregado para o R, onde serão recolhidas informações dos dados, como número de colunas, número de linhas e o tipo de dados de cada coluna, o número de classes em colunas com dados discretos. Em simultâneo é executada a fase de descrição dos dados.

Estes dados serão guardados noutra tabela já existente na base dados. Realçar que cada processo tem duas tabelas independentes para guardar os dados, mas todo o processo seguinte é guardado em tabelas comuns a todos os processos

A criação de tabelas independentes para os dados, foi definida por duas razões, o número de linhas, pois é bastante comum um processo simples ter milhares de linhas. Mas mais importante, o número de colunas. Como o número colunas e o número de carateres por coluna é diferente para cada processo/ficheiro, seria um enorme desperdício de espaço (por consequência desempenho) para a base de dados, independentemente de adotar uma estratégia de inserção de colunas com tamanho máximo de carateres ou não. Seria uma estratégia muito mais limitativa e ineficiente em termos de recursos. Desta forma o tamanho de cada tabela é ajustado ao ficheiro.

As instruções para guardar estes dados na base de dados são também enviadas no script. O R guarda os dados diretamente na base de dados, esta estratégia reduz a comunicação entre processamento e controlo e como os ficheiros também podem ter volumes enormes, desta forma mantem-se a lógica definida para cada componente, o controlo para configurar/decidir e o processamento para processar.

Após a execução do carregamento do ficheiro é requisitado ao utilizador que este selecione o target e as suas métricas. Esta parte do processo é descrita na componente Web.

Após a seleção do target é requisitado as transformações, que são executadas exatamente da mesma forma que o carregamento do ficheiro. Como já foi referido anteriormente, atualmente esta parte do processo remove apenas automaticamente todos os nulos dos dados.

É também neste componente que se encontra o Bean que executa todos os ExT do sistema. Alterando o timer neste componente altera-se a frequência que os processos são executados.

Resumindo este componente é responsável por:

- Iniciar novo processo;
- Registar e formular o pedido de carregamento/descrição do ficheiro;
- Registar e formular o pedido de transformações;
- Controlar a periocidade dos ExTs.

b. Controlo de modelos

Responsável por duas tarefas de segundo nível da quarta fase do *CRISP-DM* (Modelação), seleção da técnica de modelação e geração *design* de teste.

Um dos componentes mais complexos do *DME* é responsável pelo funcionamento dos modelos, nomeadamente da ferramenta R, implementa as classes responsáveis pela inserção de novos modelos, faz a sua seleção e configuração, bem como a geração do script de execução e *scoring*. É responsável também pelo envio de todos os pedidos para os servidores R.

No subcomponente anterior a descrição do funcionamento do *DME* ficou na definição das métricas do target, que será descrito na secção do componente Web.

Seleção de modelos

Após a definição das métricas do target, a próxima tarefa a realizar é a seleção de modelos, apesar de atualmente esta seleção ser predefinida e não utilizar qualquer processamento, é esperado em versões futuras a utilização de processos de SSD, nomeadamente para a determinação de prioridades de execução dos modelos baseados no tempo de execução e memória necessária.

Teste design

Esta tarefa é a única que foi alterada na ordem de execução em relação ao *CRISP-DM*. A necessidade desta alteração remete aos requerimentos associados a *pervasive computing*. Um dos requisitos é o número mínimo de interação com o utilizador, e como o tempo de processamento das tarefas (e de espera), pode superar o da atenção

do utilizador, todas as interações com o utilizador devem ser realizadas no início e o mais rápido possível. Apesar de atualmente muitas das tarefas não executem o processamento, conceptualmente é necessário fazer esta alteração. Desta forma o utilizador configura todo o sistema inicialmente e não requer mais intervenção do utilizador a não ser que o pretenda fazer.

Apesar de a definição do teste *design* ser realizada no início, a recolha das métricas para avaliação é realizada após a criação do modelo, como pode-se verificar na Figura 13. A vantagem da recolha das métricas só ser realizada no fim da criação do modelo, permite que as modificações do utilizador nas métricas tenham um efeito imediato.

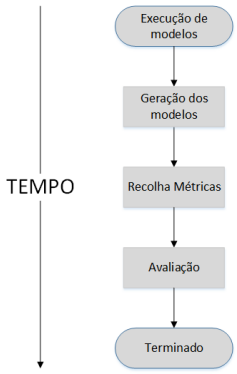

Figura 13 – Execução de modelos

Por exemplo podem ser considerados dois modelos, um demora 6h e o outro 2h, e que o primeiro modelo foi iniciado antes das alterações, o segundo foi iniciado após as alterações. Na perspetiva do utilizador, o primeiro modelo termina depois do segundo, logo o utilizador não vai perceber porque a alteração das métricas aplicou-se num modelo que terminou antes do outro. Estas questões são extremamente importantes, as características *pervasive* implicam que o funcionamento da ferramenta tem que corresponder às espectativas do utilizador.

Atualmente a determinação do teste *design* é realizada pelo utilizador, inserindo os valores das métricas pretendidas. As métricas atualmente disponíveis são:

- Regressão;
    - *RAE – Relative absolute error;*
    - *MAE – Mean absolute error;*
    - *RSE – Relative square error;*

- Classificação;
    - *Precisão;*
    - ACC – Acuidade;
    - TFN – Taxa de falsos negativos;
    - TFP – Taxa de falsos positivos.

Configuração de modelos

Para a realização da configuração de modelos em modo automático, são utilizados os valores do modelo por defeito, e variações dos intervalos definidos. Atualmente está definido por defeito no sistema, que o intervalo será multiplicado por três, e será subtraído e somado ao valor de defeito, gerando assim 3 combinações possíveis para cada atributo de configuração. Por exemplo na Tabela 8, para o atributo 1, os valores 2, 5 e 8 seriam gerados, para o atributo 2 seria 0.35, 0.5 e 0.65 (estes valores são arbitrários e não representam uma técnica real). Esta estratégia é apenas uma implementação temporária para gerar configurações seguindo uma lógica, sendo substituída em versões futuras do *DME*.

Tabela 8 – Exemplo valores dos atributos de configuração

Atributo	Defeito	Intervalo
Atributo 1	5	1
Atributo 2	0.5	0.05

Como já foi descrito anteriormente, automaticamente são geradas três configurações para cada atributo, caso a técnica tenha mais que um atributo, mais configurações podem ser geradas como podemos verificar na Tabela 9.

Tabela 9 – Exemplo do número de configurações geradas automaticamente

Técnica	Nº atributos de configuração	Configurações geradas
Svn	3	3x3x3=27
Rpart	2	3x3=9
naiveBays	1	3

Pode-se verificar na Tabela 10 as configurações geradas para os valores descritos na Tabela 8, estas configurações são guardadas na base de dados para ser executadas posteriormente. Os valores apresentados nestas três tabelas são exclusivamente utilizados como exemplo, não representando a implementação atual.

Tabela 10 – Exemplo de configurações geradas automaticamente

Configuração	Atributo 1	Atributo 2
1	2	0.35
2	2	0.5
3	2	0.65
4	5	0.35
5	5	0.5
6	5	0.65
7	8	0.35
8	8	0.5
9	8	0.65

Após o registo das configurações o ExT é requisitado a execução de modelos, e registado o término da configuração de modelos.

Esta configuração também pode ser realizada manualmente, o utilizador introduz as configurações que pretende e são registadas normalmente na base de dados.

Execução de Modelos

A execução dos modelos já foi explicada na descrição do componente de processamento, a necessidade de um conector e a forma como são executados os scripts, no entanto ainda não foi mencionado como são criados estes scripts de modelação.

- **Modelos (Técnicas)**

Foi inicialmente definido que por questões de tempo apenas seriam implementadas três técnicas de modelação, garantindo apenas que a regressão e classificação estavam presentes em uma das três técnicas. Com esta definição em mente a primeira tentativa foi executada.

- **Primeira tentativa**

Uma classe (implementação) individual para cada técnica. Foi identificado inicialmente que seria ineficiente, mas como seria necessária para servir de base para a criação de uma implementação que suporta qualquer técnica R foi implementada até ao fim. A análise do funcionamento da primeira implementação foi o ponto de partida inicial para a segunda implementação. Foi possível identificar as partes comuns de todo o processo e as suas possíveis divisões. Partes do script de modelação são completamente iguais entre modelos. A criação de uma classe para cada modelo não só seria ineficiente pela replicação de código já existente, como alterações futuras no código implicava alterações em todas as classes. A introdução de novos modelos implicava o *deployment* de uma nova versão do *DME*. Esta implementação foi então descartada por uma implementação que suporte todos os modelos baseados em R.

- **Implementação final**

A implementação final permite que sejam introduzidos novos modelos a qualquer momento e apesar de ser de maior complexidade, é uma solução muito mais elegante que pode ser utilizada no futuro, ao contrário da anterior que seria apenas uma solução possível para protótipo.

Esta implementação está dividida em 6 partes principais:

- Acesso à base de dados
- Carregamento dos dados
- Estratégia de divisão
- Modelação
- Avaliação
- Limpeza

Após a primeira implementação ficou identificado que todas fases excetuando a modelação eram iguais para todos os modelos. Para a realização desta estrutura duas classes principais são utilizadas (*RBaseScript*, e *Rmodel*).

RBaseScript

Para cada configuração na base de dados, no momento de execução é gerado um objeto da classe *RBaseScript*, este objeto é responsável pela criação do script de modelação. Responsável pelas cinco partes principais excetuando a modelação.

- Acesso à base de dados

  Como referido anteriormente, este código é comum a todos os modelos, e permite que caso sejam necessárias alterações no acesso à base de dados, estas são efetuadas nesta secção afetando imediatamente todos os modelos ainda por executar (mesmo configurados previamente).
- Carregamento dos dados

  Este código é idêntico para todos os modelos, existe apenas um código extra para a classificação para garantir que os dados das colunas são de facto dados do tipo correto, e interpretados pelo R como fator.
- Estratégia de divisão

  Atualmente implementa apenas duas estratégias de divisão, *nFold* e *standard*. Futuras estratégias de divisão podem ser implementadas neste ponto, ficando disponíveis imediatamente para todos os modelos já inseridos no sistema. Todas estas funcionalidades seriam impossíveis na primeira implementação. Na Figura 14 pode-se verificar como é possível realizar esta divisão.

```
a1DbAcess();
a2LoadingData(columns, databaseName);
if (divisionMethod.equals("nFold")) {
    a3DataDivisionNfold(this.trainingPercOrNFold);
} else if (divisionMethod.equals("standard")) {
    a3DataDivisionStandard(this.trainingPercOrNFold, eval
}
a4ModelCode();
if (this.method.equals("regression")) {
    a5EvaluationRegression();
} else if (this.method.equals("classification")) {
    a5EvaluationClassification();
} else {
    db.DbConnection.getInstance().saveErrorToDB("RBaseScr
}
a6Clean();
```

Figura 14 – Exemplo funcionamento *RBaseScript*

- Avaliação

  Avaliação tem métricas completamente distintas para regressão e classificação, como tal têm a sua secção distinta para implementação. A implementação de avaliação de *clustering* por exemplo, não afeta as implementações de outras avaliações existentes.
- Limpeza

  Responsável pela limpeza de todas as variáveis de forma a garantir que os processos seguintes a ser executados no servidor não são afetados por processos anteriores.

RModel

Na Figura 14 é possível verificar a chamada do método "*modelCode*" no *RBaseScript*, este método requisita o código específico da técnica a ser executado. *RModel* é responsável apenas pelo código específico de cada técnica, e as suas configurações. É utilizado previamente pelo ExT configuração de modelos, para gerar as configurações para os modelos selecionados. Nesta fase o RModel recebe as configurações de cada modelo e gera o código R de execução da técnica.

Toda esta implementação está assente numa estrutura de palavras-chave. Todos os dados dos modelos são carregados para as mesmas palavras-chave, as configurações dos modelos são guardadas por palavra-chave, etc. Por exemplo "__target__" é a palavra-chave referente ao atributo target. Esta palavra-chave será substituída na altura de criação do script pelo atributo dos dados referentes ao target do processo a ser executado, o mesmo se aplica para todos os atributos de configuração por exemplo.

Apesar da execução e avaliação poderem ser realizados em separado a inutilidade do modelo sem avaliação e o acréscimo de processamento desnecessário, tornam esta separação não recomendável.

No momento de avaliação é ainda verificado se as métricas de avaliação do processo foram atingidas e é requisitado o ExT de *scoring*.

Scoring

O processo de *scoring* é muito semelhante ao "*RBaseScript*", mas numa escala muito mais pequena. Tem também um código específico para regressão e classificação e um objeto para cada pedido.

Após a execução da modelação e avaliação do modelo, os resultados são guardados pelo R na base de dados e o conector notifica este módulo que terminou a sua execução. Nesta altura caso a avaliação do módulo tenha atingido as métricas definidas é gerado um script de *scoring* no controlo de modelos e enviado para um servidor do tipo *R-Scoring*, para posteriormente serem visualizados os resultados no módulo Web.

c. Serviços

Módulo de suporte, criado por questões de implementação, realiza operações como alteração de estado de cada fase e tarefa dos processos, recolhe informação sobre os processos, modelos, etc. Garante que as operações comuns entre módulos são realizadas com consistência e num único local.

A necessidade de criação deste módulo advém das limitações/necessidades do *Java EE*. O *java EE* não permite que existam chamadas em *loop* entre os seus módulos, como tal a divisão de alguns processos globais tem que ser executadas num módulo diferente que não executa chamadas a mais nenhum módulo.

d. Sistema de suporte à decisão

Modulo apenas parcialmente desenvolvido e ainda não completamente funcional. Realiza atualmente apenas *Extract Tramsform and Load* (*ETL)* para o futuro processo de *DM* com o objetivo de prever o tempo de execução de cada modelo de *DM*. O objetivo deste módulo é ser responsável pelos processos de tomada de decisão interna do *DME*. Apesar de apenas realizar parte do *ETL* necessário para a previsão do tempo de execução foi possível determinar algumas das necessidades para o seu funcionamento e de outros processos internos do sistema de suporte à decisão. Todos os processos necessitarão de *ETL*, ou alterações à base de dados para facilitar a organização dos dados, independentemente da estratégia a execução de *ETL* será necessário. Um controlador especial para os processos internos, que definirá a estratégia para a geração de novos modelos será criado de modo a perceber que

modelo será utilizado para fazer as previsões. É um processo prioritário, com foco na realização do *scoring*, para que o sistema não demore a tomar decisões.

5.2.4. Interface

O interface é responsável pela interação com o utilizador e permite executar a maior parte das tarefas realizadas pelo sistema, manualmente pelo utilizador. Atualmente o *DME* é composto pelas seguintes componentes *Web*:

- **Registo**

  Implementação *web* que permite ao utilizador o registo no sistema inserindo um nome, correio eletrónico e palavra-chave.

- **Login**

  Implementação *web* que permite ao utilizador que faça o login no sistema, inserindo o seu correio eletrónico e palavra-chave.

- **Carregamento de dados**

  Permite ao utilizador que insira um *dataset* para ser carregado para o sistema e inicializado um novo processo. Esta pagina *web*, apenas recebe o ficheiro, guarda-o e requisita ao controlo o carregamento para a base de dados e a criação de um novo processo. O carregamento do *dataset* é realizado no controlo e processamento.

- **Informação do processo**

  Esta Implementação *web* permite ao utilizador verificar o estado atual do processo. Por exemplo, se o carregamento de dados está à espera ou se está a ser executado, se necessita de configuração, etc. Caso necessite de intervenção do utilizador a pagina redireciona para a tarefa em questão.

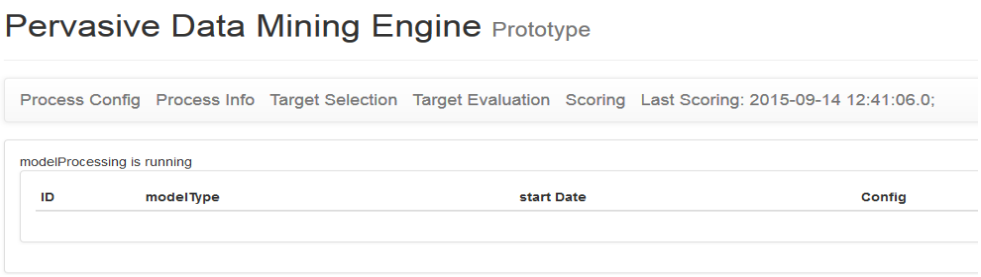

Figura 15 – Informação do processo

- **Seleção de processo**

  Implementação de suporte que permite ao utilizador selecionar um de vários processos que pode ter no sistema. De realçar que todos os processos estão a ser executados em simultâneo, simplesmente apenas se pode configurar um de cada vez.

- **Configuração do processo**

  Implementação de suporte que permite ao utilizador definir que tarefas pretende que o sistema realize automaticamente. É possível definir que a seleção de modelos seja realizada automaticamente e a sua configuração seja definida manualmente pelo utilizador ou vice-versa.

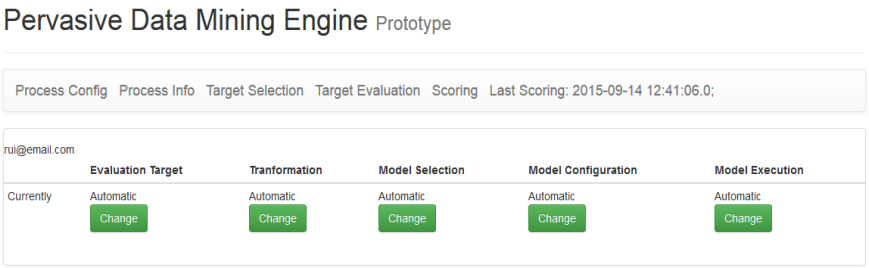

Figura 16 – Configuração do processo

- **Seleção do target**

  Esta Implementação disponibiliza informação sobre o *dataset* introduzido pelo utilizador, nomeadamente o nome dos campos e o seu tipo de dados. O utilizador seleciona um dos campos como o *target* do processo de *DM*.

- **Avaliação do target**

  Esta Implementação permite definir que métricas são objetivo do processo de DM e os seus valores, é uma das tarefas mais importantes do sistema pois define quando o processo de *scoring* será executado, e o objetivo do processo é atingido. Como pode ser visualizado na Figura 17 o utilizador pode selecionar a métrica e a lógica que pretende. Pode também fazer uma combinação de várias métricas por exemplo “RRSE <50 e RAE <150”. Quando estas métricas forem atingidas o sistema irá realizar o *scoring* para o modelo em questão e mostrar os resultados ao utilizador.

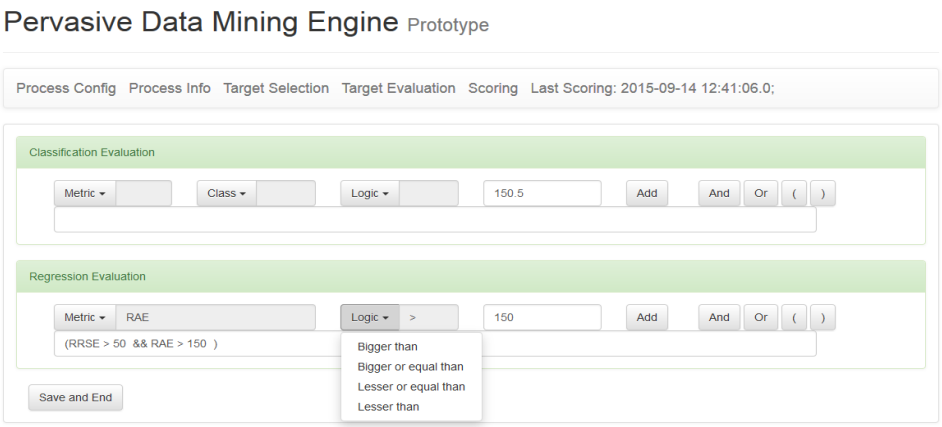

Figura 17 – Avaliação dos modelos

- **Seleção dos modelos**

  Esta Implementação permite ao utilizador selecionar os algoritmos que pretende utilizar, por exemplo, rpart, naivebays, etc.

- **Configuração dos modelos**

  Esta Implementação permite ao utilizador configurar os modelos previamente selecionados, seja pelo utilizador ou pelo *DME* (cada tarefa é independente). As configurações disponibilizadas dependem do modelo selecionado. Esta página não está atualmente a funcionar, porque não houve tempo para a conversão para *Java EE*.

- ***Scoring***

  Esta Implementação permite ao utilizador visualizar os resultados do *scoring*.

6. Processo de investigação

Como foi descrito na secção 1.3, esta dissertação utiliza a metodologia de investigação *Design Science Research*. Até agora apenas foi apresentado o resultado final da investigação, neste capítulo será apresentado as fases e os passos realizados para chegar ao resultado final.

6.1.1. Iteração 1

Na fase inicial do trabalho o conhecimento é extremamente reduzido, sendo difícil de prever as necessidades.

- Consciencialização do problema
  Criação de scripts R – Inicialmente o conhecimento de scripts R era muito limitado, e as implicações não eram conhecidas.
- Sugestão
  Criação de scripts para o carregamento e descrição dos dados, o processo de modelação e o processo de *scoring*.
- Desenvolvimento
  Scripts desenvolvidos diretamente no R, sem a utilização de qualquer outra tecnologia. O resultado são vários scripts que englobam todo o processo de *DM*. É um processo demorado, pois já foi desenvolvido considerando que todos os dados eram voláteis (número de colunas variável por exemplo).
- Avaliação
  Todo o processo é possível de realizar no R, este facto já era conhecido. O principal objetivo desta iteração era adquirir conhecimentos na tecnologia R, e preparar já os scripts para o processo automático.
- Conclusão
  É possível verificar que o processo pode ser divido em partes independentes. É identificado também as necessidades para a configuração manual.

6.1.2. Iteração 2

- Consciencialização do problema
  Com os scripts criados manualmente e introduzidos manualmente é necessário que sejam executados automaticamente.

- Sugestão

Criação de um servidor R que permita receber e executar scripts R automaticamente.

- Desenvolvimento

  Após a pesquisa inicial foi identificado um *plugin* para o R desenvolvido exatamente para a comunicação com o *java*. Este *plugin* permite inicializar vários servidores no mesmo computador, e comunicar diretamente utilizando *java*, não necessitando de mais nenhum intermediário.

- Avaliação

  Os resultados foram melhores que o previsto. Inicialmente previa-se que seria necessário desenvolver a comunicação entre *java* e o R, mas o *plugin* resolveu o problema.

- Conclusão

  É possível disponibilizar o R como servidor com relativa facilidade e comunicação direta com o R. As responsabilidades do conector (para todas as ferramentas) são definidas:

  - Receber as instruções do controlador;
  - Inicializar e monitorizar os servidores da ferramenta;
  - Enviar e monitorizar os processos nos servidores da ferramenta.

6.1.3. Iteração 3

- Consciencialização do problema

  Com o R a realizar o processo automaticamente é necessário controlar estes processos.

- Sugestão

  Desenvolver em *java* o controlo sem interface gráfico mas já com divisões das tarefas por métodos.

- Desenvolvimento

  Objetivo deste desenvolvimento é terminar com uma ferramenta que execute o processo do início ao fim sem a introdução de qualquer parâmetro. É nesta fase também que o maior desenvolvimento da base de dados é realizado, pois já são guardados na base de dados, todos os resultados do processo, mas ainda não são guardadas as configurações.

- Avaliação

O processo correu como esperado, e o *java* revelou-se como o meio mais adequado para realizar esta parte do processo.

- Conclusão

  Este passo apresenta as conclusões mais importantes de toda a dissertação. Com o processo a correr automaticamente, e com divisão de tarefas é possível confirmar que todo o processo *DM* pode ser executado automaticamente e manualmente, tarefa a tarefa. As únicas implicações em falta é a escalabilidade, para uma ferramenta para utilizador único, no entanto está provado neste momento que é possível.

  O objetivo nesta altura era confirmar que o processo poderia ser executado automaticamente e manualmente, para garantir este objetivo foi proposto a independência das tarefas. Quando é equacionado a escalabilidade, a independência de componentes é visionada. As diferentes necessidades das tarefas são visualizadas e a divisão de componentes no controlo é criada agrupando as tarefas de acordo com as necessidades. Este será o próximo passo.

  É também conclusão desta iteração a necessidade de processos (*threads*) que executam tarefas multicliente.

6.1.4. Iteração 4

- Consciencialização do problema

  Com o processo a ser executado para um cliente é necessário converter para uma versão multicliente.

- Sugestão

  Remodelação do código existente para suportar multiutilizador e diversos componentes.

- Desenvolvimento

  O principal desenvolvimento desta fase é a transformação do código para multicliente, nascendo as divisões dos componentes conforme as necessidades.

  Nesta fase a base de dados é reformulado para permitir a persistência das configurações e definições do *DME*.

  Outro desenvolvimento importante é a alteração na abordagem à introdução de algoritmos de *DM* no *DME*.

- Avaliação

  Em termos de avaliação nada poderia prever os resultados obtidos neste desenvolvimento.

  Diversas pequenas iterações foram equacionadas, experimentadas, analisadas e avaliadas imediatamente. Em termos de desenvolvimento esta foi a iteração mais importante. Toda a estrutura dos ExT foi equacionada nesta iteração apesar da implementação e definição final só ter sido executado na iteração 5. Inicialmente cada algoritmo teria a sua implementação em *java* (opções de configuração, geração de código etc), o que se verificou terrivelmente ineficiente. Foi então desenvolvido uma implementação em *java* que suporta qualquer algoritmo R (explicado na secção.

- Conclusão

  É concluído nesta iteração que os subcomponentes do controlo necessitam de divisões físicas. Apesar de na iteração 2 e 3 já se visionar que o controlo também necessitaria de ser dividido fisicamente, nesta iteração é confirmado que é viável esta divisão. É também definido praticamente a arquitetura final do projeto.

6.1.5. Iteração 5

- Consciencialização do problema

  Com o processo a funcionar com vários utilizadores, dois passos finais estavam em falta, interface, e escalabilidade dos componentes do controlo. A escalabilidade dos componentes não estava previsto no início do projeto, mas as suas implicações para a solução eram demasiado importantes para ignorar. O protótipo estaria muito muito longe das previsões para trabalho futuro se este problema não fosse equacionado nesta altura.

- Sugestão

  Inicialmente foi definido fazer criação do interface, o que se verificou um erro, mas a falta de conhecimento em *java EE* justifica este erro.

- Desenvolvimento

  Após o início do desenvolvimento do interface, é fácil de verificar que o *java EE* simplificaria todas as comunicações entre o controlo e interface, e que as existentes seriam obsoletas.
- Avaliação

  A iteração foi abandonada pois seria ineficiente terminar tudo, para no fim ser refeito.
- Conclusão

  Iteração necessária para entender a necessidade de introdução de java *EE* e as suas potencialidades, o erro foi não ter abandonado mais cedo o desenvolvimento.

6.1.6. Iteração 6

- Consciencialização do problema

  A comunicação entre o controlo e interface seria extremamente complexa sem a utilização de *Java EE,* e a escalabilidade teria que ter a sua implementação também.
- Sugestão

  Conversão de todo o controlo e interface para *Java EE*. A conversão resolve os problemas identificados na iteração 5 e os problemas de escalabilidade identificados na iteração 4. Nesta fase é também concluído o interface *web*.
- Desenvolvimento

  Um processo relativamente demorado visto que o conhecimento em *Java EE* era zero, e todo o desenvolvimento da iteração teria que ser adaptado, principalmente os ExTs. Outro problema era o avançado desenvolvimento do interface, que ditou que algumas das opções não estejam ainda ativas.
- Avaliação

  Uma reformulação absolutamente necessária, que garante a escalabilidade do controlo como era previsto.
- Conclusão

  Nesta iteração foi finalmente confirmada a independência dos componentes. Cada um dos componentes pode ser executado no seu próprio servidor. O *DME* é capaz de realizar todo o processo automaticamente/manualmente, e garantindo as características *pervasive*.

7. Caso de estudo

Como forma de efetuar um primeiro estudo da viabilidade da solução utilizando dados reais foi criado um pequeno caso de estudo utilizando dados provenientes da Unidade de Cuidados Intensivos do Centro Hospitalar do Porto. Este estudo foi meramente experimental, não sendo os resultados atingidos relevantes para a discussão mas sim a possibilidade de executar too o processo automaticamente. Para a realização dos testes foi utilizado um *dataset* com 214709 registos, 32 colunas, com a informação dos sinais vitais dos doentes. O *target* deste *dataset* era a coluna "crítico", sendo este um campo numérico com duas possibilidades (1 – critico, 0 – estável). O *DME* faz o carregamento dos dados automaticamente, e o utilizador seleciona a coluna "crítico" como target. Como o número de possibilidades é reduzido, ou seja, existem duas classes de saída a técnica utilizada foi a classificação. O utilizador definiu as métricas de avaliação, para este exercício como:

- Classificação
  "ACC > 78"

O sistema automaticamente seleciona o modelo *naiveBayes* (classificação). Após a seleção automática o sistema gera as configurações, neste caso 494 configurações foram criadas.

Após conclusão da geração destes modelos, foram obtidos 10 modelos de classificação que cumpriram as métricas, com o melhor modelo de regressão um ACC de 78.63. O *scoring* foi posteriormente realizado para apenas estes modelos, e os resultados apresentados ao utilizador (em formato de % - probabilidade de um doente ter um valor crítico).

Este exercício permite comprovar que o *DME* funciona, e permite obter resultados relevantes, sem conhecimento aprofundado em *DM* e sem grande intervenção do utilizador.

8. Discussão

O protótipo atual permite provar que um processo de *DM* pode ser executado automaticamente desde a recolha dos dados à avaliação dos resultados, mantendo as características de um *DME* normal, continuando a permitir a um utilizador mais experiente fazer as configurações necessárias. A capacidade de executar um processo do início ao fim automaticamente, ou com decisões manuais pelo utilizador, provam a questão de investigação. É um facto que não faz todas as operações definidas no *CRISP-DM*, mas o objetivo não era provar que consegue realizar todas as tarefas descritas no *CRISP-DM*, mas sim a viabilidade de executar um processo *DM* completo. Os principais processos estão incorporados e a funcionar, e a implementação das tarefas em falta, não é uma questão de viabilidade mas sim porque estes fazem parte da segunda versão da solução.

A definição da arquitetura, principalmente a independência dos componentes permite garantir as principais características *pervasive*. Resolvendo problemas de escalabilidade, mas principalmente garante que será sempre possível adaptar a ferramenta para corresponder às necessidades dos utilizadores, sendo este o maior contributo para questões *pervasive*. Por exemplo se for necessário realizar uma tarefa que o R não consegue executar, outra ferramenta pode ser implementada para executar essa tarefa específica. As necessidades dos utilizadores hoje não são as mesmas de amanhã e as ferramentas têm que ser capazes de se adaptar.

Tendo em conta os resultados obtidos no caso de estudo, que foram bastante satisfatórios e como forma de fazer uma avaliação mais ampla da solução desenvolvida, atualmente estão a ser induzidos modelos de *DM* já processados em outras ferramentas de *DM* de modo a avaliar a potencialidade do *PDME*.

9. Conclusão

Os objetivos desta dissertação foram atingidos. Ficou provado a viabilidade de um sistema de *DM* semiautónomo com características *pervasive*, sem perder a sua identidade e funcionalidades. Esta ferramenta é capaz de "executar" as fases do *CRISP-DM* de forma automática e manual, estando a solução disponível a partir de qualquer lugar em qualquer sistema e com o mínimo de interação possível com o utilizador. A execução do processo do início ao fim é a prova desta afirmação.

Sendo este um trabalho de investigação foram várias as áreas abordadas na realização deste trabalho. É de salientar a multidisciplinariedade de conceitos e abordagens:

- *Data Mining*;
- Sistemas distribuídos;
- Processamento paralelo;
- Base de dados;
- *Java*/*Java EE*;
- Tecnologias *Web*;
- Ubiquidade / *Pervasive*;
- R.

Todos estes conceitos/abordagens/ferramentas foram abordados separadamente durante o curso, excetuando *Java EE* e *pervasive*, conceitos novos desenvolvidos durante esta dissertação. Apesar de muitos conceitos já terem sido abordados nunca o foram desenvolvidos num só projeto, e em conjunto. O entrosamento de todos estes conceitos foi um dos grandes desafios deste projeto.

Os resultados obtidos são extremamente animadores nomeadamente para o futuro bem como as possibilidades que este *DME* pode oferecer à comunidade. A curto prazo, este *DME* pode ser utilizado nas universidades como ferramenta de ensino. O registo de todas as instâncias do processo permite, sem ser necessário um conhecimento aprofundado de como tudo funciona (por parte do utilizador), analisar as diferenças de acordo com as decisões tomadas e que decisões são necessárias para um correto processo de *DM*. A longo prazo a ferramenta pode finalmente ser utilizada para a criação de modelos dinâmicos, permitindo a outras ferramentas de qualquer área (não *DM*), utilizar os serviços deste *DME* para tomar as suas decisões.

10.Trabalho futuro

Sendo este um trabalho de continuidade e inovador, com esta dissertação foi possível desenvolver um protótipo capaz de executar por completo um processo de *DM*. As fases seguintes deste projeto surgem na sequência do trabalho realizado até ao momento e estão presentes no plano de investigação para a versão 2 do projeto. Assim foi gerado um plano de trabalho de continuidade tendo como principais ideias:

Sistema de suporte à decisão

A principal área de pesquisa no futuro, apesar de já ter sido inicializada nesta dissertação, não foi possível provar a exequibilidade dos processos de DSS. Para trabalho futuro será necessário a implementação de processos internos de *DM*. Alguns destes processos são necessários para o bom funcionamento do *DME*, outros são necessários para aumentar a eficiência do *DME*.

Tempo execução

Previsão do tempo de execução de um modelo, não é só importante para o utilizador, como também é extremamente importante e necessário para uma eficiente gestão dos recursos do *DME*. Considerar um *DME* com três servidores R e apenas um processo, inúmeras configurações são geradas, mas quais devem ser executadas em primeiro? O tempo é sempre relevante, como determinar quais os modelos que cumprem as necessidades do utilizador? E se forem alocados três modelos que demoram dois dias a executar e um segundo processo é iniciado? Um processo interno de previsão de tempo é absolutamente necessário, e prioritário.

Memória necessária

A previsão da memória utilizada pelo R para executar os modelos é necessária para garantir que o servidor não fica sem memória e termina o processo, ou caso esteja configurado para utilizar o disco rígido, como memória primária, diminui significativamente a performance do servidor R. É também um processo extremamente prioritário pois o impacto não (pode) é só no modelo em causa, mas como existe (pode) partilha dos recursos entre os servidores lógicos, vários servidores podem terminar abruptamente por falta de memória.

Estes dois processos são prioritários, mas por si só não resolvem todos os problemas, como em todos os processos de *DM*, são apenas previsões, uma estratégia de gestão de risco, e resolução de problemas relacionados com estas questões são também necessárias.

Que modelo executar primeiro?

A primeira questão a ser feita e por responder até hoje. Sim, existem imensas configurações mas qual executar primeiro? Uma possível resposta é, qual de todas as configurações geradas se prevê atingir as métricas definidas previamente.

Previsão das avaliações das métricas

Por exemplo todos os processos executam sempre em primeiro a configuração com os valores de defeito. Utilizando as métricas atingidas neste passo, e as observações em modelos anteriores, as variações nas métricas podem (possivelmente) determinar a métrica para cada configuração. Este processo pode não ser possível, ou pode simplesmente necessitar de mais execuções. É uma possível estratégia a considerar.

Previsão importância dos atributos

No processo de *DM* diferente atributos têm diferentes pesos no resultado final, a remoção de atributos previamente determinados como (possivelmente) irrelevantes podem tornar processos demorosos (devido ao elevado número de atributos), em processo exequíveis temporalmente e em termos de memória, ou determinar se uma prioridade em modelos mais intensivos será valorizado significativamente. Este processo possivelmente será baseado nas instâncias recolhidas na fase de descrição dos dados, ou possivelmente depois da geração de alguns modelos mais rápidos.

Base de dados

A escalabilidade da base de dados seria impossível de investigar nesta dissertação, as necessidades finais do *DME* ainda não são totalmente conhecidas. Como foi referido anteriormente, o *DME* adapta-se às necessidades. Diferentes estratégias de escalabilidade serão necessárias, possivelmente não existirá uma que comporte todas as possibilidades. A necessidade de pesquisa da estratégia de base de dados será necessário quando o desenvolvimento do *DME* estiver numa fase mais estável e aplicado a uma situação mais específica.

Novas ferramentas

Apesar da ferramenta R implementar imensas técnicas de modelação e fornecer imensas funcionalidades, não será a solução para todos os problemas, novas ferramentas, ou problemas que o R não consegue resolver ou é ineficiente, poderão ser resolvidos com a implementação de novas ferramentas.

Novos tipos de modelos

DME atualmente apenas implementa regressão e classificação, a implementação de *clusters* é o próximo passo logico, principalmente porque também pode ser utilizado para processos internos no DSS, a possível caracterização dos *datasets* em classes pode ser necessária, mesmo não sabendo o seu significado, a divisão em classes pode ser relevante para outros processos internos, as possibilidades existem, embora não podendo comprovar até a sua implementação.

11.Referências

Araya, A. A. (1995). Questioning ubiquitous computing. In *Proceedings of the 1995 ACM 23rd annual conference on Computer science* (pp. 230–237).

Azevedo, A. I. R. L. (2008). KDD, SEMMA and *CRISP-DM*: a parallel overview.

Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., & Zukowski, D. (2000). Challenges: an application model for pervasive computing. In *Proceedings of the 6th annual international conference on Mobile computing and networking* (pp. 266–274).

Bradley, P. S., Fayyad, U. M., & Mangasarian, O. L. (1999). Mathematical programming for *data mining*: formulations and challenges. *INFORMS Journal on Computing*, *11*(3), 217–238.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). *CRISP-DM* 1.0 Step-by-step *data mining* guide.

Cichosz, P. (2014). *Data Mining Algorithms: Explained Using R*. John Wiley & Sons.

Conti, M., Das, S. K., Bisdikian, C., Kumar, M., Ni, L. M., Passarella, A., … Zambonelli, F. (2012). Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber--physical convergence. *Pervasive and Mobile Computing*, *8*(1), 2–21.

Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). *Distributed systems: concepts and design*. pearson education.

Denning, D. E. (1976). A lattice model of secure information flow. *Communications of the ACM*, *19*(5), 236–243.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., & others. (1996). Knowledge Discovery and *Data Mining*: Towards a Unifying Framework. In *KDD* (Vol. 96, pp. 82–88).

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From *data mining* to knowledge discovery in databases. *AI Magazine*, *17*(3), 37.

Forman, G. H., & Zahorjan, J. (1994). The challenges of mobile computing. *Computer*, *27*(4), 38–47.

Giraud-Carrier, C., & Povel, O. (2003). Characterising *data mining* software. *Intelligent Data Analysis*, *7*(3), 181–192.

Hand, D. J., Mannila, H., & Smyth, P. (2001). *Principles of data mining*. MIT press.

Huang, T. C.-K., Liu, C.-C., & Chang, D.-C. (2012). An empirical investigation of factors influencing the adoption of *data mining* tools. *International Journal of Information Management*, *32*(3), 257–270. http://doi.org/http://dx.doi.org/10.1016/j.ijinfomgt.2011.11.006

Kantardzic, M. (2011). *Data-Mining* Concepts. *Data Mining: Concepts, Models, Methods, and Algorithms, Second Edition*, 1–25.

KDnuggets. (2014a). *CRISP-DM*, stil the top methodology for analytics, *data mining* , or *data* science projects. Retrieved February 14, 2015, from http://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-*data-mining-data*-science-projects.html

KDnuggets. (2014b, December 10). What Analytics, *Data Mining*, *Data* Science software/tools you used in the past 12 months for a real project Poll. Retrieved December 10, 2014, from http://www.kdnuggets.com/polls/2014/analytics-*data-mining-data*-science-software-used.html

Krumm, J. (2009). *Ubiquitous computing fundamentals*. CRC Press.

Lyytinen, K., & Yoo, Y. (2002). Issues and challenges in Ubiquitous computing. *Communications of the ACM*, *45*(12), 63–96.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), 251–266.

Mark, W. (1999). Turning pervasive computing into mediated spaces. *IBM Systems Journal*, *38*(4), 677–692.

Mikut, R., & Reischl, M. (2011). *Data mining* tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *1*(5), 431–443.

Nieuwdorp, E. (2007). The pervasive discourse: an analysis. *Computers in Entertainment (CIE)*, *5*(2), 13.

Pfleeger, C. P., & Pfleeger, S. L. (2006). Security in Computing.

Saha, D., & Mukherjee, A. (2003). Pervasive computing: a paradigm for the 21st century. *Computer*, *36*(3), 25–31.

Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, *8*(4), 10–17.

Vaishnavi, V. K., & Kuechler Jr, W. (2007). *Design science research methods and patterns: innovating information and communication technology*. CRC Press.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 94–104.

Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM*, *36*(7), 75–84.

Witten, I. H., Frank, E., & Mark, A. (2011). *Data Mining*: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco. Retrieved.

Ye, J., Dobson, S., & Nixon, P. (2008). An overview of pervasive computing systems. In *Ambient Intelligence with Microsystems* (pp. 3–17). Springer.

Zhao, R., & Wang, J. (2011). Visualizing the research on pervasive and ubiquitous computing. *Scientometrics*, *86*(3), 593–612.